

Oracle TopLink

Developer's Guide, Volume 5

10g Release 3 (10.1.3)

B13593-02

February 2006

Using TopLink in Your Application

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, and PeopleSoft are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software—Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Retek are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Contents

Preface	li

Part I Building a TopLink Application

1 Understanding TopLink

What is TopLink?	1-1
Solving the Object-Persistence Impedance Mismatch	1-2
TopLink Key Features.....	1-4
TopLink Application Architectures	1-4

2 Understanding TopLink Application Development

Developing Your Application With TopLink	2-1
Typical Development Stages	2-1
Oracle Development Support.....	2-3
Designing Your Application With TopLink	2-3
Understanding TopLink Usage.....	2-3
Relational Database Usage	2-4
Object-Relational Database Usage	2-4
Oracle XML Database (XDB) Usage	2-4
Enterprise Information System (EIS) Usage.....	2-4
XML Usage.....	2-4
Understanding Target Platforms	2-4
Selecting an Architecture With TopLink	2-5
Tiers.....	2-5
Three Tier	2-5
Two Tier	2-6
Service Layer.....	2-6
EJB Session Beans.....	2-7
EJB Entity Beans	2-7
Plain Old Java Objects (POJO)	2-8
Data Access	2-8
Data Type	2-8
Multiple Data Sources	2-8
Isolating Data Access.....	2-8
Historical Data Access.....	2-9

Caching	2-9
Cache Type.....	2-9
Refreshing	2-9
Cache Coordination	2-9
Locking	2-10
Optimistic Locking	2-10
Pessimistic Locking.....	2-10
Building and Using the Persistence Layer	2-11
Implementation Options	2-11
Persistent Class Requirements	2-11
Persistence Layer Components	2-12
Mapping Metadata	2-12
Session Metadata.....	2-12
Cache.....	2-12
Queries and Expressions.....	2-13
Transactions	2-13
Using the Persistence Layer	2-13
Deploying the Application.....	2-14
Understanding Deployments	2-14
TopLink in a J2EE Application.....	2-14
Optimizing and Customizing the Application.....	2-15
Troubleshooting the Application	2-15
Understanding Object Persistence.....	2-15
Application Object Model.....	2-16
Data Storage Schema	2-16
Primary Keys and Object Identity	2-16
Mappings.....	2-16
Foreign Keys and Object Relationships	2-17
Inheritance.....	2-17
Concurrency.....	2-17
Caching	2-18
Nonintrusive Persistence	2-18
Indirection	2-18
Understanding TopLink Metadata.....	2-19
Advantages of the TopLink Metadata Architecture	2-19
Creating Project Metadata	2-20
Descriptors and Mappings	2-20
Data Source Login Information	2-20
Creating Session Metadata	2-21
Deploying Metadata	2-21
Understanding the Three-Tier Architecture.....	2-21
Example Implementations	2-22
Advantages and Disadvantages	2-22
Variation Using Remote Sessions	2-22
Technical Challenges	2-22
Understanding the Two-Tier Architecture	2-23
Example Implementations	2-24

Advantages and Disadvantages	2-24
Technical Challenges	2-24
Understanding the EJB Session Bean Facade Architecture.....	2-24
Example Implementation.....	2-25
Advantages and Disadvantages	2-25
Understanding Session Beans	2-25
Technical Challenges	2-26
Unit of Work Merge.....	2-26
Understanding the EJB Entity Beans with CMP Architecture	2-27
Example Implementation.....	2-28
Advantages and Disadvantages	2-28
Technical Challenges	2-28
External JDBC Pools	2-29
JTA/JTS Integration.....	2-29
Cache Coordination.....	2-29
Maintaining Bidirectional Relationships.....	2-29
Managing Dependent Objects in EJB 1.1	2-30
Managing Dependent Objects in EJB 2.0.....	2-32
Managing Collections of EJBObjects in EJB 1.1	2-32
Managing Collections of EJBObjects in EJB 2.0	2-33
Understanding the EJB Entity Beans With BMP Architecture	2-33
Example Implementations.....	2-35
Advantages and Disadvantages	2-35
Technical Challenges	2-35
External JDBC Pools	2-35
JTA/JTS Integration.....	2-36
Cache Coordination.....	2-36
Understanding the Web Services Architecture.....	2-36
Example Implementations.....	2-36
Advantages and Disadvantages	2-36
Technical Challenges	2-37

Part II Using TopLink Development Tools

3 Understanding TopLink Development Tools

Development Environment.....	3-2
TopLink Run-Time Environment.....	3-2

4 Using TopLink Workbench

Understanding TopLink Workbench.....	4-1
Configuring the TopLink Workbench Environment.....	4-2
Working With TopLink Workbench	4-3
Using the Menus	4-5
Menu Bar Menus.....	4-5
Context Menus	4-5
Using the Toolbars	4-6

Standard Toolbar	4-6
Context Toolbar	4-7
Using the Navigator.....	4-9
Using the Editor	4-10
Using the Problems Window	4-11
Using the Online Help.....	4-12
Working With TopLink Workbench Preferences	4-12
General Preferences	4-13
Help Preferences	4-14
Mappings Preferences	4-15
Class Preferences	4-16
EJB Preferences	4-16
Database Preferences	4-17
Sessions Configuration Preferences	4-18
New Names Preferences	4-19
Platform Preferences.....	4-20
Working With Databases	4-21
Working With Database Tables in the Navigator Window	4-21
Logging In and Out of a Database.....	4-21
Creating New Tables	4-22
Importing Tables From a Database	4-22
Removing Tables.....	4-24
Renaming Tables.....	4-24
Refreshing Tables From the Database.....	4-24
Working With Database Tables in the Editor Window	4-25
Working With Column Properties	4-25
Setting a Primary Key for Database Tables.....	4-26
Creating Table References	4-27
Creating Field Associations.....	4-28
Generating Data From Database Tables	4-29
Generating SQL Creation Scripts.....	4-29
Generating Classes and Descriptors From Database Tables	4-30
Generating EJB Entities and Descriptors From Database Tables.....	4-31
Generating Tables on the Database.....	4-32
Working With XML Schemas	4-33
Working With XML Schemas in the Navigator.....	4-33
Working With XML Schema Structure	4-34
Importing an XML Schema.....	4-34
Configuring XML Schema Reference	4-36
Using TopLink Workbench	4-36
Using Java	4-37
Configuring XML Schema Namespace.....	4-38
Using TopLink Workbench	4-38
Using Java	4-39
Working With Classes	4-40
Creating Classes	4-40
Using TopLink Workbench	4-40

Configuring Classes	4-41
Configuring Class Information	4-41
Configuring Class Modifiers	4-42
Configuring Class Interfaces	4-43
Adding Attributes.....	4-44
Configuring Attribute Modifiers	4-44
Configuring Attribute Type Information	4-45
Configuring Attribute Accessing Methods	4-46
Adding Methods	4-47
Configuring Method Modifiers	4-48
Configuring Method Type Information	4-48
Configuring Method Parameters.....	4-49
Importing and Updating Classes.....	4-50
Using TopLink Workbench	4-50
Managing Nondesoriptor Classes.....	4-51
Renaming Packages	4-52
Using TopLink Workbench	4-52
Integrating TopLink Workbench With Apache Ant	4-53
Configuring Ant to Use TopLink Workbench Tasks	4-53
Library Dependencies	4-53
Declaring TopLink Workbench Tasks	4-54
Understanding TopLink Workbench Ant Task API.....	4-54
Creating TopLink Workbench Ant Tasks.....	4-55
mappings.validate.....	4-56
Parameters.....	4-57
Parameters Specified as Nested Elements.....	4-57
Examples	4-57
session.validate.....	4-57
Parameters.....	4-58
Parameters Specified as Nested Elements.....	4-58
Examples	4-58
mappings.export	4-58
Parameters.....	4-58
Parameters Specified as Nested Elements.....	4-59
Examples	4-59
classpath	4-59
Parameters.....	4-60
Parameters Specified as Nested Elements.....	4-60
Examples	4-60
ignoreerror	4-60
Parameters.....	4-60
Parameters Specified as Nested Elements.....	4-60
Examples	4-61
ignoreerrorset.....	4-61
Parameters.....	4-61
Parameters Specified as Nested Elements.....	4-61
Examples	4-61

loginspec.....	4-62
Parameters.....	4-62
Parameters Specified as Nested Elements.....	4-62
Examples	4-62

5 Using an Integrated Development Environment

Configuring TopLink for Oracle JDeveloper.....	5-1
Using TopLink Mappings.....	5-1
Using TopLink Sessions	5-3
Configuring TopLink Workbench With Source Control Management Software.....	5-3
Using a Source Control Management System.....	5-3
Merging Files	5-4
Merging Project Files.....	5-4
Merging Table, Descriptor, and Class Files	5-5
Sharing Project Objects.....	5-6
Managing the ejb-jar.xml File.....	5-6
Working With Locked Files	5-6

6 Using the Schema Manager

Understanding the Schema Manager	6-1
Schema Manager Java and Database Type Conversion	6-3
Sequencing	6-3
Creating a Table Creator.....	6-4
Using TopLink Workbench During Development.....	6-4
Using the Default Table Generator at Run Time	6-4
Using Java.....	6-4
Creating a TableCreator Class.....	6-5
Creating a TableDefinition Class	6-5
Adding Fields to a TableDefinition.....	6-5
Defining Sybase and Microsoft SQL Server Native Sequencing.....	6-6
Creating Tables With a Table Creator.....	6-6
Automatic Database Table Creation	6-6

Part III Deploying a TopLink Application

7 Integrating TopLink With an Application Server

Application Server Support	7-1
Application Server Integration Concepts	7-2
Software Requirements	7-2
XML Parser Platform Configuration.....	7-2
Configuring XML Parser Platform	7-3
Creating an XML Parser Platform	7-3
XML Parser Limitations	7-3
Security Permissions.....	7-4
Persistence Manager Migration	7-4
Clustering	7-4

Oracle Containers for J2EE (OC4J)	7-5
CMP Integration.....	7-5
Migrating OC4J Orion Persistence to OC4J TopLink Persistence.....	7-5
Overview.....	7-6
Using the TopLink Migration Tool from TopLink Workbench.....	7-9
Using the TopLink Migration Tool From the Command Line.....	7-10
Post-Migration Changes.....	7-12
Troubleshooting Your Migration.....	7-13
JTA Integration.....	7-14
BEA WebLogic Server	7-14
Classpath.....	7-15
CMP Integration.....	7-15
Migrating BEA WebLogic Persistence to OC4J TopLink Persistence.....	7-16
Overview.....	7-16
Using the TopLink Migration Tool From TopLink Workbench.....	7-18
Using the TopLink Migration Tool From the Command Line.....	7-18
JTA Integration.....	7-20
Security Manager.....	7-20
IBM WebSphere Application Server	7-21
Classpath.....	7-21
Configuring Classpath for IBM WebSphere Application Server 4.0.....	7-21
Configuring Classpath for IBM WebSphere Application Server 5.0 and Later.....	7-21
CMP Integration.....	7-22
JTA Integration.....	7-22
Clustering on IBM WebSphere Application Server.....	7-22
Understanding Security Permissions	7-22
Permissions Required by TopLink Features.....	7-23
System Properties.....	7-23
Loading project.xml or sessions.xml Files.....	7-23
Cache Coordination.....	7-23
Accessing a Data Source by Port.....	7-24
Logging With java.util.logging.....	7-24
J2EE Application Deployment.....	7-24
Permissions Required When doPrivileged is Disabled.....	7-24
Disabling doPrivileged Operation.....	7-25
Configuring Miscellaneous EJB Options	7-25
Setter Parameter Type Checking.....	7-25
Unknown Primary Key Class Support.....	7-25
Single-Object Finder Return Type Checking.....	7-26

8 Creating TopLink Files for Deployment

Understanding TopLink Deployment File Creation	8-1
project.xml File.....	8-2
XSD File Format.....	8-2
Non-CMP Applications and Project Metadata.....	8-2
CMP Applications and Project Metadata.....	8-3
Creating project.xml With TopLink Workbench.....	8-3

Creating project.xml Programatically	8-3
EJB 3.0 and the project.xml File	8-3
sessions.xml File	8-3
XSD File Format	8-4
Non-CMP Applications and Session Metadata	8-4
CMP Applications and Session Metadata	8-4
EJB 3.0 and the sessions.xml File	8-5
ejb-jar.xml File	8-5
EJB 3.0 and the ejb-jar.xml File	8-5
<J2EE-Container>-ejb-jar.xml File	8-5
OC4J and the orion-ejb-jar.xml File	8-5
BEA WebLogic Server and the weblogic-ejb-jar.xml File	8-6
EJB 3.0 and the <J2EE-Container>-jar.xml File	8-6
toplink-ejb-jar.xml File	8-6
OC4J and the toplink-ejb-jar.xml File	8-6
BEA WebLogic Server and the toplink-ejb-jar.xml File	8-7
IBM WebSphere Application Server and the toplink-ejb-jar.xml File	8-8
EJB 3.0 and the toplink-ejb-jar.xml File	8-8
Java Applications	8-8
JavaServer Pages and Servlet Applications	8-8
Session Bean Applications	8-8
CMP Applications	8-9
BMP Applications	8-9
Configuring the orion-ejb-jar.xml File for OC4J	8-9
Configuring persistence-manager Entries	8-9
Configuring pm-properties	8-10
Configuring cache-synchronization Properties	8-11
Configuring default-mapping Properties	8-12
Configuring the weblogic-ejb-jar.xml File for BEA WebLogic Server	8-13
Configuring persistence-descriptor Entries	8-13
Unsupported weblogic-ejb-jar.xml File Tags	8-14

9 Packaging a TopLink Application

Java Applications	9-1
JavaServer Pages and Servlet Applications	9-2
TopLink Domain JAR	9-2
Session Bean Applications	9-3
TopLink Domain JAR	9-3
EJB JAR	9-4
CMP Applications	9-4
EJB JAR	9-4
BMP Applications	9-5
TopLink Domain JAR	9-6
EJB JAR	9-7
Packaging With TopLink Metadata File Resource Paths	9-7

10 Deploying a TopLink Application

Java Applications	10-1
JavaServer Pages and Servlets	10-1
Session Bean Applications	10-1
CMP Applications	10-2
Deploying a CMP Application to OC4J	10-2
Deploying a CMP Application to BEA WebLogic Server	10-2
Troubleshooting ejbc	10-3
Deploying a CMP Application to IBM WebSphere Application Server 4.0	10-3
Starting the Entity Bean	10-3
BMP Applications	10-4
Hot Deployment of EJB	10-4
Hot Deployment in a CMP Application	10-4
Hot Deployment in a non-CMP Application	10-4
Using the WebSphere Deploy Tool	10-5
Using the Deploy Tool on its Own	10-5
Using the Deploy Tool With WebSphere Studio Application Developer	10-5
Troubleshooting	10-6

Part IV Optimizing and Customizing a TopLink Application

11 Optimization

Understanding Optimization	11-1
Sources of Application Performance Problems	11-2
Measuring Performance With the TopLink Profiler	11-2
Configuring the TopLink Performance Profiler	11-3
Accessing the TopLink Profiler Results	11-3
Measuring Performance With the Oracle Dynamic Monitoring System (DMS)	11-4
Configuring the Oracle DMS Profiler	11-6
OC4J Applications	11-7
Non-OC4J Applications	11-7
Accessing Oracle DMS Profiler Data Using JMX	11-7
Accessing Oracle DMS Profiler Data Using the DMS Spy Servlet.....	11-7
General Performance Optimization	11-8
Schema Optimization	11-8
Schema Case 1: Aggregation of Two Tables into One	11-9
Schema Case 2: Splitting One Table Into Many	11-9
Schema Case 3: Collapsed Hierarchy	11-11
Schema Case 4: Choosing One out of Many	11-12
Mapping and Descriptor Optimization	11-13
Session Optimization	11-13
Cache Optimization	11-13
Data Access Optimization	11-14
JDBC Driver Properties Optimization.....	11-14
Data Format Optimization.....	11-15
Batch Writing	11-15

Parameterized SQL (Binding) and Prepared Statement Caching	11-15
Query Optimization	11-16
Parameterized SQL and Prepared Statement Caching.....	11-17
Named Queries.....	11-17
Batch and Join Reading	11-17
Partial Object Queries and Fetch Groups	11-17
JDBC Fetch Size	11-17
Cursored Streams and Scrollable Cursors.....	11-18
Read Optimization Examples.....	11-19
Reading Case 1: Displaying Names in a List	11-20
Reading Case 2: Batch Reading Objects.....	11-22
Reading Case 3: Using Complex Custom SQL Queries	11-24
Reading Case 4: Using View Objects	11-24
Reading Case 5: Inheritance Views	11-26
Write Optimization Examples.....	11-26
Writing Case: Batch Writes.....	11-27
Unit of Work Optimization	11-30
Application Server and Database Optimization	11-30

12 Customization

Overview	12-1
Creating Custom Data Types	12-1
Using Public Source.....	12-2

Part V Troubleshooting a TopLink Application

13 TopLink Exception Reference

Descriptor Exceptions (1 – 200)	13-2
Concurrency Exceptions (2001 – 2006)	13-25
Conversion Exceptions (3001– 3008)	13-26
Database Exceptions (4002 – 4018).....	13-27
Optimistic Lock Exceptions (5001 – 5008)	13-29
Query Exceptions (6001 – 6121)	13-30
Validation Exceptions (7001 – 7147)	13-42
EJB QL Exceptions (8001 – 8010)	13-55
Session Loader Exceptions (9000 - 9010)	13-56
EJB Exception Factory Exceptions (10001 - 10069)	13-58
Communication Exceptions (12000 - 12003).....	13-64
XML Data Store Exceptions (13000 - 13020)	13-64
Deployment Exceptions (14001 - 14033)	13-67
Synchronization Exceptions (15001 - 15025)	13-70
SDK Data Store Exceptions (17001 - 17006)	13-71
EIS Exceptions (17007 – 17025)	13-72
JMS Processing Exceptions (18001 - 18002).....	13-74
SDK Descriptor Exceptions (19001 - 19003)	13-75
Default Mapping Exceptions (20001 - 20008)	13-75

Discovery Exceptions (22001 - 22004).....	13-77
Remote Command Manager Exceptions (22101 - 22111)	13-77
Transaction Exceptions (23001 - 23011)	13-79
XML Conversion Exceptions (25001 - 25016)	13-80
Migration Utility Exceptions (26001 - 26017).....	13-82
EJB JAR XML Exceptions (72000 – 72022)	13-84

14 TopLink Workbench Error Reference

Miscellaneous Errors (1 – 89).....	14-1
Project Errors (100 – 126)	14-2
Descriptor Errors (200 – 350)	14-4
Mapping Errors (400 – 483)	14-11
Table Errors (500 – 610)	14-15
XML Schema Errors (700 – 706)	14-20
Session Errors (800 – 812)	14-21
Common Classpath Problems.....	14-22
Data Source Problems	14-23
Database Connection Problems	14-23

15 Troubleshooting Application Deployment

Generating Deployment JAR Files	15-1
Common J2SE Deployment Exceptions.....	15-1
Classpath Exceptions	15-2
Communication Exceptions.....	15-2
Descriptor Validation Exceptions	15-2
Common BEA WebLogic Server Deployment Exceptions	15-3
Common BEA WebLogic Server 6.1 Exceptions	15-5
Development Exceptions	15-6
Deployment and Run Time Exceptions	15-6
Common BEA WebLogic 7.0 Exceptions	15-8
Development Exceptions	15-8
Deployment Exceptions	15-9
Common BEA WebLogic 8.1 Exceptions	15-10
Development Exceptions	15-10
Deployment Exceptions	15-11
Common IBM WebSphere Application Server Exceptions	15-12
Problems at Run Time	15-14
Common TopLink for IBM WebSphere Deploy Tool Exceptions.....	15-14

Part VI TopLink Tutorial

16 Planning the Tutorial Project

Overview	16-1
Before you Begin	16-1
Understanding the Tutorial Project.....	16-2
Source Files.....	16-2

Object Model Classes.....	16-2
Application Classes	16-3
Data Model.....	16-4
Creating and Populating Database Tables	16-7

17 Building the Tutorial Project

Step 1: Creating a Project	17-1
TopLink Workbench Project.....	17-1
Adding Java Classes	17-3
Step 2: Adding a Data Source.....	17-5
Creating a Database Development Login.....	17-5
Importing Database Tables	17-6
Project-Level Sequencing	17-8
Step 3: Associating Descriptors to Tables	17-9
Associating With Multiple Tables	17-10
Descriptor-Level Sequencing.....	17-11
Step 4: Mapping Attributes	17-12
Direct-to-Field Mappings.....	17-12
Using Automap	17-13
Additional Direct-to-Field Mappings	17-14
One-to-One Mappings.....	17-14
Additional One-to-One Mappings	17-16
One-to-Many Mappings.....	17-16
Additional One-to-Many Mappings	17-18
Object Type Mappings	17-18
Aggregate Object Mappings.....	17-19
Aggregate Descriptor	17-20
Aggregate Mapping.....	17-20
Direct Collection Mappings.....	17-21
Many-to-Many Mappings.....	17-21
Transformation Mappings.....	17-23
Step 5: Using Advanced Descriptor Properties	17-25
Inheritance.....	17-25
Root Parent Class	17-25
Child Class	17-26
Indicator Values	17-27
Locking Policy	17-28
Named Queries.....	17-29
Step 6: Verifying the Project	17-32

18 Deploying the Tutorial Project

Step 1: Generating Deployment Information	18-1
Creating the project.xml File	18-1
Step 2: Creating Sessions Configuration	18-3
Creating the sessions.xml File	18-3
Creating a Server Session.....	18-3
Associating the Project With the Session.....	18-4

Updating the data-sources.xml File.....	18-5
Creating a Database Deployment Login.....	18-6
Configuring Logging Options.....	18-7
Step 3: Packaging for Deployment.....	18-8
Domain JAR File.....	18-8
WAR File	18-8
EAR File.....	18-8
Step 4: Deploying to an Application Server.....	18-9
Step 5: Running the Tutorial Application.....	18-9

Part VII Mapping and Configuration Overview

19 Understanding TopLink Mapping and Configuration Concepts

Mapping and Configuration Concepts	19-1
Projects.....	19-1
Descriptors	19-2
Mappings.....	19-2

Part VIII Projects

20 Understanding Projects

TopLink Project Types	20-1
Project Concepts	20-2
Project Architecture	20-2
Relational and Nonrelational Projects.....	20-2
Persistent and Nonpersistent Projects.....	20-2
Projects and Login.....	20-3
Non-CMP Session Role: Session Login.....	20-3
CMP Deployment Role: Deployment Login	20-3
Development Role: Development Login	20-4
Projects and Platforms.....	20-4
Projects and Sequencing.....	20-4
Configuring how to Obtain Sequence Values.....	20-5
Configuring Where to Write Sequence Values.....	20-5
XML Namespaces.....	20-5
Relational Projects.....	20-6
Building Relational Projects for a Relational Database	20-6
Building Relational Projects for an Object-Relational Database	20-6
EIS Projects	20-7
Building EIS Projects with XML Records	20-9
Building EIS Projects With Indexed or Mapped Records	20-9
XML Projects	20-9
TopLink Support for Java Architecture for XML Binding (JAXB)	20-10
Understanding JAXB-Specific Generated Files	20-10
Understanding TopLink-Specific Generated Files.....	20-11
Using TopLink JAXB Compiler Generated Files at Run Time	20-13

JAXB Validation	20-14
Understanding the Project API	20-14
Project Inheritance Hierarchy.....	20-14
Understanding Sequencing in Relational Projects	20-14
Sequencing Configuration Options	20-15
Sequencing Types.....	20-16
Table Sequencing	20-16
Unary Table Sequencing	20-17
Query Sequencing.....	20-18
Default Sequencing.....	20-18
Native Sequencing With an Oracle Database Platform.....	20-18
Native Sequencing With a Non-Oracle Database Platform.....	20-19
Sequencing and Preallocation Size	20-20
Sequencing With CMP Entity Beans	20-21
Understanding XML Namespaces	20-22
TopLink Workbench Namespace Resolution	20-22
Element and Attribute Form Options	20-22
Element Form Default Qualified and Attribute Form Default Unqualified	20-23
Element and Attribute Form Default Unqualified.....	20-24
Element and Attribute Form Default Qualified	20-25
TopLink Runtime Namespace Resolution.....	20-26

21 Creating a Project

Project Creation Overview	21-1
Using TopLink Workbench	21-2
Creating New TopLink Workbench Projects.....	21-2
Using Java.....	21-3
Creating a Project for an Existing Object and Data Model	21-5
Using TopLink Workbench	21-5
Creating a Project From an Existing Object Model	21-5
Using TopLink Workbench	21-5
Creating a Project From an Existing Data Model	21-5
Using TopLink Workbench	21-6
Creating an XML Project From an XML Schema	21-6
Using TopLink Workbench	21-6
Using the Command Line	21-8
Creating a Project by Migrating an EAR to OC4J	21-9
Creating a Project From an OC4J EJB 2.0 CMP EAR at Deployment Time	21-10
Working With Projects	21-10
Opening Existing Projects	21-10
Saving Projects.....	21-11
Saving Projects With a New Name or Location	21-12
Generating the Project Status Report	21-12
Exporting Project Information	21-13
Exporting Deployment XML Information.....	21-14
Exporting Model Java Source.....	21-14
Exporting Project Java Source	21-14

Exporting Table Creator Files.....	21-15
Working With the ejb-jar.xml File.....	21-15
Writing to the ejb-jar.xml File.....	21-16
Reading From the ejb-jar.xml File.....	21-16

22 Configuring a Project

Configuring Common Project Options.....	22-1
Configuring Project Save Location	22-2
Using TopLink Workbench	22-2
Configuring Project Classpath.....	22-3
Using TopLink Workbench	22-3
Configuring Mapped Field Access at the Project Level.....	22-4
Using TopLink Workbench	22-4
Configuring Persistence Type.....	22-5
Using TopLink Workbench	22-6
Configuring Default Descriptor Advanced Properties.....	22-7
Using TopLink Workbench	22-8
Configuring Existence Checking at the Project Level.....	22-8
Using TopLink Workbench	22-9
Configuring Project Deployment XML Options.....	22-10
Using TopLink Workbench	22-10
Configuring Model Java Source Code Options.....	22-11
Using TopLink Workbench	22-11
Configuring Deprecated Direct Mappings	22-12
Using TopLink Workbench	22-13
Configuring Cache Type and Size at the Project Level.....	22-13
Using TopLink Workbench	22-14
Configuring Cache Isolation at the Project Level	22-16
Using TopLink Workbench	22-16
Configuring Cache Coordination Change Propagation at the Project Level.....	22-17
Using TopLink Workbench	22-18
Configuring Cache Expiration at the Project Level	22-19
Using TopLink Workbench	22-20
Configuring Project Comments.....	22-21
Using TopLink Workbench	22-21

23 Configuring a Relational Project

Relational Project Configuration Overview	23-1
Configuring Relational Database Platform at the Project Level.....	23-2
Using TopLink Workbench	23-2
Configuring Sequencing at the Project Level.....	23-3
Using TopLink Workbench	23-3
Using Java.....	23-4
Configuring Login Information	23-5
Using TopLink Workbench	23-5
Configuring Development and Deployment Logins	23-6

Using TopLink Workbench	23-6
Logging in to the Database	23-7
Configuring Named Query Parameterized SQL and Statement Caching at the Project Level.....	23-7
Using TopLink Workbench	23-8
Configuring Table Generation Options.....	23-9
Using TopLink Workbench	23-9
Configuring Table Creator Java Source Options.....	23-10
Using TopLink Workbench	23-10
Configuring Project Java Source Code Options.....	23-11
Using TopLink Workbench	23-11

24 Configuring an EIS Project

EIS Project Configuration Overview.....	24-1
Configuring EIS Data Source Platform at the Project Level.....	24-2
Using TopLink Workbench	24-2
Configuring EIS Connection Specification Options at the Project Level.....	24-2
Using TopLink Workbench	24-3

25 Configuring an XML Project

XML Project Configuration Overview	25-1

Part IX Descriptors

26 Understanding Descriptors

Descriptor Types	26-1
Descriptor Concepts.....	26-2
Descriptor Architecture.....	26-2
Descriptors and Inheritance.....	26-3
Descriptors and EJB	26-3
Nondeferred Changes	26-3
Creating a New Entity Bean and ejbCreate / ejbPostCreate Methods	26-4
Inheritance	26-4
Fetch Groups	26-4
Amendment and After-Load Methods	26-5
Descriptors and Aggregation	26-5
Aggregate and Composite Descriptors in Relational Projects	26-5
Root and Composite Descriptors in EIS Projects.....	26-8
Composite Descriptors in XML Projects.....	26-8
Descriptor Event Manager.....	26-8
Descriptor Query Manager.....	26-8
Descriptors and Sequencing	26-9
Descriptors and Locking	26-9
Default Root Element.....	26-9
Relational Descriptors.....	26-11
Object-Relational Descriptors	26-11

EIS Descriptors	26-11
XML Descriptors.....	26-12
Understanding Descriptors and Inheritance	26-12
Specifying a Class Indicator.....	26-13
Using Class Indicator Fields.....	26-13
Using Class Extraction Methods.....	26-14
Inheritance and Primary Keys (Relational and EIS Only).....	26-15
Single and Multi-Table Inheritance (Relational Only).....	26-15
Single Table Inheritance.....	26-15
Multitable Inheritance.....	26-16
Aggregate and Composite Descriptors and Inheritance	26-17
Inheritance and EJB.....	26-17
Understanding Descriptors and Locking	26-17
Optimistic Version Locking Policies	26-17
Optimistic Version Locking Policies and Cascading	26-18
Optimistic Locking and Rollbacks.....	26-20
Optimistic Field Locking Policies	26-20
Pessimistic Locking Policy.....	26-21
Locking in a Three-Tier Application	26-21
Optimistic Locking in a Three-Tier Application	26-21
Pessimistic Locking in a Three-Tier Application	26-22
Understanding the Descriptor API	26-22
Descriptor Inheritance Hierarchy	26-23

27 Creating a Descriptor

Descriptor Creation Overview	27-1
Creating a Relational Descriptor.....	27-1
Using TopLink Workbench	27-2
Relational Class Descriptors.....	27-2
Relational Aggregate Descriptors.....	27-2
Relational Interface Descriptors.....	27-2
Using Java.....	27-2
Creating an Object-Relational Descriptor.....	27-3
Using Java.....	27-3
Creating an EIS Descriptor.....	27-4
Using TopLink Workbench	27-4
EIS Root Descriptors.....	27-5
EIS Composite Descriptors	27-5
Using Java.....	27-5
Creating an XML Descriptor	27-5
Using TopLink Workbench	27-5
Using Java.....	27-5
Validating Descriptors.....	27-6
Generating Java Code for Descriptors	27-6

28 Configuring a Descriptor

Configuring Common Descriptor Options	28-1
Configuring Primary Keys	28-2
Using TopLink Workbench	28-3
Using Java.....	28-4
Relational Projects.....	28-4
EIS Projects.....	28-4
Configuring Read-Only Descriptors	28-4
Using Read-Only Entity Beans.....	28-5
Using TopLink Workbench	28-5
Using Java.....	28-6
Configuring Unit of Work Conforming at the Descriptor Level	28-6
Using TopLink Workbench	28-7
Using Java.....	28-7
Configuring Descriptor Alias	28-7
Using TopLink Workbench	28-8
Using Java.....	28-9
Configuring Descriptor Comments	28-9
Using TopLink Workbench	28-9
Configuring Named Queries at the Descriptor Level	28-10
Using TopLink Workbench	28-10
Adding Named Queries.....	28-12
Configuring Named Query Type and Parameters	28-13
Configuring Named Query Selection Criteria.....	28-14
Configuring Read All Query Order	28-15
Configuring Named Query Optimization.....	28-16
Configuring Named Query Attributes	28-17
Configuring Named Query Group/Order Options	28-19
Creating an EIS Interaction for a Named Query	28-20
Configuring Named Query Options.....	28-22
Configuring Named Query Advanced Options.....	28-24
Using Java.....	28-25
Configuring Query Timeout at the Descriptor Level	28-26
Using TopLink Workbench	28-26
Using Java.....	28-27
Configuring Cache Refreshing	28-27
Using TopLink Workbench	28-28
Using Java.....	28-29
Configuring Query Keys.....	28-29
Using TopLink Workbench	28-31
Using Java.....	28-31
Configuring Interface Query Keys	28-33
Using TopLink Workbench	28-34
Using Java.....	28-34
Configuring Cache Type and Size at the Descriptor Level	28-35
Using TopLink Workbench	28-35
Using Java.....	28-37

Configuring Cache Isolation at the Descriptor Level	28-37
Using TopLink Workbench	28-37
Using Java.....	28-38
Configuring Cache Coordination Change Propagation at the Descriptor Level	28-38
Using TopLink Workbench	28-39
Using Java.....	28-40
Configuring Cache Expiration at the Descriptor Level.....	28-40
Using TopLink Workbench	28-41
Using Java.....	28-42
Configuring Cache Existence Checking at the Descriptor Level	28-42
Using TopLink Workbench	28-42
Using Java.....	28-43
Configuring a Descriptor With EJB Information	28-44
Using TopLink Workbench	28-44
Using Java.....	28-46
Configuring CMP Information	28-46
Configuring BMP Information.....	28-47
Configuring Reading Subclasses on Queries	28-47
Using TopLink Workbench	28-48
Using Java.....	28-48
Configuring Inheritance for a Child (Branch or Leaf) Class Descriptor	28-49
Using TopLink Workbench	28-49
Using Java.....	28-50
Configuring Inheritance for a Parent (Root) Descriptor.....	28-50
Using TopLink Workbench	28-51
Using Java.....	28-52
Configuring Inheritance Expressions for a Parent (Root) Class Descriptor	28-53
Using Java.....	28-55
Configuring Inherited Attribute Mapping in a Subclass.....	28-56
Using TopLink Workbench	28-56
Using Java.....	28-57
Configuring a Domain Object Method as an Event Handler.....	28-57
Using TopLink Workbench	28-58
Using Java.....	28-60
Configuring a Descriptor Event Listener as an Event Handler.....	28-60
Using Java.....	28-62
Configuring Locking Policy	28-62
Using TopLink Workbench	28-63
Using Java.....	28-65
Configuring an Optimistic Locking Policy	28-65
Configuring Optimistic Locking Policy Cascading	28-65
Configuring a Pessimistic Locking Policy	28-65
Configuring Returning Policy	28-65
Using TopLink Workbench	28-66
Using Java.....	28-67
Configuring Instantiation Policy	28-68
Using TopLink Workbench	28-68

Using Java.....	28-69
Configuring Copy Policy	28-69
Using TopLink Workbench	28-70
Using Java.....	28-70
Configuring Change Policy	28-70
Using Java.....	28-71
Configuring Deferred Change Detection Policy	28-71
Configuring Object Change Tracking Policy	28-71
Configuring Attribute Change Tracking Policy	28-72
Configuring a History Policy	28-73
Using Java.....	28-75
Configuring Write Responsibility	28-75
Configuring Wrapper Policy	28-75
Using Java.....	28-76
Configuring Fetch Groups.....	28-77
Using Java.....	28-78
Configuring Amendment Methods	28-78
Using TopLink Workbench	28-79

29 Configuring a Relational Descriptor

Relational Descriptor Configuration Overview.....	29-1
Configuring Associated Tables.....	29-2
Using TopLink Workbench	29-2
Using Java.....	29-3
Configuring Sequencing at the Descriptor Level	29-3
Using TopLink Workbench	29-4
Using Java.....	29-5
Configuring a Sequence by Name	29-5
Configuring the Same Sequence for Multiple Descriptors	29-5
Configuring the Platform Default Sequence	29-6
Configuring Custom SQL Queries for Basic Persistence Operations.....	29-6
Using TopLink Workbench	29-7
Using Java.....	29-8
Configuring Interface Alias	29-10
Using TopLink Workbench	29-11
Using Java.....	29-11
Configuring a Relational Descriptor as a Class or Aggregate Type	29-11
Using TopLink Workbench	29-12
Using Java.....	29-12
Configuring Multitable Information	29-13
Using TopLink Workbench	29-13
Using Java.....	29-14

30 Configuring an Object-Relational Descriptor

Object-Relational Descriptor Configuration Overview	30-1
Configuring Field Ordering	30-2
Using Java.....	30-2

31 Configuring an EIS Descriptor

EIS Descriptor Configuration Overview	31-1
Configuring Schema Context for an EIS Descriptor	31-2
Using TopLink Workbench	31-2
Choosing a Schema Context	31-3
Using Java	31-3
Configuring Default Root Element	31-3
Using TopLink Workbench	31-4
Choosing a Root Element	31-4
Using Java	31-5
Configuring Record Format	31-5
Using Java	31-6
Configuring Custom EIS Interactions for Basic Persistence Operations	31-6
Using TopLink Workbench	31-6
Using Java	31-8
Configuring an EIS Descriptor as a Root or Composite Type	31-8
Using TopLink Workbench	31-9
Using Java	31-9

32 Configuring an XML Descriptor

XML Descriptor Configuration Overview	32-1
Configuring Schema Context for an XML Descriptor	32-1
Using TopLink Workbench	32-2
Choosing a Schema Context	32-2
Using Java	32-3
Configuring for Complex Type of anyType	32-3
Using TopLink Workbench	32-4
Configuring Default Root Element	32-5
Using TopLink Workbench	32-5
Choosing a Root Element	32-5
Configuring Document Preservation	32-6
Using TopLink Workbench	32-6
Using Java	32-6

Part X Mappings

33 Understanding Mappings

Mapping Types	33-1
Mapping Concepts	33-2
Mapping Architecture	33-2
Example Mapping	33-3
Automatic Mappings	33-4
Automapping With TopLink Workbench at Development Time	33-4
Default Mapping in EJB 2.0 or 3.0 CMP Projects Using OC4J at Run Time	33-4
JAXB Project Generation at Development Time	33-5
Indirection	33-5

Value Holder Indirection.....	33-6
Transparent Indirect Container Indirection.....	33-8
Proxy Indirection	33-8
Indirection and EJB.....	33-9
Indirection, Serialization, and Detachment.....	33-9
Method Accessors and Attribute Accessors.....	33-10
Mapping Converters and Transformers	33-10
Serialized Object Converter.....	33-10
Type Conversion Converter	33-11
Object Type Converter	33-12
Simple Type Translator.....	33-12
Transformation Mappings.....	33-14
Mappings and XPath	33-15
XPath by Position.....	33-15
XPath by Path and Name.....	33-15
XPath by Name.....	33-16
Self XPath	33-16
Mappings and xsd:list and xsd:union Types	33-17
Mapping an xsd:union Type	33-17
Mapping an xsd:list Type	33-17
Mapping a List of Unions	33-18
Mapping a Union of Lists	33-19
Mapping a Union of Unions.....	33-19
Mappings and the jaxb:class Customization.....	33-20
all, choice, or sequence Structure.....	33-20
group Structure	33-21
sequence or choice Structure Containing a group	33-21
group Structure Containing a sequence or choice	33-22
group Structure Containing a group.....	33-23
Limitations of jaxb:class Customization Support.....	33-23
Mappings and JAXB Typesafe Enumerations.....	33-24
Understanding the Mapping API.....	33-25
Relational Mappings	33-25
Object-Relational Mappings.....	33-26
XML Mappings	33-27
EIS Mappings.....	33-27

34 Creating a Mapping

Mapping Creation Overview	34-1
Creating Mappings Manually During Development	34-1
Using TopLink Workbench	34-1
Creating Mappings Automatically During Development	34-2
Using TopLink Workbench	34-2
Creating Mappings Automatically During Deployment	34-2
Creating Mappings to Oracle LOB Database Objects.....	34-2
Using the Oracle JDBC OCI Driver or Server Driver.....	34-3
Using the Oracle JDBC Thin Driver	34-5

Removing Mappings	34-6
Using TopLink Workbench	34-6

35 Configuring a Mapping

Configuring Common Mapping Options.....	35-1
Configuring Read-Only Mappings.....	35-2
Using TopLink Workbench	35-3
Using Java.....	35-3
Configuring Indirection.....	35-3
Using TopLink Workbench	35-4
Using Java.....	35-5
Configuring ValueHolder Indirection	35-6
Configuring ValueHolder Indirection With Method Accessing.....	35-7
Configuring ValueHolder Indirection With EJB 3.0 on OC4J	35-8
Configuring IndirectContainer Indirection	35-8
Configuring Proxy Indirection.....	35-9
Configuring XPath	35-10
Using TopLink Workbench	35-11
Choosing the XPath	35-12
Configuring a Default Null Value at the Mapping Level	35-12
Using TopLink Workbench	35-13
Using Java.....	35-13
Configuring Method Accessing.....	35-14
Using TopLink Workbench	35-14
Using Java.....	35-15
Configuring Private or Independent Relationships.....	35-16
Using TopLink Workbench	35-17
Using Java.....	35-17
Configuring Mapping Comments.....	35-18
Using TopLink Workbench	35-18
Configuring a Serialized Object Converter	35-18
Using TopLink Workbench	35-19
Using Java.....	35-19
Configuring a Type Conversion Converter.....	35-20
Using TopLink Workbench	35-20
Using Java.....	35-21
Configuring an Object Type Converter	35-22
Using TopLink Workbench	35-22
Using Java.....	35-23
Configuring a Simple Type Translator.....	35-23
Using TopLink Workbench	35-24
Using Java.....	35-24
Configuring a JAXB Typesafe Enumeration Converter.....	35-25
Using Java.....	35-26
Configuring Container Policy.....	35-26
Using TopLink Workbench	35-27
Using Java.....	35-28

Configuring Attribute Transformer	35-29
Using TopLink Workbench	35-30
Using Java.....	35-31
Configuring Field Transformer Associations	35-31
Using TopLink Workbench	35-32
Specifying Field-to-Transformer Associations	35-32
Using Java.....	35-33
Configuring Mutable Mappings	35-33
Using TopLink Workbench	35-34
Using Java.....	35-34
Configuring Bidirectional Relationship	35-34
Using TopLink Workbench	35-35
Configuring the Use of a Single Node	35-36
Using TopLink Workbench	35-36
Using Java.....	35-37

Part XI Relational Mappings

36 Understanding Relational Mappings

Relational Mapping Types	36-1
Relational Mapping Concepts	36-2
Directionality	36-2
Converters and Transformers	36-3
Using a Direct Mapping.....	36-3
Using a Converter Mapping.....	36-3
Using a Transformation Mapping.....	36-3
Relational Mappings and EJB.....	36-3
Direct-to-Field Mapping	36-4
Direct to XMLType Mapping	36-4
One-to-One Mapping	36-5
One-to-One Mappings and EJB.....	36-6
Variable One-to-One Mapping	36-6
One-to-Many Mapping	36-7
One-to-Many Mappings and EJB.....	36-8
Many-to-Many Mapping	36-8
Many-to-Many Mappings and EJB.....	36-9
Aggregate Collection Mapping	36-10
Aggregate Collection Mappings and Inheritance	36-10
Aggregate Collection Mappings and EJB	36-11
Implementing Aggregate Collection Mappings.....	36-11
Direct Collection Mapping	36-11
Direct Map Mapping	36-12
Aggregate Object Mapping	36-12
Aggregate Object Mappings with a Single Source Object.....	36-13
Aggregate Object Mappings With Multiple Source Objects.....	36-14
Implementing an Aggregate Object Relationship Mapping.....	36-14
Transformation Mapping	36-15

37	Configuring a Relational Mapping	
	Configuring Common Relational Mapping Options.....	37-1
	Configuring a Database Field.....	37-2
	Using TopLink Workbench	37-4
	Configuring Reference Descriptor.....	37-5
	Using TopLink Workbench	37-5
	Configuring Batch Reading.....	37-6
	Using TopLink Workbench	37-7
	Using Java.....	37-7
	Configuring Query Key Order	37-8
	Using TopLink Workbench	37-8
	Configuring Table and Field References (Foreign and Target Foreign Keys).....	37-8
	Using TopLink Workbench	37-9
38	Configuring a Relational Direct-to-Field Mapping	
	Relational Direct-to-Field Mapping Configuration Overview	38-1
39	Configuring a Relational Direct-to-XMLType Mapping	
	Relational Direct-to-XMLType Mapping Overview.....	39-1
	Configuring Read Whole Document.....	39-1
	Using TopLink Workbench	39-1
40	Configuring a Relational One-to-One Mapping	
	Relational One-to-One Mapping Configuration Overview	40-1
	Configuring Joining at the Mapping Level.....	40-1
	Using TopLink Workbench	40-2
41	Configuring a Relational Variable One-to-One Mapping	
	Relational Variable One-to-One Mapping Configuration Overview	41-1
	Configuring Class Indicator.....	41-1
	Using TopLink Workbench	41-2
	Configuring Unique Primary Key	41-3
	Understanding Unique Primary Key	41-3
	Using TopLink Workbench	41-3
	Using Java.....	41-4
	Configuring Query Key Association.....	41-4
	Using TopLink Workbench	41-4
42	Configuring a Relational One-to-Many Mapping	
	Relational One-to-Many Mapping Configuration Overview	42-1
43	Configuring a Relational Many-to-Many Mapping	
	Relational Many-to-Many Mapping Configuration Overview.....	43-1
	Configuring a Relation Table.....	43-1

Using TopLink Workbench	43-2
44 Configuring a Relational Aggregate Collection Mapping	
Relational Aggregate Collection Mapping Configuration Overview.....	44-1
45 Configuring a Relational Direct Collection Mapping	
Relational Direct Collection Mapping Configuration Overview	45-1
Configuring Target Table	45-1
Using TopLink Workbench	45-2
Configuring Direct Value Field	45-2
Using TopLink Workbench	45-2
46 Configuring a Relational Direct Map Mapping	
Relational Direct Map Mapping Configuration Overview	46-1
Configuring Direct Value Field	46-1
Using TopLink Workbench	46-1
Configuring Direct Key Field	46-2
Using TopLink Workbench	46-2
Configuring Key Converters.....	46-3
Using TopLink Workbench	46-3
Configuring Value Converters.....	46-4
Using TopLink Workbench	46-4
47 Configuring a Relational Aggregate Object Mapping	
Relational Aggregate Object Mapping Configuration Overview	47-1
Configuring Aggregate Fields	47-1
Using TopLink Workbench	47-2
Configuring Allowing Null Values	47-2
Using TopLink Workbench	47-2
48 Configuring a Relational Transformation Mapping	
Relational Transformation Mapping Configuration Overview.....	48-1
Part XII Object-Relational Mappings	
49 Understanding Object-Relational Mappings	
Object-Relational Mapping Types.....	49-1
Object-Relational Structure Mapping.....	49-2
Object-Relational Reference Mapping	49-2
Object-Relational Array Mapping	49-2
Object-Relational Object Array Mapping.....	49-2
Object-Relational Nested Table Mapping.....	49-3

50	Configuring an Object-Relational Mapping	
	Configuring Common Object-Relational Mapping Options	50-1
	Configuring Reference Class	50-2
	Using Java.....	50-2
	Configuring Attribute Name	50-2
	Using Java.....	50-3
	Configuring Field Name	50-3
	Using Java.....	50-3
	Configuring Structure Name	50-4
	Using Java.....	50-4
51	Configuring an Object-Relational Structure Mapping	
	Object-Relational Structure Mapping Configuration Overview.....	51-1
52	Configuring an Object-Relational Reference Mapping	
	Object-Relational Reference Mapping Configuration Overview	52-1
53	Configuring an Object-Relational Array Mapping	
	Object-Relational Array Mapping Configuration Overview	53-1
54	Configuring an Object-Relational Object Array Mapping	
	Object-Relational Object Array Mapping Configuration Overview.....	54-1
55	Configuring an Object-Relational Nested Table Mapping	
	Object-Relational Nested Table Mapping Configuration Overview.....	55-1
Part XIII EIS Mappings		
56	Understanding EIS Mappings	
	EIS Mapping Types.....	56-1
	EIS Mapping Concepts.....	56-2
	EIS Record Type	56-2
	Indexed Records.....	56-2
	Mapped Records	56-3
	XML Records	56-3
	XPath Support.....	56-3
	xsd:list and xsd:union Support	56-3
	jaxb:class Support.....	56-3
	Typesafe Enumeration Support	56-3
	Composite and Reference EIS Mappings	56-4
	Composite EIS Mappings	56-4
	Reference EIS Mappings	56-4
	EIS Mapping Architecture	56-5
	EIS Direct Mapping	56-5

EIS Composite Direct Collection Mapping.....	56-6
EIS Composite Object Mapping.....	56-7
EIS Composite Collection Mapping.....	56-7
EIS One-to-One Mapping.....	56-8
EIS One-to-One Mappings With Key on Source.....	56-9
EIS One-to-One Mappings With Key on Target.....	56-10
EIS One-to-Many Mapping.....	56-12
EIS One-to-Many Mappings With Key on Source.....	56-13
EIS One-to-Many Mappings With Key on Target.....	56-15
EIS Transformation Mapping.....	56-17
 57 Configuring an EIS Mapping	
Configuring Common EIS Mapping Options.....	57-1
Configuring Reference Descriptors.....	57-2
Using TopLink Workbench.....	57-2
Configuring Selection Interaction.....	57-3
Using TopLink Workbench.....	57-4
 58 Configuring an EIS Direct Mapping	
EIS Direct Mapping Configuration Overview.....	58-1
 59 Configuring an EIS Composite Direct Collection Mapping	
EIS Composite Direct Collection Mapping Configuration Overview.....	59-1
 60 Configuring an EIS Composite Object Mapping	
EIS Composite Object Mapping Configuration Overview.....	60-1
 61 Configuring an EIS Composite Collection Mapping	
EIS Composite Collection Mapping Configuration Overview.....	61-1
 62 Configuring an EIS One-to-One Mapping	
EIS One-to-One Mapping Configuration Overview.....	62-1
Configuring Foreign Key Pairs.....	62-1
Using TopLink Workbench.....	62-2
 63 Configuring an EIS One-to-Many Mapping	
EIS One-to-Many Mapping Configuration Overview.....	63-1
Configuring Foreign Key Pairs.....	63-1
Using TopLink Workbench.....	63-2
Configuring Delete All Interactions.....	63-3
Using TopLink Workbench.....	63-4
 64 Configuring an EIS Transformation Mapping	
EIS Transformation Mapping Configuration Overview.....	64-1

Part XIV XML Mappings

65 Understanding XML Mappings

XML Mapping Types	65-1
XML Mapping Concepts	65-2
Mapping to Simple and Complex Types	65-2
Mapping Order	65-3
XPath Support	65-3
xsd:list and xsd:union Support	65-3
xs:any and xs:anyType Support	65-4
jaxb:class Support	65-4
Typesafe Enumeration Support	65-4
Mapping Extensions	65-4
XML Direct Mapping	65-5
Mapping to a Text Node	65-5
Mapping to a Simple Text Node	65-5
Mapping to a Text Node in a Simple Sequence	65-6
Mapping to a Text Node in a Subelement	65-6
Mapping to a Text Node by Position	65-7
Mapping to an Attribute	65-8
Mapping to a Specified Schema Type	65-9
Mapping to a List Field With an XML Direct Mapping	65-10
Mapping to a Union Field With an XML Direct Mapping	65-10
Mapping to a Union of Lists With an XML Direct Mapping	65-12
Mapping to a Union of Unions With an XML Direct Mapping	65-12
Mapping With a Simple Type Translator	65-13
XML Composite Direct Collection Mapping	65-14
Mapping to Multiple Text Nodes	65-15
Mapping to a Simple Sequence	65-15
Mapping to a Sequence in a Subelement	65-15
Mapping to Multiple Attributes	65-16
Mapping to a Single Text Node With an XML Composite Direct Collection Mapping	65-17
Mapping to a Single Attribute With an XML Composite Direct Collection Mapping	65-18
Mapping to a List of Unions With an XML Composite Direct Collection Mapping	65-18
Mapping to a Union of Lists With an XML Composite Direct Collection Mapping	65-19
Specifying the Content Type of a Collection With an XML Composite Direct Collection Mapping	65-20
XML Composite Object Mapping	65-21
Mapping Into the Parent Record	65-21
Mapping to an Element	65-22
Mapping to Different Elements by Element Name	65-23
Mapping to Different Elements by Element Position	65-24
XML Composite Collection Mapping	65-25
XML Any Object Mapping	65-27
XML Any Collection Mapping	65-29
XML Transformation Mapping	65-31

66	Configuring an XML Mapping	
	Configuring Common XML Mapping Options	66-1
	Configuring Reference Descriptor.....	66-2
	Using TopLink Workbench	66-2
	Configuring Maps to Wildcard.....	66-3
	Using TopLink Workbench	66-3
67	Configuring an XML Direct Mapping	
	XML Direct Mapping Configuration Overview.....	67-1
68	Configuring an XML Composite Direct Collection Mapping	
	XML Composite Direct Collection Mapping Configuration Overview.....	68-1
69	Configuring an XML Composite Object Mapping	
	XML Composite Object Mapping Configuration Overview.....	69-1
70	Configuring an XML Composite Collection Mapping	
	XML Composite Collection Mapping Configuration Overview	70-1
71	Configuring an XML Any Object Mapping	
	XML Any Object Mapping Configuration Overview.....	71-1
72	Configuring an XML Any Collection Mapping	
	XML Any Collection Mapping Configuration Overview	72-1
73	Configuring an XML Transformation Mapping	
	XML Transformation Mapping Configuration Overview.....	73-1
Part XV	Using TopLink Overview	
74	Understanding the Persistence Layer	
	Overview of the Persistence Layer.....	74-1
	Sessions	74-1
	Data Access	74-1
	Cache.....	74-2
	Queries and Expressions.....	74-2
	Transactions.....	74-3
Part XVI	TopLink Sessions	
75	Understanding TopLink Sessions	
	Session Types	75-1
	Session Concepts	75-2

Session Architecture	75-2
Object Cache	75-3
Connection Pools	75-3
Query Mechanism.....	75-4
Java Object Builder	75-4
Session Configuration and the sessions.xml File.....	75-4
Session Customization	75-4
Acquiring a Session at Run Time With the Session Manager.....	75-5
Managing Session Events With the Session Event Manager	75-5
Session Event Manager Events	75-6
Session Event Listeners	75-7
Logging.....	75-7
Log Types	75-8
Log Output.....	75-10
Log Level.....	75-10
Logging SQL.....	75-10
Logging Chained Exceptions	75-10
Viewing TopLink Log Messages From the Application Server Control Console	75-10
Profiler	75-11
TopLink Profiler.....	75-11
Oracle Dynamic Monitoring System (DMS)	75-11
Integrity Checker.....	75-12
Exception Handlers.....	75-12
Registering Descriptors	75-12
Sessions and CMP	75-13
Sessions and Sequencing.....	75-13
Server and Client Sessions	75-13
Three-Tier Architecture Overview	75-14
Advantages of the TopLink Three-Tier Architecture	75-15
Shared Resources	75-15
Providing Read Access.....	75-15
Providing Write Access.....	75-16
Security and User Privileges	75-17
Concurrency.....	75-18
Connection Allocation.....	75-18
Unit of Work Sessions	75-19
Isolated Client Sessions	75-19
Isolated Client Sessions and Oracle Virtual Private Database (VPD)	75-20
VPD With Oracle Database Proxy Authentication	75-21
VPD Without Oracle Database Proxy Authentication	75-21
Isolated Client Session Life Cycle.....	75-22
Isolated Client Session Limitations.....	75-23
Historical Client Sessions.....	75-24
Historical Client Session Limitations	75-25
Session Broker and Client Sessions	75-25
Session Broker Architecture	75-26
Committing a Transaction with a Session Broker	75-27

Committing a Session with a JTA Driver: Two-Phase Commits	75-27
Committing a Session Without a JTA Driver: Two-Stage Commits.....	75-27
Session Broker Session Limitations	75-27
Many-to-Many Join Tables and Direct Collection Tables	75-27
Session Broker Alternatives.....	75-28
Database Linking	75-28
Multiple Sessions	75-28
Database Sessions	75-28
Remote Sessions	75-29
Architectural Overview.....	75-30
Application Layer	75-30
Transport Layer.....	75-30
Server Layer.....	75-31
Remote Session Concepts.....	75-31
Securing Remote Session Access	75-31
Queries.....	75-31
Refreshing	75-31
Indirection	75-32
Cursored Streams.....	75-32
Unit of Work	75-32
Sessions and the Cache	75-32
Server and Database Session Cache	75-32
Isolated Session Cache.....	75-33
Historical Session Cache	75-33
Understanding the Session API	75-33

76 Creating Sessions

Session Creation Overview	76-1
Creating a Sessions Configuration	76-1
Using TopLink Workbench	76-2
Configuring a Sessions Configuration	76-2
Using TopLink Workbench	76-2
Creating a Server Session	76-4
Using TopLink Workbench	76-4
Using Java.....	76-5
Creating Session Broker and Client Sessions	76-6
Using TopLink Workbench	76-6
Using Java.....	76-8
Creating Database Sessions	76-8
Using TopLink Workbench	76-8
Using Java.....	76-10
Creating Remote Sessions	76-10
Using Java.....	76-10
Server	76-11
Client.....	76-11

77 Configuring a Session

Configuring Common Session Options	77-1
Configuring a Primary Mapping Project.....	77-2
Using TopLink Workbench	77-3
Using Java.....	77-3
Configuring a Session Login.....	77-4
Configuring Logging	77-4
Using TopLink Workbench	77-5
Using Java.....	77-7
Using Session Logging API	77-7
Configuring a Session to use java.util.logging Package.....	77-8
Configuring Logging in a CMP Application	77-9
Configuring Multiple Mapping Projects.....	77-9
Using TopLink Workbench	77-9
Using Java.....	77-10
Configuring a Performance Profiler	77-10
Using TopLink Workbench	77-11
Using Java.....	77-12
Configuring an Exception Handler.....	77-12
Using TopLink Workbench	77-12
Using Java.....	77-12
Configuring Customizer Class	77-13
Using TopLink Workbench	77-14
Configuring the Server Platform.....	77-14
Using TopLink Workbench	77-15
Using Java.....	77-17
Configuring Session Event Listeners	77-17
Using TopLink Workbench	77-17
Using Java.....	77-18
Configuring the Integrity Checker	77-18
Using Java.....	77-19
Configuring Connection Policy	77-19
Using TopLink Workbench	77-20
Using Java.....	77-21
Configuring Named Queries at the Session Level	77-21
Using Java.....	77-22

78 Acquiring and Using Sessions at Run Time

Session Acquisition Overview	78-1
Understanding the Session Manager	78-2
Multiple Sessions.....	78-2
Acquiring the Session Manager	78-2
Acquiring a Session from the Session Manager	78-3
Loading a Session from sessions.xml Using Defaults.....	78-3
Loading a Session from sessions.xml with an Alternative Class Loader.....	78-4
Loading a Session from an Alternative Session Configuration File	78-4

Loading a Session Without Logging In.....	78-5
Reloading and Refreshing Session Configuration.....	78-5
Refreshing a Session when the Class Loader Changes.....	78-6
Acquiring a Client Session	78-6
Acquiring an Isolated Client Session	78-7
Acquiring a Historical Client Session.....	78-7
Acquiring a Client Session That Uses Exclusive Connections	78-7
Acquiring a Client Session that Uses Connection Properties	78-8
Acquiring a Client Session that Uses a Named Connection Pool.....	78-8
Acquiring a Client Session that Does Not Use Lazy Connection Allocation	78-9
Logging In to a Session	78-9
Using Session API	78-9
Logging Out of a Session	78-10
Storing Sessions in the Session Manager Instance	78-10
Destroying Sessions in the Session Manager Instance	78-10
 79 Configuring Server Sessions	
Server Session Configuration Overview	79-1
Configuring Internal Connection Pools	79-1
Configuring External Connection Pools	79-2
 80 Configuring Exclusive Isolated Client Sessions for Virtual Private Database	
Exclusive Isolated Client Session Configuration Overview.....	80-1
PostAcquireExclusiveConnection Event Handler	80-1
Using Java.....	80-2
PreReleaseExclusiveConnection Event Handler.....	80-2
Using Java.....	80-2
NoRowsModifiedSessionEvent Event Handler.....	80-3
Using Java.....	80-3
ValidationException Handler.....	80-3
 81 Configuring Historical Client Sessions	
Historical Client Session Configuration Overview.....	81-1
Configuring Historical Client Sessions Using an Oracle Platform	81-1
Configuring Historical Client Sessions Using a TopLink HistoryPolicy.....	81-1
 82 Configuring Session Broker and Client Sessions	
Session Broker and Client Session Configuration Overview	82-1
Removing, Renaming, or Adding Sessions	82-1
Using TopLink Workbench	82-2
 83 Configuring Database Sessions	
Database Session Configuration Overview.....	83-1
Configuring External Connection Pools	83-1

Part XVII Data Access

84 Understanding Data Access

Data Access Concepts	84-1
Externally Managed Transactional Data Sources.....	84-1
Data Source Login Types	84-2
DatabaseLogin.....	84-2
EISLogin	84-3
Data Source Platform Types	84-3
Database Platforms.....	84-3
EIS Platforms	84-4
Authentication	84-5
Simple JDBC Authentication.....	84-5
Oracle Database Proxy Authentication.....	84-5
Auditing	84-6
Connections.....	84-6
Connection Pools.....	84-7
Internal Connection Pools.....	84-7
External Connection Pools.....	84-8
Default (Write) and Read Connection Pools.....	84-8
Sequence Connection Pools.....	84-8
Application-Specific Connection Pools.....	84-9
Understanding Data Access API	84-9
Login Inheritance Hierarchy	84-10
Platform Inheritance Hierarchy	84-10

85 Configuring a Data Source Login

Configuring Common Data Source Login Options	85-1
Configuring User Name and Password	85-1
Using TopLink Workbench	85-2
Configuring Password Encryption	85-2
Using Java.....	85-2
Configuring External Connection Pooling	85-2
Using TopLink Workbench	85-3
Configuring Properties	85-4
Using TopLink Workbench	85-4
Using Java.....	85-5
Configuring a Default Null Value at the Login Level	85-5
Using Java.....	85-6

86 Configuring a Database Login

Database Login Configuration Overview	86-1
Configuring a Relational Database Platform at the Session Level	86-1
Using TopLink Workbench	86-1
Configuring Database Login Connection Options	86-2
Using TopLink Workbench	86-2

Configuring Sequencing at the Session Level.....	86-4
Using TopLink Workbench	86-4
Using Java.....	86-5
Using the Platform Default Sequence	86-5
Configuring Multiple Sequences	86-6
Configuring Query Sequencing	86-7
Configuring a Table Qualifier	86-8
Using TopLink Workbench	86-8
Configuring JDBC Options	86-9
Using TopLink Workbench	86-9
Using Java.....	86-11
Configuring Advanced Options	86-11
Using TopLink Workbench	86-11
Configuring Oracle Database Proxy Authentication	86-12
Using Java.....	86-13

87 Configuring an EIS Login

EIS Login Configuration Overview.....	87-1
Configuring an EIS Data Source Platform at the Session Level	87-1
Using TopLink Workbench	87-1
Configuring EIS Connection Specification Options at the Session Level.....	87-2
Using TopLink Workbench	87-2

88 Creating an Internal Connection Pool

Internal Connection Pool Creation Overview	88-1
Using TopLink Workbench	88-1

89 Configuring an Internal Connection Pool

Internal Connection Pool Configuration Overview	89-1
Configuring Connection Pool Sizes	89-1
Using TopLink Workbench	89-2
Configuring Properties.....	89-2
Using TopLink Workbench	89-2
Using Java.....	89-3
Configuring a Nontransactional Read Login.....	89-3
Using TopLink Workbench	89-3
Configuring Connection Pool Connection Options.....	89-4
Using TopLink Workbench	89-4
Configuring Exclusive Read Connections.....	89-6
Using TopLink Workbench	89-6

Part XVIII Cache

90 Understanding the Cache

Cache Architecture	90-1
Session Cache.....	90-2

Unit of Work Cache	90-2
Cache Concepts	90-2
Cache Type and Object Identity	90-3
Full Identity Map	90-3
Weak Identity Map	90-3
Soft and Hard Cache Weak Identity Maps.....	90-4
No Identity Map.....	90-4
Guidelines for Configuring the Cache and Identity Maps	90-4
Understanding the Internals of Soft and Hard Cache Weak Identity Map	90-5
Querying and the Cache.....	90-6
Handling Stale Data.....	90-6
Configure a Locking Policy	90-6
Configure the Cache on a Per-Class Basis	90-7
Force a Cache Refresh When Required on a Per-Query Basis.....	90-7
Configure Cache Invalidation	90-7
Configure Cache Coordination.....	90-7
Explicit Query Refreshes.....	90-7
Refresh Policy	90-8
EJB Finders and Refresh Policy	90-8
Cache Invalidation	90-8
Cache Coordination	90-9
Cache Isolation	90-9
Cache Locking and Transaction Isolation.....	90-9
Cache Optimization	90-9
Understanding Cache Coordination	90-10
When to use Cache Coordination.....	90-10
Coordinated Cache Architecture	90-11
Session	90-11
Descriptor	90-11
Unit of Work	90-12
Coordinated Cache Types.....	90-12
JMS Coordinated Cache	90-12
RMI Coordinated Cache	90-12
CORBA Coordinated Cache.....	90-13
Custom Coordinated Cache	90-13
Understanding the Cache API	90-13
Object Identity API	90-13
Cache Refresh API	90-14
Cache Invalidation API	90-14
Cache Coordination API	90-15

91 Configuring a Coordinated Cache

Configuring Common Coordinated Cache Options	91-1
Configuring the Synchronous Change Propagation Mode.....	91-2
Using TopLink Workbench	91-2
Configuring a Service Channel	91-3
Using TopLink Workbench	91-3

Configuring a Multicast Group Address	91-4
Using TopLink Workbench	91-5
Configuring a Multicast Port	91-5
Using TopLink Workbench	91-6
Configuring a Naming Service Type	91-7
Configuring JNDI Naming Service Information	91-7
Using TopLink Workbench	91-8
Configuring RMI Registry Naming Service Information	91-9
Using TopLink Workbench	91-10
Configuring an Announcement Delay	91-11
Using TopLink Workbench	91-12
Configuring Connection Handling	91-13
Using TopLink Workbench	91-13
Configuring Context Properties	91-14
Using TopLink Workbench	91-14
Configuring a Packet Time-to-Live	91-15
Using TopLink Workbench	91-16

92 Configuring a JMS Coordinated Cache

JMS Coordinated Cache Configuration Overview	92-1
Configuring a Topic Name	92-1
Using TopLink Workbench	92-1
Configuring a Topic Connection Factory Name	92-2
Using TopLink Workbench	92-2
Configuring a Topic Host URL	92-3
Using TopLink Workbench	92-3

93 Configuring an RMI Coordinated Cache

RMI Coordinated Cache Configuration Overview	93-1

94 Configuring a CORBA Coordinated Cache

CORBA Coordinated Cache Configuration Overview	94-1

95 Configuring a Custom Coordinated Cache

Custom Coordinated Cache Configuration Overview	95-1
Configuring Transport Class	95-1
Using TopLink Workbench	95-1

Part XIX Queries

96 Understanding TopLink Queries

Query Types	96-1
Query Concepts	96-2
Call.....	96-3
DatabaseQuery	96-3

Data-Level and Object-Level Queries	96-3
Summary Queries	96-3
Descriptor Query Manager	96-3
TopLink Expressions	96-3
Query Keys.....	96-4
Query Languages	96-4
SQL Queries.....	96-4
EJB QL Queries.....	96-5
XML Queries.....	96-5
EIS Interactions.....	96-6
Query by Example	96-6
Building Queries	96-6
Executing Queries	96-7
Handling Query Results	96-8
Collection Query Results.....	96-8
Report Query Results	96-8
Stream and Cursor Query Results	96-8
Session Queries	96-9
Read-Object Session Queries	96-9
Create, Update, and Delete Object Session Queries.....	96-10
Database Queries	96-10
Object-Level Read Query	96-11
ReadObjectQuery	96-11
ReadAllQuery.....	96-11
Partial Object Queries.....	96-11
Join Reading and Object-Level Read Queries.....	96-12
Fetch Groups and Object Level Read Queries	96-12
Data-Level Read Query	96-13
DataReadQuery	96-13
DirectReadQuery	96-13
ValueReadQuery	96-13
Object-Level Modify Query	96-13
WriteObjectQuery.....	96-14
UpdateObjectQuery	96-14
InsertObjectQuery.....	96-14
DeleteObjectQuery.....	96-14
UpdateAllQuery.....	96-14
Object-Level Modify Queries and Privately Owned Parts	96-14
Data-Level Modify Query	96-14
Report Query	96-15
Named Queries	96-15
Call Queries	96-16
SQL Calls	96-16
SQLCall.....	96-17
StoredProcedureCall.....	96-17
StoredFunctionCall	96-17
Oracle Extensions.....	96-17

EJB QL Calls	96-18
Enterprise Information System (EIS) Interactions	96-18
IndexedInteraction	96-19
MappedInteraction	96-19
XMLInteraction	96-19
XQueryInteraction	96-19
QueryStringInteraction	96-19
Redirect Queries	96-19
Historical Queries	96-20
Using an ObjectLevelReadQuery With an AsOfClause	96-21
Using an ObjectLevelReadQuery With Expression Operator asOf	96-21
Using an ObjectLevelReadQuery in a Historical Session.....	96-22
Interface and Inheritance Queries	96-22
Descriptor Query Manager Queries	96-22
Configuring Named Queries	96-22
Configuring Default Query Implementations	96-22
Configuring Additional Join Expressions	96-23
EJB Finders	96-23
Predefined Finders	96-24
Predefined CMP Finders for EJB 2.0	96-24
Predefined CMP Finders for EJB 1.1	96-25
Predefined BMP Finders	96-25
Default Finders	96-26
Call Finders	96-26
DatabaseQuery Finders.....	96-26
Named Query Finders.....	96-27
Primary Key Finders.....	96-27
Expression Finders.....	96-27
EJB QL Finders.....	96-27
SQL Finders.....	96-28
Redirect Finders.....	96-28
The ejbSelect Method	96-28
Queries and the Cache	96-29
Configuring the Cache	96-29
Using In-Memory Queries	96-30
Configuring Cache Usage for In-Memory Queries.....	96-30
Expression Options for In-Memory Queries.....	96-31
Handling Exceptions Resulting From In-Memory Queries.....	96-33
Primary Key Queries and the Cache	96-34
Disabling the Identity Map Cache Update During a Read Query	96-34
Refreshing the Cache	96-35
Object Refresh.....	96-35
Cascading Object Refresh	96-35
Refreshing the Identity Map Cache During a Read Query	96-35
Caching Query Results in the Session Cache	96-36
Caching Query Results	96-36
Internal Query Cache Restrictions.....	96-36

Caching and EJB Finders.....	96-36
Caching Options.....	96-37
Disabling Cache for Returned Finder Results	96-37
Refreshing Finder Results.....	96-38
Understanding the Query API.....	96-38

97 Understanding TopLink Expressions

Understanding the Expression Framework	97-1
Expressions Compared to SQL	97-1
Expression Components	97-2
Boolean Logic.....	97-3
Database Functions	97-3
Mathematical Functions	97-4
XMLType Functions	97-4
Platform and User-Defined Functions	97-4
Expressions for One-to-One and Aggregate Object Relationships.....	97-5
Expressions for Joining and Complex Relationships	97-5
Understanding Joins.....	97-5
Using TopLink Expression API For Joins.....	97-6
Parameterized Expressions	97-7
Expression Method getParameter.....	97-8
Expression Method getField.....	97-9
Query Keys and Expressions	97-9
Using Multiple Expressions	97-10
Subselects and Subqueries	97-10
Parallel Expressions	97-11
Data Queries and Expressions	97-11
getField	97-11
getTable	97-12
Creating an Expression	97-12
Using TopLink Workbench	97-12
Adding Arguments.....	97-14
Using Java.....	97-15
Creating and Using a User-Defined Function	97-16
Making a User-Defined Function Available to a Specific Platform.....	97-16
Making a User-Defined Function Available to All Platforms	97-17
Using a User-Defined Function	97-17

98 Using Basic Query API

Using Session Queries	98-1
Reading Objects With a Session Query	98-1
Reading an Object With a Session Query	98-2
Reading All Objects With a Session Query	98-2
Refreshing an Object with a Session Query	98-2
Creating, Updating, and Deleting Objects With a Session Query	98-3
Writing a Single Object to the Database with a Session Query	98-3

Writing All Objects to the Database With a Session Query	98-3
Adding New Objects to the Database with a Session Query	98-4
Modifying Existing Objects in the Database with a Session Query	98-4
Deleting Objects in the Database with a Session Query	98-4
Using DatabaseQuery Queries	98-4
Reading Objects Using a DatabaseQuery	98-4
Basic DatabaseQuery Read Operations	98-5
Reading Objects Using Partial Object Queries	98-5
Reading Objects Using Report Queries	98-6
Reading Objects Using Query By Example	98-6
Specifying Read Ordering	98-8
Specifying a Collection Class	98-9
Specifying the Maximum Rows Returned	98-9
Configuring Query Timeout at the Query Level	98-10
Using Batch Reading	98-10
Using Join Reading	98-11
Creating, Updating, and Deleting Objects with a DatabaseQuery	98-12
Write Query Overview	98-12
UpdateAll Queries	98-13
Noncascading Write Queries	98-13
Disabling the Identity Map Cache During a Write Query	98-14
Reading Data With a DatabaseQuery	98-14
Using a DataReadQuery	98-15
Using a DirectReadQuery	98-15
Using a ValueReadQuery	98-15
Updating Data With a DatabaseQuery	98-15
Specifying a Custom SQL String in a DatabaseQuery	98-16
Specifying a Custom EJB QL String in a DatabaseQuery	98-16
Using Parameterized SQL and Statement Caching in a DatabaseQuery	98-17
Using Named Queries	98-17
Using SQL Calls	98-18
Using an SQLCall	98-18
Specifying a SQLCall Input Parameter	98-19
Specifying a SQLCall Output Parameter	98-19
Specifying a SQLCall Input / Output Parameter	98-20
Using a StoredProcedureCall	98-20
Specifying an Input Parameter	98-21
Specifying an Output Parameter	98-21
Specifying an Input / Output Parameter	98-22
Using an Output Parameter Event	98-22
Using a StoredFunctionCall	98-22
Using EJB QL Calls	98-23
Using EIS Interactions	98-24
Handling Exceptions	98-24
Handling Collection Query Results	98-25
Handling Report Query Results	98-25

99 Using Advanced Query API

Using Redirect Queries	99-1
Creating a Redirect Query	99-1
Using Historical Queries	99-2
Using Queries With Fetch Groups	99-2
Configuring Default Fetch Group Behavior	99-3
Querying With a Static Fetch Group	99-3
Querying With a Dynamic Fetch Group	99-4
Querying on Interfaces	99-4
Querying on an Inheritance Hierarchy	99-4
Appending Additional Join Expressions	99-4
Using Java	99-5
Using Queries on Variable One-to-One Mappings	99-5
Using Oracle Database Features	99-5
Oracle Hints	99-6
Hierarchical Queries	99-6
startWith Parameter	99-7
connectBy Parameter	99-7
orderSibling Parameter	99-7
Using EJB Finders	99-7
Creating a Finder	99-8
ejb-jar.xml Finder Options	99-9
Using DatabaseQuery Finders	99-10
Using Named Query Finders	99-10
Using Primary Key Finders	99-11
Using EJB QL Finders	99-11
Using SQL Finders	99-12
Using Redirect Finders	99-12
Using the ejbSelect Method	99-15
Handling Cursor and Stream Query Results	99-16
Cursors and Java Iterators	99-16
Traversing Data With Scrollable Cursors	99-16
Java Streams	99-17
Cursored Stream Support	99-17
Optimizing Streams	99-18
Using Cursors and Streams With EJB Finders	99-19
Building the Query	99-19
Executing the Finder From the Client in EJB 1.1	99-19
Executing the Finder From the Client in EJB 2.0	99-20
Using Queries and the Cache	99-21
Caching Results in a ReadQuery	99-21
Configuring Cache Expiration at the Query Level	99-22

Part XX Transactions

100 Understanding TopLink Transactions

Unit of Work Architecture	100-1
Unit of Work Transaction Context.....	100-2
Unit of Work Transaction Demarcation.....	100-2
JTA Controlled Transactions	100-3
OTS Controlled Transactions	100-3
CMP Controlled Transactions	100-3
Unit of Work Transaction Isolation	100-4
Unit of Work Concepts	100-4
Unit of Work Benefits	100-4
Unit of Work Life Cycle	100-5
Unit of Work and Change Policy	100-6
Deferred Change Detection Policy	100-7
Object-Level Change Tracking Policy	100-7
Attribute Change Tracking Policy	100-8
Change Policy Mapping Support	100-8
Clones and the Unit of Work.....	100-9
Nested and Parallel Units of Work.....	100-9
Nested Unit of Work	100-10
Parallel Unit of Work.....	100-10
Commit and Rollback Transactions	100-10
Commit Transactions	100-10
Rollback Transactions.....	100-11
Primary Keys	100-11
Unit of Work Optimization.....	100-11
Understanding the Unit of Work API	100-11
Unit of Work as Session	100-12
Reading and Querying Objects with the Unit of Work.....	100-12
Locking and the Unit of Work	100-12
Example Model Object and Schema	100-12

101 Using Basic Unit of Work API

Acquiring a Unit of Work	101-1
Creating an Object	101-2
Modifying an Object	101-2
Associating a New Target to an Existing Source Object	101-3
Associating Without Reference to the Cache Object.....	101-3
Associating With Reference to the Cache Object.....	101-4
Associating a New Source to an Existing Target Object	101-6
Associating an Existing Source to an Existing Target Object	101-7
Deleting Objects	101-7
Using privateOwnedRelationship	101-8
Explicitly Deleting from the Database	101-9
Understanding the Order in Which Objects Are Deleted	101-9

102 Using Advanced Unit of Work API

Registering and Unregistering Objects	102-1
Creating and Registering an Object in One Step	102-2
Using registerNewObject	102-2
Registering a New Object with registerNewObject	102-2
Associating New Objects With One Another	102-3
Using registerAllObjects	102-4
Using Registration and Existence Checking	102-5
Check Database	102-5
Assume Existence	102-5
Assume Nonexistence	102-5
Working with Aggregates.....	102-6
Unregistering Working Clones	102-6
Declaring Read-Only Classes	102-6
Configuring Read-Only Classes for a Single Unit of Work	102-6
Configuring Default Read-Only Classes	102-7
Read-Only Descriptors	102-7
Writing Changes Before Commit Time	102-7
Using Conforming Queries and Descriptors	102-8
Guidelines for Using Conforming	102-8
Use Only In-Memory Queries that Support Conforming.....	102-8
Consider How Conforming Affects Database Results	102-9
If You Want an Object in the Conformed Results, Register it in the Unit of Work.....	102-9
Using Conforming Queries.....	102-10
Using Conforming Descriptors	102-11
Conforming Query Alternatives	102-11
Using Unit of Work Method writeChanges Instead of Conforming.....	102-11
Using Unit of Work Properties Instead of Conforming	102-12
Merging Changes in Working Copy Clones	102-12
Resuming a Unit of Work After Commit	102-14
Reverting a Unit of Work	102-14
Using a Nested or Parallel Unit of Work	102-14
Parallel Unit of Work.....	102-15
Nested Unit of Work.....	102-15
Using a Unit of Work With Custom SQL	102-15
Controlling the Order of Delete Operations	102-16
Using the Unit of Work setShouldPerformDeletesFirst Method	102-16
Using the Descriptor addConstraintDependencies Method	102-16
Using deleteAllObjects Without addConstraintDependencies	102-16
Using deleteAllObjects with addConstraintDependencies	102-17
Using Optimistic Read Locking With forceUpdateToVersionField	102-17
Forcing a Check of the Optimistic Read Lock.....	102-17
Forcing a Version Field Update	102-18
Disabling forceUpdateToVersionField	102-20
Implementing User and Date Auditing with the Unit of Work	102-20
Integrating the Unit of Work With an External Transaction Service	102-20
Acquiring a Unit of Work With an External Transaction Service.....	102-21

Using a Unit of Work When an External Transaction Exists	102-21
Using a Unit of Work When No External Transaction Exists.....	102-22
Using the Unit of Work to Handle External Transaction Timeouts and Exceptions	102-23
External Transaction Commit Timeouts.....	102-23
External Transaction Commit Exceptions	102-23
Integrating the Unit of Work With CMP	102-24
CMP Transaction Attribute.....	102-24
Local Transactions.....	102-25
Nondeferred Changes	102-25
Database Transaction Isolation Levels	102-25
General Factors Affecting Transaction Isolation Level.....	102-26
External Applications	102-26
TopLink Coordinated Cache	102-26
DatabaseLogin Method setTransactionIsolation	102-27
Reading Through the Write Connection	102-27
Managing Cache Access.....	102-28
CMP and External Transactions	102-29
Read Uncommitted Level	102-29
Read Committed Level.....	102-30
Repeatable Read Levels.....	102-30
Serializable Read Levels	102-30
Troubleshooting a Unit of Work.....	102-30
Avoiding the Use of Post-Commit Clones	102-30
Determining Whether or Not an Object Is the Cache Object.....	102-31
Dumping the Contents of a Unit of Work	102-32
Handling Exceptions	102-33
Exceptions at Commit Time	102-33
Exceptions During Conforming.....	102-33
Validating a Unit of Work.....	102-34
Validating the Unit of Work Before Commit Time.....	102-34
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-2
.....	103-2

.....	103-2
.....	103-2
.....	103-2
.....	103-2
.....	103-2
.....	103-2
.....	103-2
.....	103-2
.....	103-2
.....	103-2

Index

Preface

Oracle TopLink Developer's Guide explains how to use Oracle TopLink to design, implement, deploy, and optimize an advanced persistence or object-to-XML transformation layer for a wide range of Java 2 Enterprise Edition (J2EE) and Java applications. It describes TopLink mapping metadata, sessions, data access, queries, transactions (both with and without an external transaction controller), and cache.

It describes how to use TopLink application development tools and how to integrate TopLink with a variety of J2EE containers. It also introduces the concepts with which you should be familiar to get the most out of TopLink.

Audience

Oracle TopLink Developer's Guide is intended for application developers creating applications that use TopLink to manage persistence.

This document assumes that you are familiar with the concepts of object-oriented programming, the Enterprise JavaBeans (EJB) specification, and your own particular Java development environment.

The document also assumes that you are familiar with your particular operating system (such as Windows, UNIX, or other). The general operation of any operating system is described in the user documentation for that system, and is not repeated in this manual.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

<http://www.oracle.com/accessibility/>

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an

otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documentation

For more information, see the following documents:

- *Oracle TopLink Release Notes*
- Oracle Application Server Release Notes
- *Oracle TopLink Getting Started Guide*
- *Oracle TopLink API Reference*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Part XV

Using TopLink Overview

This part describes how to associate a TopLink project with a particular instance of a data source and use it to manage persistence in your application. It contains the following chapters:

- [Chapter 74, "Understanding the Persistence Layer"](#)

This chapter provides an overview of how to use sessions, queries, and transactions in your application.

Understanding the Persistence Layer

This chapter describes the following:

- [Overview of the Persistence Layer](#)
- [Sessions](#)
- [Data Access](#)
- [Cache](#)
- [Queries and Expressions](#)
- [Transactions](#)

Overview of the Persistence Layer

The purpose of your application's persistence layer is to use a session (see "[Sessions](#)" on page 74-1) at run time to associate mapping metadata (see "[Mapping Metadata](#)" on page 2-12) and a data source (see "[Data Access](#)" on page 74-1) in order to create, read, update, and delete persistent objects using the TopLink cache (see "[Cache](#)" on page 74-2), queries and expressions (see "[Queries and Expressions](#)" on page 74-2), as well as transactions (see "[Transactions](#)" on page 74-3).

Sessions

A session is the primary interface between the client application and the TopLink runtime, and represents the connection to the underlying data source.

For non-container-managed persistence (non-CMP) projects, TopLink offers several different session types (see "[Understanding TopLink Sessions](#)" on page 75-1), each optimized for different design requirements and architectures. The most commonly used session is the server session—a session that clients access on the server through a client session. The server session provides a shared cache and shared connection resources.

For CMP projects, the TopLink runtime creates and uses a session internally, but your application does not acquire or use this session directly. Depending on the application server you use, you can specify some of the parameters for this internal session (see "[<J2EE-Container>-ejb-jar.xml File](#)" on page 8-5).

Data Access

The login (if any) associated with a session determines how the TopLink runtime connects to the project's data source.

A login includes details of data source access, such as authentication, use of connection pools, and use of external transaction controllers. A login (an instance of `Login` interface) owns a data source platform.

A platform includes options specific to a particular data source, including such as binding, use of native SQL, use of batch writing, and sequencing. For more information about platforms, see ["Data Source Platform Types"](#) on page 84-3.

For projects that do not persist to a data source, a login is not required. For projects that do persist to a data source, a login is always required.

For relational and For more information, see ["Understanding Data Access"](#) on page 84-1

Cache

By default, a TopLink session provides an object level cache that guarantees object identity and enhances performance by reducing the number of times the application needs to access the data source. TopLink provides a variety of cache options, including locking, refresh, invalidation, isolation, and coordination. Using cache coordination, you can configure TopLink to synchronize changes with other instances of the deployed application. You configure most cache options at the session level. You can also configure cache options on a per-query basis, or on a descriptor to apply to all queries on the reference class.

For more information, see ["Understanding the Cache"](#) on page 90-1

Queries and Expressions

TopLink provides several object and data query types, and offers flexible options for query selection criteria, including the following:

- TopLink expressions
- EJB QL
- SQL
- Stored procedures
- Query by example

With these options, developers can build any type of query. Oracle recommends using predefined queries to define application queries. Predefined queries are held in the project metadata and referenced by name. This simplifies application development and encapsulates the queries to reduce maintenance costs.

When using EJB entity beans, you can code finders using only EJB QL (in addition to any of the other TopLink query options), enabling the application to comply with the J2EE specification.

Regardless of the architecture or persistent entity type, you are free to use any of the query options. TopLink Workbench provides the simplest way to define queries. Alternatively, you can build queries in code, using the TopLink API.

For more information, see the following:

- ["Understanding TopLink Queries"](#) on page 96-1
- ["Understanding TopLink Expressions"](#) on page 97-1

Transactions

TopLink provides the ability to write transactional code isolated from the underlying database and schema by using a **unit of work**.

The unit of work isolates changes in a transaction from other threads until it successfully commits the changes to the database. Unlike other transaction mechanisms, the unit of work automatically manages changes to the objects in the transaction, the order of the changes, and changes that might invalidate other TopLink caches. The unit of work manages these issues by calculating a minimal change set, ordering the database calls to comply with referential integrity rules and deadlock avoidance, and merging changed objects into the shared cache. In a clustered environment, the unit of work also synchronizes changes with the other servers in the coordinated cache.

If an application uses EJB entity beans, you do not access the unit of work API directly, but you still benefit from its features: the integration between the TopLink runtime and the J2EE container automatically uses the unit of work.

For more information, see "[Understanding TopLink Transactions](#)" on page 100-1.

Part XVI

TopLink Sessions

This part describes the TopLink artifact used to associate a TopLink project with a particular instance of a data source. It contains the following chapters:

- [Chapter 75, "Understanding TopLink Sessions"](#)

This chapter describes each of the different TopLink session types and important session concepts.

- [Chapter 76, "Creating Sessions"](#)

This chapter contains procedures for creating TopLink sessions.

- [Chapter 77, "Configuring a Session"](#)

This chapter explains how to configure TopLink session options common to two or more session types.

- [Chapter 78, "Acquiring and Using Sessions at Run Time"](#)

This chapter explains how to acquire and use a TopLink session at runtime.

- [Chapter 79, "Configuring Server Sessions"](#)

This chapter explains how to configure TopLink server and client sessions.

- [Chapter 80, "Configuring Exclusive Isolated Client Sessions for Virtual Private Database"](#)

This chapter explains how to configure a TopLink isolated client session.

- [Chapter 81, "Configuring Historical Client Sessions"](#)

This chapter explains how to configure a TopLink historical client session.

- [Chapter 82, "Configuring Session Broker and Client Sessions"](#)

This chapter explains how to configure TopLink session broker and client sessions.

- [Chapter 83, "Configuring Database Sessions"](#)

This chapter explains how to configure a TopLink database session suitable for simple single-user, single-data source and prototyping applications.

Understanding TopLink Sessions

A TopLink session provides the primary access to the TopLink runtime. It is the means by which your application performs all persistence operations with the data source that contains persistent objects.

A session associates data source platform information, data source login information, and mapping metadata for a particular application. You can reuse mapping metadata in different applications by defining different sessions.

TopLink provides different session types, each optimized for different design requirements and data access strategies. You can combine different session types in the same application.

This chapter explains the following:

- [Session Types](#)
- [Session Concepts](#)
- [Sessions and the Cache](#)
- [Understanding the Session API](#)

Session Types

[Table 75–1](#) lists the session types that you can use in a non-container-managed persistence (non-CMP) TopLink application and classifies them as basic or advanced. See ["Sessions and CMP"](#) on page 75-13 for information on using Oracle TopLink with CMP.

Table 75–1 *TopLink Session Types*

Session Type	Description	Type	TopLink Workbench	Java
Server and Client Sessions	Server sessions provide session management to a single data source (including shared object cache and connection pools) for multiple clients in a three-tier architecture using database or EIS platforms. This is the most flexible, scalable, and commonly used session. You acquire a client session from a server session at run time to provide access to a single data source for each client.	Basic	✓	✓
Unit of Work Sessions	Acquired from any session type (directly, or by way of an external transaction controller) to transactionally modify objects.	Basic		✓
Isolated Client Sessions	A special type of client session that uses a session cache isolated from the shared object cache of its parent server session.	Advanced		✓
Historical Client Sessions	A special type of client session that provides a read-only snapshot of object versions as of a specified time and uses a session cache isolated from the shared object cache of its parent server session.	Advanced		✓

Table 75–1 (Cont.) TopLink Session Types

Session Type	Description	Type	TopLink Workbench	Java
Session Broker and Client Sessions	Provides session management to multiple data sources for multiple clients by aggregating two or more server sessions (can also be used with database sessions). You acquire a client session from a session broker at run-time to provide access to all the data sources managed by the session broker for each client.	Advanced	✓	✓
Database Sessions	Provides session management to a single database for a single client suitable for simple or two-tiered applications. Oracle does not recommend this session type in three-tiered applications because it does not offer the same flexibility and scalability as the server session.	Basic	✓	✓
Remote Sessions	A client-side session that communicates over RMI with a corresponding dedicated client session and shared server session. Remote sessions handle object identity and marshalling and unmarshalling between client-side and server-side.	Advanced		✓

For more information, see the following:

- ["Creating Sessions"](#) on page 76-1
- ["Configuring a Session"](#) on page 77-1
- ["Acquiring and Using Sessions at Run Time"](#) on page 78-1

Session Concepts

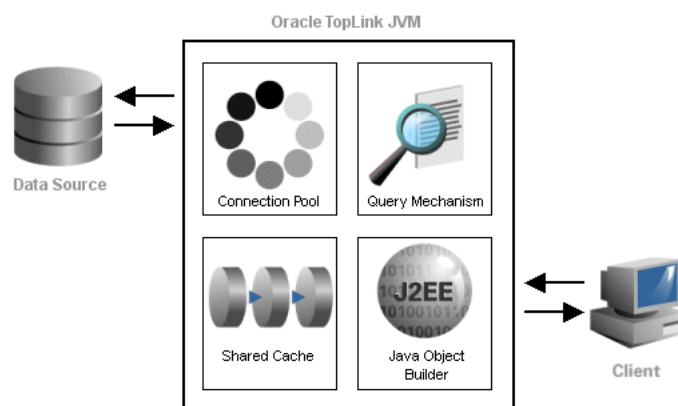
This section describes concepts unique to TopLink sessions, including the following:

- [Session Architecture](#)
- [Session Configuration and the sessions.xml File](#)
- [Session Customization](#)
- [Acquiring a Session at Run Time With the Session Manager](#)
- [Managing Session Events With the Session Event Manager](#)
- [Logging](#)
- [Profiler](#)
- [Integrity Checker](#)
- [Exception Handlers](#)
- [Registering Descriptors](#)
- [Sessions and CMP](#)
- [Sessions and Sequencing](#)

Session Architecture

As [Figure 75–1](#) illustrates, a session instance is composed of the following components:

- [Object Cache](#)
- [Connection Pools](#)
- [Query Mechanism](#)
- [Java Object Builder](#)

Figure 75-1 Simple TopLink Session Architecture

How these session components are implemented and how they interact depends on the type of session. For example, for server and client sessions, the server session provides a connection pool and shared object cache on behalf of all client sessions acquired from it.

Object Cache

TopLink sessions provide an object cache. This cache, known as the **session cache**, retains information about objects that are read from or written to the database, and is a key element for improving the performance of a TopLink application.

Typically, a server session's object cache is shared by all client sessions acquired from it. That is, for a Server session `myServerSession`, each client session acquired by calling server session method `acquireClientSession` shares the same object cache as `myServerSession`.

Isolated and historical sessions provide their own session cache isolated from the shared object cache of their parent server session. For more information, see ["Isolated Client Sessions"](#) on page 75-19 and ["Historical Client Sessions"](#) on page 75-24.

You can easily manage concurrent access to this shared cache by using a unit of work session acquired from any session. For more information, see ["Unit of Work Sessions"](#) on page 75-19.

For more information, see ["Sessions and the Cache"](#) on page 75-32.

Connection Pools

A **connection pool** is a collection of reusable connections to a single data source.

Note: To simultaneously access multiple databases from within a single session, use a session broker. For more information, see ["Session Broker and Client Sessions"](#) on page 75-25.

Because creating a data source connection is usually expensive, a properly configured connection pool significantly improves performance.

You can configure your session to use internal connection pools provided by TopLink or external connection pools provided by a JDBC driver or J2EE container. By default, TopLink uses internal connection pools.

Internal connection pools are usually used in non-EJB applications, or when an external transaction controller (JTA) is not used. If you configure your session to use internal connection pools, you can configure its default read and write connection pools. You can create special purpose connection pools for application-specific purposes (named connection pools) or exclusively for sequencing (sequence connection pool). For more information, see ["Internal Connection Pools"](#) on page 84-7.

External connection pools are usually used in EJB applications and when an external transaction controller (JTA) is used. For more information, see ["External Connection Pools"](#) on page 84-8.

For more information about data access configuration in general, see ["Understanding Data Access"](#) on page 84-1.

Query Mechanism

At run time, your application uses a session to perform all persistence operations: creating, reading, updating, and deleting objects. You perform these operations using TopLink queries and expressions with the session query API.

For more information, see ["Understanding TopLink Queries"](#) on page 96-1.

Java Object Builder

When you use object-level read queries, TopLink automatically builds Java objects from the data retrieved. When you use object-level write queries, TopLink automatically converts the affected Java objects into the appropriate data native to your data source.

Session Configuration and the sessions.xml File

TopLink provides two ways to configure your sessions: through Java code using the Session API, or using TopLink Workbench to build a session configuration file, the `sessions.xml` file.

In most cases, you configure sessions for the application using the `sessions.xml` file. This file is an Extensible Markup Language (XML) file that contains all sessions that are associated with the application. The `sessions.xml` file can contain any number of sessions and session types.

Oracle recommends that you use the `sessions.xml` file to deploy a TopLink application, because it provides the following advantages:

- It is easy to create and maintain in TopLink Workbench.
- It is easy to troubleshoot.
- It provides access to most session configuration options.
- It offers excellent flexibility, including the ability to modify deployed applications without recompiling.

For more information on creating a session in the `sessions.xml` file, see ["Session Creation Overview"](#) on page 76-1.

Session Customization

You can customize a session at run time by specifying a session customizer—a Java class that implements the `oracle.toplink.tools.sessionconfiguration.SessionCustomizer` interface and provides a default (zero-argument) constructor.

Since the `sessions.xml` file is not used for CMP projects, use `oracle.toplink.ejb.cmp.DeploymentCustomization` interface as a customizer when creating a CMP project to specify your database login information.

You use a session customizer to customize a session at run time through code API similar to how you use an amendment method to customize a descriptor (see ["Amendment and After-Load Methods"](#) on page 26-5).

For more information, see ["Configuring Customizer Class"](#) on page 77-13.

Acquiring a Session at Run Time With the Session Manager

The TopLink session manager enables developers to build a series of sessions that are maintained under a singleton object called the session manager.

The session manager is a static utility class that loads TopLink sessions from the `sessions.xml` file (see ["Session Configuration and the sessions.xml File"](#) on page 75-4), caches the sessions by name in memory, and provides a single access point for TopLink sessions.

At run time, TopLink will attempt to load the `sessions.xml` file from the two following default resource names: `sessions.xml` and `META-INF/sessions.xml`. Refer to [Chapter 9, "Packaging a TopLink Application"](#) for additional information.

The session manager supports the following session types:

- `ServerSession` (see ["Server and Client Sessions"](#) on page 75-13)
- `SessionBroker` (see ["Session Broker and Client Sessions"](#) on page 75-25)
- `DatabaseSession` (see ["Database Sessions"](#) on page 75-28)

The session manager has two main functions: it creates instances of these sessions and it ensures that only a single instance of each named session exists for any instance of a session manager.

The session manager instantiates sessions as follows:

1. The client application requests a session by name.
2. The session manager looks up the session name in the `sessions.xml` file. If the session name exists, the session manager instantiates the specified session; otherwise, it raises an exception.
3. After instantiation, the session remains viable until you shut down the application.

Once you have a session instance, you can use it to acquire additional types of sessions for special tasks. For example, you can acquire a unit of work from any session to perform transactional operations. You can acquire a client session from a server session to perform client operations in a three-tier architecture.

For more information, see ["Acquiring and Using Sessions at Run Time"](#) on page 78-1.

Managing Session Events With the Session Event Manager

Sessions raise **session events** for most session operations. Session events help you debug or coordinate the actions of multiple sessions.

The session event manager handles information about session events. Applications register session event listeners with the session event manager to receive session events.

For example, session event listeners play an important role in the configuration of isolated sessions (see ["Configuring Exclusive Isolated Client Sessions for Virtual"](#)

[Private Database](#)" on page 80-1). In an isolated session, if the TopLink runtime raises a `SessionEvent.NoRowsModified` event, it is handled by your `SessionEventListener` (see ["NoRowsModifiedSessionEvent Event Handler"](#) on page 80-3). This event listener is your opportunity to determine whether the update failure was due to a security violation (in which case you should not retry the operation) or due to an optimistic lock issue (in which case a retry may be appropriate). See ["Logging"](#) on page 75-7 for information on adding logging to your event listeners.

Another example is the use of session event listeners to configure proxy authentication in an Oracle Database. (see ["Configuring Oracle Database Proxy Authentication"](#) on page 86-12).

This section explains how to use session events, including the following:

- [Session Event Manager Events](#)
- [Session Event Listeners](#)

Session Event Manager Events

The session event manager supports the session events listed in the following tables:

- [Table 75–2, "Session Events"](#)
- [Table 75–3, "Unit of Work Events"](#)

Table 75–2 Session Events

Event	Description
MissingDescriptor	Raised if a descriptor is missing for a class being persisted. You can use this event to lazy register the descriptor or set of descriptors.
MoreRowsDetected	Raised when a <code>ReadObjectQuery</code> detects more than one row returned from the database. This event can indicate a possible error condition in your application.
NoRowsModified	Raised after update or delete SQL has been sent to the database and a row count of zero is returned.
OutputParametersDetected	Raised after a stored procedure call with output parameters executes. This event enables you to retrieve a result set and output parameters from a single stored procedure.
PostAcquireClientSession	Raised after a client Session is acquired
PostAcquireConnection	Raised after acquiring a connection
PostAcquireExclusiveConnection	Raised when a client Session, with isolated data, acquires an exclusive connection.
PostBeginTransaction	Raised after a database transaction starts
PostCommitTransaction	Raised after a database transaction commits
PostConnect	Raised after connecting to the database
PostExecuteQuery	Raised after the execution of every query on the session
PostLogin	Raised after the Session initializes and acquires connections
PostReleaseClientSession	Raised after releasing a client Session
PostRollbackTransaction	Raised after a database transaction rolls back
PreBeginTransaction	Raised before a database transaction starts
PreCommitTransaction	Raised before a database transaction commits
PreExecuteQuery	Raised before the execution of every query on the session
PreLogin	Raised before the Session initializes and acquires connections
PreReleaseClientSession	Raised before releasing a client Session

Table 75-2 (Cont.) Session Events

Event	Description
PreReleaseConnection	Raised before releasing a connection
PreReleaseExclusiveConnection	Raised before a client Session, with isolated data, releases its exclusive connection.
PreRollbackTransaction	Raised before a database transaction rolls back

Table 75-3 Unit of Work Events

Event	Description
PostAcquireUnitOfWork	Raised after a <code>UnitOfWork</code> is acquired
PostCalculateUnitOfWorkChangeSet	Raised after the commit has begun on the <code>UnitOfWork</code> and after the changes are calculated. The <code>UnitOfWorkChangeSet</code> , at this point, will contain change sets without the version fields updated and without identity field type primary keys. These will be updated after the insert, or update, of the object.
PostCommitUnitOfWork	Raised after a <code>UnitOfWork</code> commits
PostDistributedMergeUnitOfWorkChangeSet	Raised after a <code>UnitOfWork</code> change set has been merged when that change set has been received from a distributed session.
PostMergeUnitOfWorkChangeSet	Raised after a <code>UnitOfWork</code> change set has been merged.
PostReleaseUnitOfWork	Raised on a <code>UnitOfWork</code> after it is released.
PostResumeUnitOfWork	Raised on a <code>UnitOfWork</code> after it resumes.
PreCalculateUnitOfWorkChangeSet	Raised after the commit has begun on the <code>UnitOfWork</code> but before the changes are calculated.
PreCommitUnitOfWork	Raised before a <code>UnitOfWork</code> commits.
PreDistributedMergeUnitOfWorkChangeSet	Raised before a <code>UnitOfWork</code> change set has been merged when that change set has been received from a distributed session.
PreMergeUnitOfWorkChangeSet	Raised before a <code>UnitOfWork</code> change set has been merged.
PrepareUnitOfWork	Raised after the a <code>UnitOfWork</code> flushes its SQL, but before it commits its transaction.
PreReleaseUnitOfWork	Raised on a <code>UnitOfWork</code> before it is released.

Session Event Listeners

You can create session event listeners in two ways: either by implementing the `SessionEventListener` interface, or by extending the `SessionEventAdapter` class.

To register a `SessionEventListener` for session events, register it with a session using the `SessionEventManager` method `addListener`.

For more information, see ["Configuring Session Event Listeners"](#) on page 77-17.

Logging

You can configure a session to write run-time information to a TopLink log. This information includes status, diagnostic, SQL, and, when profiling is enabled, performance data (see ["Measuring Performance With the TopLink Profiler"](#) or ["Measuring Performance With the Oracle Dynamic Monitoring System \(DMS\)"](#)).

Logging options are configurable at the session level (see ["Configuring Logging"](#) on page 77-4).

Note: To facilitate the debugging, you can add logging to your listeners to only log the events that are of the interest to your application. Within the session context, use the following logging utility:

```
Session.getSessionLog().log(int level, String message)
```

Without the session context, use the following logging utility:

```
AbstractSessionLog.getLog().log(int level, String message)
```

Both the `getSessionLog` and `getLog` methods return a session log (an instance of a `SessionLog` interface) loaded with an accessor's log messages and SQL. Then the session log performs logging at the level that you specify.

For more information on session event listeners, see ["Session Event Listeners"](#) on page 75-7.

This section describes session log options, including the following:

- [Log Types](#)
- [Log Output](#)
- [Log Level](#)
- [Logging SQL](#)
- [Logging Chained Exceptions](#)
- [Viewing TopLink Log Messages From the Application Server Control Console](#)

Log Types

TopLink supports the following types of logging:

- [TopLink Native Logging](#)
- [java.util Logging](#)
- [Server Logging](#)

For a non-CMP application, you can configure the log type using TopLink Workbench (see ["Using TopLink Workbench"](#) on page 77-5). For a CMP application, see ["Configuring Logging in a CMP Application"](#) on page 77-9.

TopLink Native Logging

TopLink native logging is the default session log type. It is provided by `oracle.toplink.logging.DefaultSessionLog`. [Example 75-1](#) shows a typical TopLink native log message.

You can configure TopLink native logging options using TopLink Workbench (see ["Using TopLink Workbench"](#) on page 77-5).

Example 75-1 Sample TopLink Log Message

```
[TopLink Info]: DATE
TIME-DatabaseSession(12345)-Thread(12345)-TopLink, version: Oracle TopLink - 10g (Build
031203)
[TopLink Config]: DATE
TIME-DatabaseSession(12345)-Thread(12345)-Connection(12345)-
```

```

connecting(DatabaseLogin(
    platform=>Oracle9Platform
    user name=> "username"
    datasource URL=> "jdbc:oracle:thin:@144.23.214.115:1521:toplink"
))
[TopLink Config]: DATE
TIME-DatabaseSession(12345)-Thread(12345)-Connection(12345)-
  Connected: jdbc:oracle:thin:@144.23.214.115:1521:toplink
  User: USERNAME
  Database: Oracle Version: Oracle9i Enterprise Edition - Production
  With the Partitioning, OLAP and Oracle Data Mining options
  JServer Release 9.2.0.3.0 - Production
  Driver: Oracle JDBC driver Version: 9.2.0.3.0
[TopLink Info]: DATE
TIME-DatabaseSession(12345)-Thread(12345)-loggingTestSession login
successful

```

java.util Logging

This type of logging makes TopLink conform to the `java.util.logging` package. It is provided by `oracle.toplink.logging.JavaLog`. Logging options are configured in the `<JRE_HOME>/lib/logging.properties` file. Messages are written to any number of destinations based on this configuration. [Example 75-2](#) shows a typical `java.util.logging` log message.

For more information on using `java.util.logging` package, see ["Configuring a Session to use java.util.logging Package"](#) on page 77-8.

Example 75-2 Sample java.util.logging Log Messages

```

Dec 9, 2003 2:05:05 PM oracle.toplink.loggingTestSession DatabaseSession(32603767) Thread(10)
INFO: TopLink, version: Oracle TopLink - 10g (10.0.3) Developer Preview (Build 031203)
Dec 9, 2003 2:05:07 PM oracle.toplink.loggingTestSession.connection DatabaseSession(32603767)
Connection(927929) Thread(10)
CONFIG: connecting(DatabaseLogin(
    platform=>Oracle9Platform
    user name=> "coredev8"
    datasource URL=> "jdbc:oracle:thin:@144.23.214.115:1521:toplink"
))
Dec 9, 2003 2:05:08 PM oracle.toplink.loggingTestSession.connection DatabaseSession(32603767)
Connection(927929) Thread(10)
CONFIG: Connected: jdbc:oracle:thin:@144.23.214.115:1521:toplink
      User: COREDEV8
      Database: Oracle Version: Oracle9i Enterprise Edition Release 9.2.0.3.0 - Production
      With the Partitioning, OLAP and Oracle Data Mining options
      JServer Release 9.2.0.3.0 - Production
      Driver: Oracle JDBC driver Version: 9.2.0.3.0
Dec 9, 2003 2:05:08 PM oracle.toplink.loggingTestSession DatabaseSession(32603767) Thread(10)
INFO: loggingTestSession login successful

```

Server Logging

Server logging is used to integrate TopLink logging with an application server log.

The TopLink runtime determines the server log type to use given the server platform you configure when you create your project (["Project Creation Overview"](#) on page 21-1).

For example, if your project uses the OC4J platform, TopLink uses the `oracle.toplink.platform.server.oc4j.OjdlLog`; if your project uses the BEA WebLogic platform, TopLink uses the `oracle.toplink.platform.server.wls.WlsLog`.

Log Output

If you are using TopLink native logging, you can configure TopLink to write log messages to a file or to the console (see ["Configuring Logging"](#) on page 77-4).

If you are using `java.util.logging`, TopLink writes log messages to the destinations you configure in the `<JRE_HOME>/lib/logging.properties` file (see ["Configuring a Session to use java.util.logging Package"](#) on page 77-8).

If you are using server logging, TopLink writes log messages to the application server's log file (there is no separate TopLink log file in this case).

Log Level

You can control the amount and detail of log output by configuring the log level (in ascending order of information) in the following way:

- **SEVERE**—Logs exceptions indicating TopLink cannot continue, as well as any exceptions generated during login. This includes a stack trace.
- **WARNING**—Logs exceptions that do not force TopLink to stop, including all exceptions not logged with severe level. This does not include a stack trace.
- **INFO** (default)—Logs the login/logout per sever session, including the user name. After acquiring the session, detailed information is logged.
- **CONFIG**—Logs only login, JDBC connection, and database information.
- **FINE**—Logs SQL (including thread information).
- **FINER**—Similar to warning. Includes stack trace.
- **FINEST**—Includes additional low level information
- **ALL**—Logs everything.

By default, TopLink logs at the `oracle.toplink.logging.SessionLog.INFO` level so that some information is logged by default.

At run time, set the log level using `Session` method `setLogLevel`, passing in one of the log level constants provided by `oracle.toplink.logging.SessionLog`.

Logging SQL

In a relational project, TopLink accesses the database using SQL strings that it generates internally. This feature enables applications to use the session methods or query objects without having to perform their own SQL translation.

If, for debugging purposes, you want to review a record of the SQL that is sent to the database, set the session log level to `oracle.toplink.logging.SessionLog.FINE`—the session will log all executed SQL to the session log.

Logging Chained Exceptions

The logging chained exception facility enables you to log causality when one exception causes another as part of the standard stack back-trace. If you build your applications with JDK 1.4, causal chains appear automatically in your logs.

Viewing TopLink Log Messages From the Application Server Control Console

You can view TopLink log messages using Oracle Enterprise Manager 10g.

If you are using TopLink native logging (to a file) or `java.util.logging` package, edit the `<OC4J_HOME>\toplink\config\TOPLINK.xml` file `log` element attribute

path to match the log output file you specified in your session (see ["Configuring Logging"](#) on page 77-4), or in the `<JRE_HOME>/lib/logging.properties` file (see ["Configuring a Session to use java.util.logging Package"](#) on page 77-8). [Example 75-3](#) shows the default `TOPLINK.xml` file:

Example 75-3 Default TOPLINK.xml File

```
<?xml version="1.0" encoding="UTF-8" ?>
<logs xmlns="http://www.oracle.com/ias/emcomponent/ojdl">
  <log path="toplink/config/toplink.log" componentId="TOPLINK">
    <logviewer LogType="ERROR" />
  </log>
</logs>
```

If you are using server logging, you do not need to configure the `<OC4J_HOME>\toplink\config\TOPLINK.xml` file: in this case, it is not used. Instead, all TopLink log messages are written to the OC4J log file (which you can also view from the application server control console).

For more information, see the *Oracle Application Server Administrator's Guide*.

Profiler

The TopLink session provides profiling API that lets you identify performance bottlenecks in your application (see ["Configuring a Performance Profiler"](#) on page 77-10). When enabled, the profiler logs a summary of the performance statistics for every query that the application executes.

TopLink allows you to measure application performance using the following tools:

- [TopLink Profiler](#)
- [Oracle Dynamic Monitoring System \(DMS\)](#)

TopLink Profiler

The TopLink profiler is a high-level logging service. Instead of logging SQL statements, the profiler logs a summary of each query you execute. The summary includes a performance breakdown of the query that lets you identify performance bottlenecks. The profiler also provides a report summarizing the query performance for an entire session.

Access profiler reports and profiles through the **Profile** tab in the TopLink Web client, or create your own application or applet to view the profiler logs. For more information, see ["Accessing the TopLink Profiler Results"](#) on page 11-3.

For more information, see ["Measuring Performance With the TopLink Profiler"](#) on page 11-2.

Oracle Dynamic Monitoring System (DMS)

Oracle DMS is a library that enables application and system developers to use a variety of DMS sensors to measure and export customized performance metrics for specific software components (called nouns).

TopLink includes DMS instrumentation in essential objects to provide efficient monitoring of runtime data in TopLink enabled applications, including both J2EE and non-J2EE applications.

By enabling DMS profiling in a TopLink application, you can collect and easily access run-time data that can help you with application administration tasks and performance tuning.

You can easily access DMS data at run time using a management application that supports the Java Management Extensions (JMX) API (see ["Accessing Oracle DMS Profiler Data Using JMX"](#) on page 11-7), or using any Web browser and the DMS Spy servlet (see ["Accessing Oracle DMS Profiler Data Using the DMS Spy Servlet"](#) on page 11-7).

For more information, see ["Measuring Performance With the Oracle Dynamic Monitoring System \(DMS\)"](#) on page 11-4.

Integrity Checker

When you log into a session, TopLink initializes and validates the descriptors you registered with it. By configuring the integrity checker, you can customize this validation process.

For more information, see ["Configuring the Integrity Checker"](#) on page 77-18.

Exception Handlers

Exception handlers allow any exception that occurs in a session to be caught and processed. Exception handlers can be used for debugging purposes, or to resolve database timeouts or failures.

To use exception handlers, register an implementor of the `oracle.toplink.exceptions.ExceptionHandler` interface with the session (see ["Configuring an Exception Handler"](#) on page 77-12).

If an exception occurs during a session operation, such as executing a query, the exception is passed to the exception handler. The exception handler can either rethrow the exception, or handle the exception and retry the operation. When handling exceptions, ensure that the following conditions are met:

- If you are performing a write query and you are within a transaction, you should not retry the operation.
- If you are performing a read query, you may retry the operation, and, if successful, return the query result.

If your exception handler cannot proceed, you should throw an appropriate application-specific exception.

For more information on the types of exceptions that TopLink can throw, see [Part V, "Troubleshooting a TopLink Application"](#).

Registering Descriptors

You use a session to perform persistence operations on the objects described by TopLink mapping metadata represented as a TopLink project (see ["Understanding Projects"](#) on page 20-1). Each session must therefore be associated with the descriptors of at least one TopLink project. You associate descriptors with a session by registering them with the session.

The preferred way to register descriptors with a session is to use TopLink Workbench to configure the session with a mapping project (see ["Configuring a Primary Mapping Project"](#) on page 77-2 and ["Configuring Multiple Mapping Projects"](#) on page 77-9).

Sessions and CMP

Although TopLink is an integral part of a J2EE application, in most cases the client does not interact with TopLink directly. Instead, TopLink features are invoked indirectly by way of EJB container callbacks.

In a CMP TopLink project, you do not explicitly create, configure, or acquire a session. The TopLink runtime creates, configures, acquires and uses a session itself internally. For more information, see ["Creating Session Metadata"](#) on page 2-21. Similarly, in a CMP TopLink project, how metadata is deployed depends on the CMP container and application server you use (see ["Deploying Metadata"](#) on page 2-21).

Sessions and Sequencing

An essential part of maintaining object identity is managing the assignment of unique values to distinguish one instance from another. For more information, see ["Projects and Sequencing"](#) on page 20-4.

Sequencing options you configure in a `sessions.xml` (or `project.xml`) file determine the type of sequencing that TopLink uses.

In a CMP project, you do not configure a `sessions.xml` file directly: in this case you must configure the sequence type in the `project.xml` file (see ["Configuring Sequencing at the Project Level"](#) on page 23-3).

In a non-CMP project, you can use session-level sequence configuration to override project-level sequence configuration, on a session-by-session basis, if required (see ["Configuring Sequencing at the Session Level"](#) on page 86-4).

After configuring the sequence type at the session (or project) level, for each descriptor you must also configure sequencing options for that descriptor to use sequencing (see ["Descriptors and Sequencing"](#) on page 26-9).

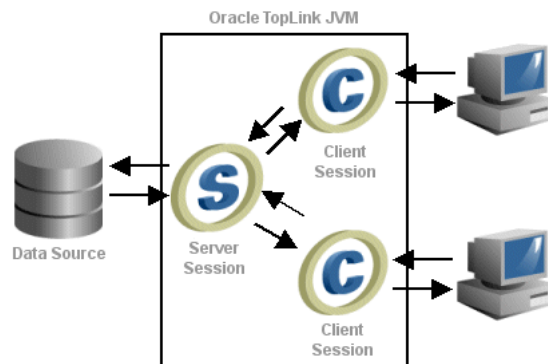
Server and Client Sessions

A server session manages the server side of client/server communications, providing shared resources, including a shared object cache and connection pools to a single data source.

A client session is a client-side communications mechanism that works together with the server session to provide the client/server connection. You acquire client sessions from a server session at run time as required. By default, a client session shares the session cache of its parent server session. Each client session serves one client. A client session communicates with the server session on behalf of the client application.

Each client session can have only one associated server session, but a server session can support any number of client sessions.

As [Figure 75–2](#) illustrates, together, the client session and server session provide a three-tier architecture that you can scale easily, by adding more client sessions. A server session is the most common TopLink session type because it supports this three-tier architecture that is common in enterprise applications. Because of this scalability, Oracle recommends that you use the three-tier architecture to build your TopLink applications.

Figure 75-2 Typical TopLink Server Session with Client Session Architecture

This section explains the advantages of using server sessions and client sessions in your TopLink application, including the following:

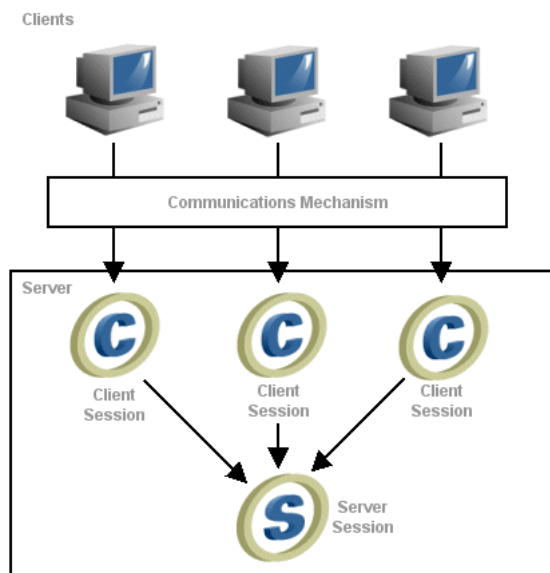
- [Three-Tier Architecture Overview](#)
- [Advantages of the TopLink Three-Tier Architecture](#)

For more information, see the following:

- ["Creating a Server Session"](#) on page 76-4
- ["Configuring Server Sessions"](#) on page 79-1
- ["Acquiring a Session from the Session Manager"](#) on page 78-3
- ["Acquiring a Client Session"](#) on page 78-6

Three-Tier Architecture Overview

In a TopLink three-tier architecture, client sessions and server sessions both reside on the server. Client applications access the TopLink application through a client session, and the client session communicates with the database using the server session.

Figure 75-3 Server Session and Client Session Usage

Advantages of the TopLink Three-Tier Architecture

Although the server session and the client session are two different session types, you can treat them as a single unit in most cases, because they are both required to provide three-tier functionality to the application. The server session provides the client session to client applications, and also supplies the majority of the session functionality.

This section discusses some of the advantages and general concepts associated with the TopLink three-tier design, including the following:

- [Shared Resources](#)
- [Providing Read Access](#)
- [Providing Write Access](#)
- [Security and User Privileges](#)
- [Concurrency](#)
- [Connection Allocation](#)

Shared Resources

The three-tier design enables multiple clients to share persistent resources. The server session provides its client sessions with a shared live object cache, read and write connection pooling, and parameterized named queries. Client sessions also share descriptor metadata.

You can use client sessions and server sessions in any application server architecture that allows for shared memory and supports multiple clients. These architectures can include HyperText Markup Language (HTML), Servlet, JavaServer Pages (JSP), Remote Method Invocation (RMI), Common Object Request Broker Architecture (CORBA), Web services, and EJB.

To support a shared object cache, client sessions must do the following:

- Implement any changes to the database with the TopLink unit of work.
- Share a common database login for reading (you can implement separate logins for writing).

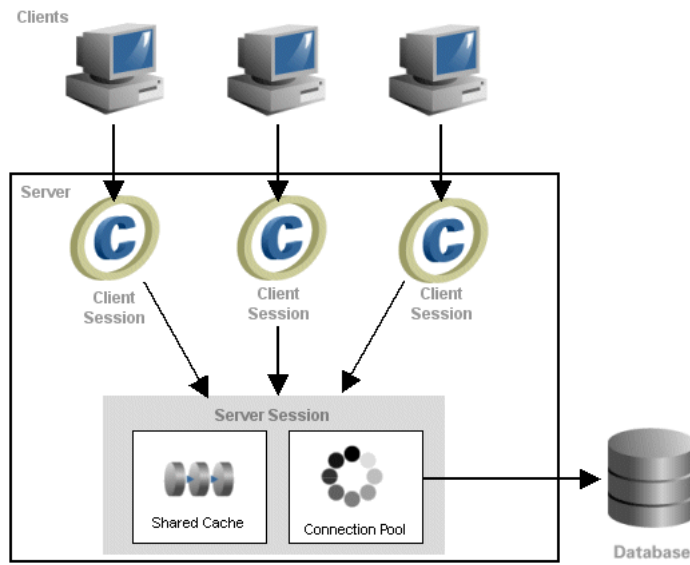
Providing Read Access

To read objects from the database, the client must first acquire a client session from the server session. Acquiring a client session gives the client access to the session cache and the database through the server session. The server session behaves as follows:

- If the object or data is in the session cache, then the server session returns the information back to the client.
- If the object or data is not in the cache, then the server session reads the information from the database and stores the object in the session cache. The objects are then available for retrieval from the cache.

Because a server session processes each client request in a separate thread, this enables multiple clients to access the database connection pool concurrently.

[Figure 75-4](#) illustrates how multiple clients read from the database using the server session.

Figure 75-4 Multiple Client Sessions Reading the Database Using the Server Session

To read objects from the database using a client session, do the following:

1. Acquire a Session from the Server:

```
Server server =
    (Server) SessionManager.getManager().getSession(
        sessionName, MyServerSession.class.getClassLoader()
    );
Session clientSession = (Session) server.acquireClientSession();
```

For more information, see ["Acquiring and Using Sessions at Run Time"](#) on page 78-1.

2. Use the Session object to perform read operations (for more information, see ["Understanding TopLink Queries"](#) on page 96-1 and ["Understanding TopLink Expressions"](#) on page 97-1).

Note: Oracle recommends that you do not use the Server session object directly to read objects from the database.

Providing Write Access

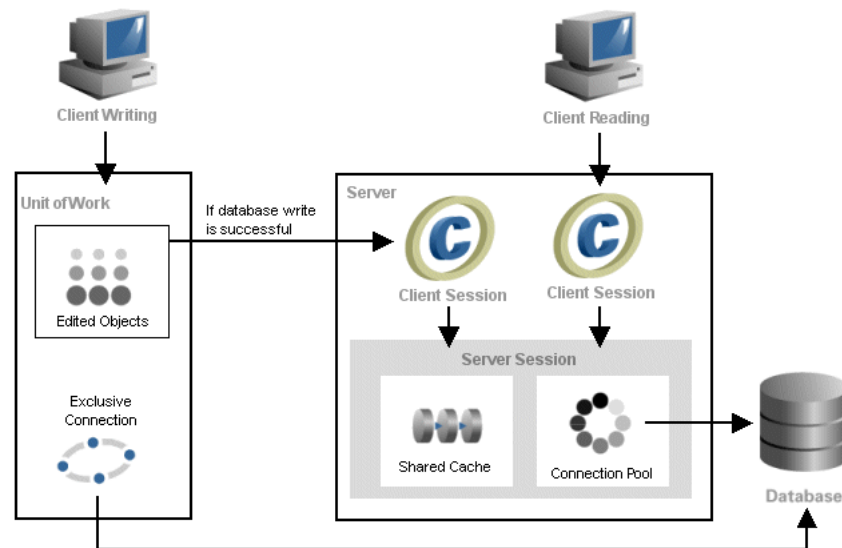
Because the client session disables all database modification methods, a client session cannot create, change, or delete objects directly. Instead, the client must obtain a unit of work from the client session to perform database modification methods.

To write to the database, the client acquires a client session from the server session and then acquires a unit of work within that client session. The unit of work acts as an exclusive transactional object space, and also ensures that any changes that are committed to the database also occur in the session cache.

Note: Although client sessions are thread-safe, do not use them to write across multiple threads. Multithread write operations from the same client session can result in errors and a loss of data. For more information, see ["Concurrency"](#) on page 75-18.

Figure 75–5 illustrates how to write to the database using a client session acquired from a server session.

Figure 75–5 Writing with Client Sessions and Server Sessions



To write to the database using a unit of work, use this procedure:

1. Acquire a session from the server session:

```
Server server =
    (Server) SessionManager.getManager().getSession(
        sessionName, MyServerSession.class.getClassLoader()
    );
Session clientSession = (Session) server.acquireClientSession();
```

For more information, see ["Acquiring and Using Sessions at Run Time"](#) on page 78-1.

2. Acquire a `UnitOfWork` object from the `Session` object.

```
UnitOfWork uow = clientSession.acquireUnitOfWork();
```

For more information, see ["Unit of Work Sessions"](#) on page 75-19.

3. Use the unit of work to perform the required updates and then commit the `UnitOfWork`.

For more information, see the following:

- ["Understanding TopLink Queries"](#) on page 96-1
- ["Understanding TopLink Expressions"](#) on page 97-1)
- ["Understanding TopLink Transactions"](#) on page 100-1

Security and User Privileges

You can define several different server sessions in your application to support users with different data access rights. For example, your application may serve a group called "Managers," who has access rights to salary information, and a group called "Employees," who do not. Because each session you define in the `sessions.xml` file

has its own login information, you can create multiple sessions, each with its own login credentials, to meet the needs of both of these groups.

When you use internal TopLink connection pools (see ["Connection Pools"](#) on page 84-7), each server session provides a read connection pool and a write connection pool. All read queries use connections from the read connection pool and all queries that write changes to the data store use connections from the write connection pool. This ensures that connections for one session are kept separate from the connections used in another.

To further isolate users from one another, you can use an isolated session: a special type of client session that provides its own session cache isolated from the shared object cache of its parent server session to provide improved user-based security, or to avoid caching highly volatile data. For more information, see ["Isolated Client Sessions"](#) on page 75-19.

Concurrency

The server session supports concurrent clients by providing each client with a dedicated thread of execution. Dedicated threads enable clients to operate asynchronously—that is, client processes execute as they are called and do not wait for other client processes to complete.

TopLink safeguards thread safety with a concurrency manager. The concurrency manager ensures that no two threads interfere with each other when performing operations such as creating new objects, executing a transaction on the database, or accessing value holders.

For more information about handling concurrency issues, see ["Handling Stale Data"](#) on page 90-6.

Connection Allocation

When you instantiate the server session, it creates a pool of data source connections. The session then manages the connection pool based on your session configuration, and shares the connections among its client sessions. When the client session releases the connection, the server session recovers the connection and makes it available to other client processes. Reusing connections reduces the number of connections required by the application and allows a server session to support a larger number of clients.

The server session provides connections to client sessions as needed. By default, the server session does not allocate a data source connection for a client session until a transaction starts (a lazy data source connection). Alternatively, you can acquire a client session that allocates a connection immediately (see ["Acquiring a Client Session that Does Not Use Lazy Connection Allocation"](#) on page 78-9).

The server session allocates read connections from its read connection pool to all client sessions. If your application requires multiple read security levels then you must use multiple server sessions or TopLink isolated sessions (see ["Isolated Client Sessions"](#) on page 75-19).

The server session also supports multiple write connection pools and nonpooled connections. By default, all client sessions use the default write connection pool. However, if your application requires multiple security levels or user logins for write access, then you can use multiple write connection pools. You can configure a client session to use a specific write connection pool or nonpooled connection when it is acquired (see ["Acquiring a Client Session that Uses a Named Connection Pool"](#) on page 78-8). This connection is only used for writes, not reads (reads still go through the server session read connection pool).

For more information, see the following:

- ["Internal Connection Pools"](#) on page 84-7
- ["External Connection Pools"](#) on page 84-8

Unit of Work Sessions

The unit of work ensures that the client edits objects in a separate object transaction space. This feature lets clients perform object transactions in parallel. When transactions are committed, the unit of work makes any required changes in the database, and then merges the changes into the shared TopLink session cache. The modified objects are then available to all other users.

For information on creating, configuring, and using a unit of work, see ["Understanding TopLink Transactions"](#) on page 100-1.

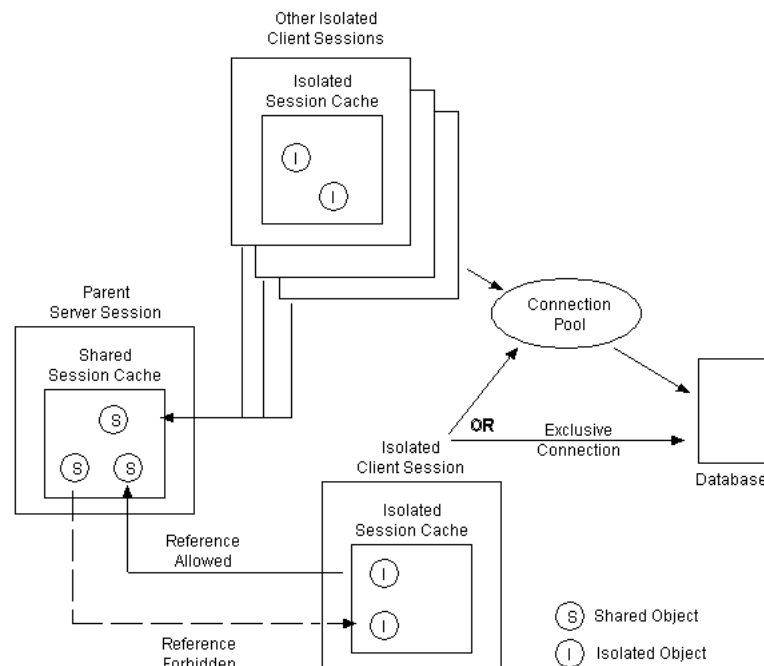
Isolated Client Sessions

An isolated client session is a special type of client session that provides its own session cache. This session cache is isolated from the shared session cache of its parent server session.

If in your TopLink project you configure all classes as isolated (see ["Configuring Cache Isolation at the Project Level"](#) on page 22-16), or one or more classes as isolated (see ["Configuring Cache Isolation at the Descriptor Level"](#) on page 28-37), then all client sessions that you acquire from a parent server session will be isolated client sessions.

[Figure 75-6](#) illustrates the relationship between a parent server session's shared session cache and its child isolated client sessions.

Figure 75-6 Isolated Client Sessions



Each isolated client session owns an initially empty cache and identity maps used exclusively for isolated objects that the isolated client session accesses while it is active.

The isolated client session's isolated session cache is discarded when the isolated client session is released.

When you use an isolated client session to read an isolated class, the client session reads the isolated object directly from the database and stores it in that client session's isolated session cache. When you use the client session to read a shared class, the client session reads the shared object from the parent server session's shared session cache. If the shared object is not in the parent server session's shared session cache, it will read it from the database and store it in the parent server session's shared session cache.

Isolated objects in an isolated client session's isolated session cache may reference shared objects in the parent server session's shared session cache, but shared objects in the parent server session's shared session cache cannot reference isolated objects in an isolated client session's isolated session cache.

Note: You cannot define mappings from shared classes to isolated classes. If using CMP 2.0, you also cannot define references from isolated EJB to shared EJB.

Client sessions can access the data source using a connection pool, or an exclusive connection. To use an exclusive connection, acquire the isolated client session using a `ConnectionPolicy` (see ["Acquiring a Client Session That Uses Exclusive Connections"](#) on page 78-7). Using an exclusive connection provides improved user-based security for reads and writes. Named queries can also use an exclusive connection (see ["Configuring Named Query Advanced Options"](#) on page 28-24).

Note: If an isolated session contains an exclusive connection, you must release the session when you are finished using it. Relying on the finalizer to release the connection when the session is garbage collected may cause errors when dealing with Java Transaction API (JTA) transactions.

Use isolated client sessions to do the following:

- avoid caching highly volatile data in the shared session cache
- achieve serializable transaction isolation (see ["Isolated Client Session Cache"](#) on page 102-29)
- use the Oracle Virtual Private Database (VPD) feature in your TopLink-enabled application (see ["Isolated Client Sessions and Oracle Virtual Private Database \(VPD\)"](#) on page 75-20)

For more information, see the following:

- ["Isolated Client Session Limitations"](#) on page 75-23
- ["Acquiring an Isolated Client Session"](#) on page 78-7
- ["Configuring Exclusive Isolated Client Sessions for Virtual Private Database"](#) on page 80-1

Isolated Client Sessions and Oracle Virtual Private Database (VPD)

Oracle9i Database Server (and later) provides a server-enforced, fine-grained access control mechanism called Virtual Private Database (VPD). VPD ties a security policy to a table by dynamically appending SQL statements with a predicate to limit data access

at the row level. You can create your own security policies, or use Oracle's custom implementation of VPD called Oracle Label Security (OLS). For more information on VPD and OLS, see:

<http://www.oracle.com/technology/deploy/security/index.html>.

To use the Oracle Database VPD feature in your TopLink-enabled application, use isolated client sessions.

Any class that maps to a table that uses VPD must have the descriptor configured as isolated (see "[Configuring Cache Isolation at the Descriptor Level](#)" on page 28-37).

When you use isolated client sessions with VPD, you typically use exclusive connections (see "[Acquiring a Client Session That Uses Exclusive Connections](#)" on page 78-7).

To support VPD, you are responsible for implementing session event handlers that the TopLink runtime invokes during the isolated client session life cycle (see "[Isolated Client Session Life Cycle](#)" on page 75-22). The session event handler you must implement depends on whether or not you are using Oracle Database proxy authentication (see "[VPD With Oracle Database Proxy Authentication](#)" on page 75-21 and "[VPD Without Oracle Database Proxy Authentication](#)" on page 75-21).

For information, see "[Configuring Exclusive Isolated Client Sessions for Virtual Private Database](#)" on page 80-1.

VPD With Oracle Database Proxy Authentication

If you are using Oracle Database proxy authentication ("[Oracle Database Proxy Authentication](#)" on page 84-5), you must implement a session event handler for the following session events:

- `noRowsModifiedSessionEvent` (see "[NoRowsModifiedSessionEvent Event Handler](#)" on page 80-3)

By using Oracle Database proxy authentication, you can set up VPD support entirely in the database. That is, rather than making the isolated client session execute SQL (see "[PostAcquireExclusiveConnection Event Handler](#)" on page 80-1 and "[PreReleaseExclusiveConnection Event Handler](#)" on page 80-2), the database performs the required setup in an after login trigger using the proxy `session_user`.

VPD Without Oracle Database Proxy Authentication

If you are not using Oracle Database proxy authentication, you must implement session event handlers for the following session events:

- `postAcquireExclusiveConnection` (see "[PostAcquireExclusiveConnection Event Handler](#)" on page 80-1): used to perform VPD setup at the time TopLink allocates a dedicated connection to an isolated session and before the isolated session user uses the connection to interact with the database.
- `preReleaseExclusiveConnection` (see "[PreReleaseExclusiveConnection Event Handler](#)" on page 80-2): used to perform VPD cleanup at the time the isolated session is released and after the user is finished interacting with the database.
- `noRowsModifiedSessionEvent` (see "[NoRowsModifiedSessionEvent Event Handler](#)" on page 80-3)

In your implementation of these handlers, you obtain the required user credentials from the `ConnectionPolicy` associated with the session (see "[Acquiring a Client Session that Uses Connection Properties](#)" on page 78-8).

Isolated Client Session Life Cycle

This section provides an overview of the key phases in the life cycle of an isolated session, including the following:

- Setup required before using an isolated session
- Interaction among isolated session objects
- Clean-up required after using an isolated session

To enable the life cycle of an isolated session, use this procedure:

1. Prepare VPD configuration in the database.
2. Configure your project and session:
 - Designate descriptors as isolated (see ["Configuring Cache Isolation at the Descriptor Level"](#) on page 28-37).
 - Configure your server session to allocate exclusive connections (see ["Configuring Connection Policy"](#) on page 77-19).
 - Implement session event listeners for the required connection events
 - If you are using Oracle Database proxy authentication (see ["Oracle Database Proxy Authentication"](#) on page 84-5), see ["NoRowsModifiedSessionEvent Event Handler"](#) on page 80-3.
 - If you are not using Oracle Database proxy authentication, see ["PostAcquireExclusiveConnection Event Handler"](#) on page 80-1, ["PreReleaseExclusiveConnection Event Handler"](#) on page 80-2, and ["NoRowsModifiedSessionEvent Event Handler"](#) on page 80-3

Note: You must add these session event listeners to the server session from which you acquire your isolated client session. You cannot add them to the isolated client session itself. For more information, see ["Configuring Session Event Listeners"](#) on page 77-17

- Implement exception handlers for the appropriate exceptions (see ["ValidationException Handler"](#) on page 80-3).
3. Acquire an isolated session:
 - If you are using Oracle Database proxy authentication (see ["Oracle Database Proxy Authentication"](#) on page 84-5):

```
Session myIsolatedClientSession = server.acquireClientSession();
```

Because you configured one or more descriptors as isolated, `myIsolatedClientSession` is an isolated session with an exclusive connection.

- If you are not using Oracle Database proxy authentication:

```
ConnectionPolicy myConnPolicy =  
    (ConnectionPolicy)server.getDefaultConnectionPolicy().clone();  
myConnectionPolicy.setProperty("credentials", myUserCredentials);  
Session myIsolatedClientSession =  
    server.acquireClientSession(myConnectionPolicy);
```

Set the user's credentials as appropriate properties on `myConnectionPolicy`. Because you configured one or more descriptors as

isolated, `myIsolatedClientSession` is an isolated session with an exclusive connection.

The TopLink runtime raises a `SessionEvent.PostAcquireExclusiveConnection` event handled by your `SessionEventListener` (see ["PostAcquireExclusiveConnection Event Handler"](#) on page 80-1).

4. Use `myIsolatedClientSession` to interact with the database.

If the TopLink runtime raises a `SessionEvent.NoRowsModified` event, it is handled by your `SessionEventListener` (see ["NoRowsModifiedSessionEvent Event Handler"](#) on page 80-3).

5. When you are finished using `myIsolatedClientSession`, release the isolated session:

```
myIsolatedClientSession.release();
```

The TopLink runtime prepares to destroy the isolated cache and to close the exclusive connection associated with this isolated session.

The TopLink runtime raises a `SessionEvent.PreReleaseExclusiveConnection` event handled by your `SessionEventListener` (see ["PreReleaseExclusiveConnection Event Handler"](#) on page 80-2).

6. Repeat steps 3 to 5 (as required) until the application exits.

Isolated Client Session Limitations

For the purposes of security as well as efficiency, observe the limitations described in the following section, when you use isolated client sessions in your TopLink three-tier application:

- [Mapping](#)
- [Inheritance](#)
- [Caching and Cache Coordination](#)
- [Sequencing](#)
- [CMP](#)
- [Transactions and JTA](#)

Mapping

Consider the following mapping and relationship restrictions when using isolated sessions with your relational model:

- Isolated objects may be related to shared objects, but shared objects cannot have any relationships with isolated objects.
- If a table has a VPD security policy associated with it, then the class mapped to that table must be isolated.
- If one of the tables in a multiple table mapping is isolated, then the main class must also be isolated.

The TopLink runtime enforces these restrictions during descriptor initialization.

Inheritance

Aggregates and aggregate mappings inherit the isolated configuration of their parents.

If a class is isolated, then all inheriting classes should be isolated. Otherwise, if you relate a shared class to a shared superclass with isolated subclasses, it is possible that some of the isolated subclasses will lose object identity when the isolated session is released.

To give you the flexibility to mix shared and isolated classes, the TopLink runtime does not enforce these restrictions during descriptor initialization. If you wish to mix shared and isolated classes in your inheritance hierarchy, then you must be prepared to deal with this possible loss of object identity.

Caching and Cache Coordination

Isolated classes are never loaded into the shared cache of a parent server session. Isolated classes cannot be used with cache coordination.

Sequencing

Oracle recommends that you do not configure a sequencing object as isolated. TopLink does not access sequencing objects using the isolated session's dedicated connection, and so the sequence values are not available to the isolated session.

CMP

For CMP 2.0, relationships between isolated and shared data is not allowed. This is because of the relationship maintenance requirements of having bidirectional references for all relationships.

Transactions and JTA

Oracle recommends that you explicitly release an isolated session when you are finished using it, rather than wait for the Java garbage collector to invoke the finalizer. The finalizer is provided as a last resort: waiting for the garbage collector may cause errors when dealing with a JTA transaction.

Historical Client Sessions

By default, a session represents a view of the most current version of objects, and when you execute a query in that session, it returns the most current version of selected objects.

If your data source maintains past versions of objects, you can configure TopLink to access this historical data so that you can express read queries conditional on how your objects are changing over time. You can also do the following:

- Make series of queries relative to any point in time—not just the time of the first query.
- Provide read consistency so that a series of read operations or report queries all execute as if at the same time.
- Use the `mergeClone` method to provide deep recovery of an object by passing in a past version of it.

In addition, you can express query selection criteria as either of the following:

- A condition at a past time: for example, "employees who used to..."
- A change over time: for example, "employees who recently..."

For more information, see the following:

- ["Historical Client Session Limitations"](#) on page 75-25
- ["Configuring Historical Client Sessions"](#) on page 81-1
- ["Acquiring a Historical Client Session"](#) on page 78-7
- ["Historical Queries"](#) on page 96-20.

Historical Client Session Limitations

The `HistoryPolicy` provides a very flexible means of accommodating a wide variety of historical schemas. However, be aware of the following restrictions:

- You cannot use the `HistoryPolicy`, if your design combines both current and historical data in a single schema.
- You cannot use historical sessions, nor historical queries, with EJB entity beans.
- `TopLink` assumes that the current version of an object corresponds to the row in the historical table whose row end field is `NULL`.
- You cannot directly map the start and end fields of a history table because they do not exist in the regular schema.
- You cannot query on ranges of historical objects, only as of a specific point in time.

Session Broker and Client Sessions

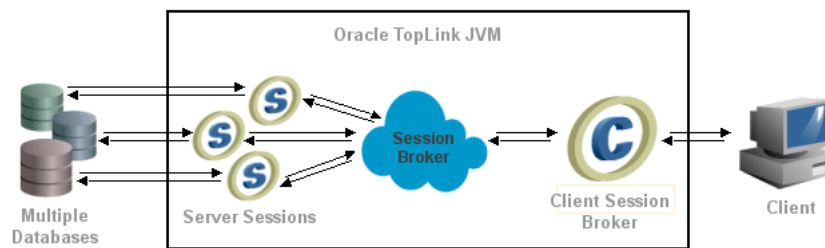
The **TopLink session broker** is a mechanism that enables client applications to transparently access multiple databases through a single `TopLink` session.

The `TopLink` session broker enables client applications to access two or more databases through a single session. If your application stores objects in multiple databases, the session broker, which provides seamless communication for client applications, enables the client to view multiple databases as if they were a single database.

When a three-tier session broker application uses server sessions to communicate with the database, clients require a client session to access the database. Similarly, when you implement a session broker, the client requires a *client session broker* to access the database.

A **client session broker** is a collection of client sessions, one from each server session associated with the session broker. When a client acquires a client session broker, the session broker collects one client session from each associated server session, and wraps the client sessions so that they appear to be a single client session to the client application.

As [Figure 75–7](#) illustrates, a session broker connects to the databases through two or more server sessions or database sessions.

Figure 75–7 TopLink Session Broker with Server Session Architecture

This section explains the following:

- [Session Broker Architecture](#)
- [Committing a Transaction with a Session Broker](#)
- [Session Broker Session Limitations](#)
- [Session Broker Alternatives](#)

For information, see:

- ["Creating Session Broker and Client Sessions"](#) on page 76-6
- ["Configuring Session Broker and Client Sessions"](#) on page 82-1
- ["Acquiring a Session from the Session Manager"](#) on page 78-3
- ["Acquiring a Client Session"](#) on page 78-6

Session Broker Architecture

As [Figure 75–7](#) illustrates, a session broker contains a broker object that acts as an intermediary between the application and the multiple sessions added to the session broker.

To construct a session broker, use TopLink Workbench to modify your `sessions.xml` file as follows:

1. Define two or more sessions (of the same type, either server sessions or database sessions).
2. Define a session broker.
3. Add the sessions to the session broker.

When you use `SessionManager` method `getSession(sessionBrokerName)` where `sessionBrokerName` is the name of the session broker you defined, the session manager returns the corresponding session broker session (call it `mySessionBroker`) that contains an instance of each of the sessions you added to it. When you use `mySessionBroker` method `login`, it logs into each defined session. Thereafter, you use `mySessionBroker` as you would any other session: TopLink transparently handles access to the multiple databases.

In the case of a three-tier architecture where the session broker contains two or more server sessions, you use session broker method `acquireClientSessionBroker` to acquire a single client session that lets you query across all the data sources managed by the various server sessions. You use this client session as you would any other client session.

Committing a Transaction with a Session Broker

By default, when you commit a transaction with a session broker session, a two-stage commit is performed.

Ideally, you should incorporate a JTA external transaction controller in order to benefit from its two-phase commit.

Committing a Session with a JTA Driver: Two-Phase Commits

If you use a session broker, incorporate a JTA external transaction controller wherever possible. The external transaction controller provides a *two-phase commit*, which passes the SQL statements that are required to commit the transaction to the JTA driver. The JTA driver handles the entire commit process.

JTA guarantees that the transaction commits or rolls back completely, even if the transaction involves more than one database. If the commit operation to any one database fails, then all database transactions roll back. The two-phase commit operation is the safest method available to commit a transaction to the database.

Two-phase commit support requires integration with a compliant JTA driver.

Committing a Session Without a JTA Driver: Two-Stage Commits

If there is no JTA driver available, then the session broker provides a *two-stage commit* algorithm. A two-stage commit differs from a two-phase commit in that it guarantees data integrity only up to the point of the final commit of the transaction. If the SQL script executes successfully on all databases, but the commit operation then fails on one database, only the database that experiences the commit failure rolls back.

Although unlikely, this scenario is possible. As a result, if your system does not include a JTA driver and you use a two-stage commit, build a mechanism into your application to deal with this type of potential problem.

Session Broker Session Limitations

Although the session broker is a powerful tool that lets you use data that is distributed across multiple databases from a single application, it has some limitations:

- It may not meet the needs of your particular distributed data application (see ["Session Broker Alternatives"](#) on page 75-28).
- You cannot split multiple table descriptors across databases.
- Each class must reside on only one database.
- You cannot use joins through expressions across databases.
- Many-to-many join tables must reside on the same database as the target object (See ["Many-to-Many Join Tables and Direct Collection Tables"](#) on page 75-27 for a work-around for this limitation).

Many-to-Many Join Tables and Direct Collection Tables

By default, TopLink assumes that many-to-many and direct collection tables are on the same database as the source object. If they are on a different database, then you must configure the mapping's session name using `ManyToManyMapping` or `DirectCollectionMapping` method `setSessionName` as [Example 75-4](#) illustrates.

Note that a many-to-many join table must still reside on the same database as the target object.

Example 75–4 Using Mapping `setSessionName` in a Descriptor Amendment Method

```
public void addToDescriptor(ClassDescriptor descriptor) {  
    descriptor.getMappingForAttributeName("projects").setSessionName("branch-database");  
}
```

To work around this problem for data-level queries, use the `DatabaseQuery` method `setSessionName`.

Session Broker Alternatives

When evaluating whether or not to use a session broker in your application, consider the following alternatives:

- [Database Linking](#)
- [Multiple Sessions](#)

Database Linking

Most enterprise databases, such as the Oracle Database, support linking other databases on the database server. This allows querying and two-phase commit across linked databases. Using the session broker is not the same as linking databases. If your database allows linking, Oracle recommends that you use that functionality to provide multiple database access instead of using a session broker.

Multiple Sessions

An alternative to the session broker is to use multiple sessions to work with multiple databases:

- If the data on each database is unrelated to data on the other databases, and relationships do not cross database boundaries, then you can create a separate session for each database. For example, you might have individual databases and associated sessions dedicated to each department.

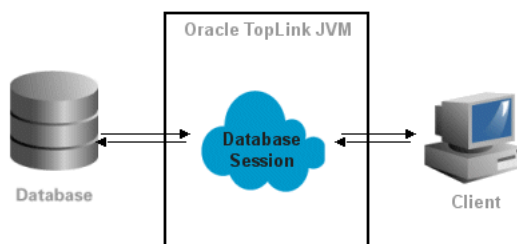
This arrangement requires that you to manage each session manually and ensure that the class descriptors for your project reside in the correct session.

- You can use additional sessions to house a standard batch job. In this case, you can create two or more sessions on the same database. In addition to the main session that supports client queries, you create other sessions that support batch inserts at low-traffic times in your system. This lets you maintain the client cache.

Database Sessions

A **database session** provides a client application with a single data source connection, for simple, standalone applications in which a single connection services all data source requests for one user.

Figure 75–8 TopLink Database Session Architecture



A database session is the simplest session TopLink offers. It provides both client and server communications and supports only a single client and a single database connection. It is suitable for simple applications or 2-tier applications.

Note: Oracle does not recommend using this session type in a 3-tier application because it is not as flexible or scalable as a server and client session. Oracle recommends that you use server sessions and client sessions (see ["Server and Client Sessions"](#) on page 75-13). Applications that are built using database sessions may be difficult to migrate to a scalable architecture in the future.

A database session contains and manages the following information:

- An instance of **Project** and **DatabaseLogin**, which store database login and configuration information
- The JDBC connection and the database access
- The descriptors for each of the application persistent classes
- Identity maps that maintain object identity and act as a cache

For more information, see the following:

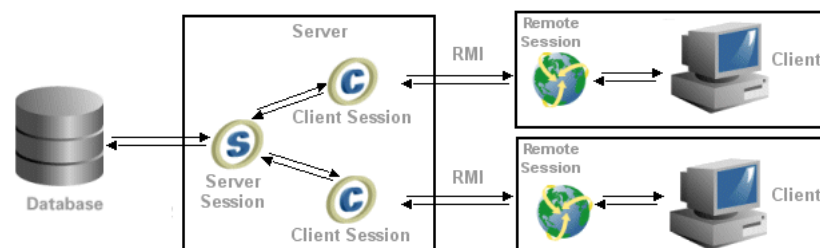
- ["Creating Database Sessions"](#) on page 76-4
- ["Configuring Database Sessions"](#) on page 76-8
- ["Acquiring a Session from the Session Manager"](#) on page 78-3

Remote Sessions

A **remote session** is a client-side session that communicates over RMI with a corresponding client session and server session on the server-side. Remote sessions handle object identity and marshalling and unmarshalling between client-side and server-side.

A remote session resides on the client rather than the TopLink server. The remote session does not replace the client session; rather, a remote session requires a client session to communicate with the server session.

Figure 75–9 Typical TopLink Server Session with Remote Session Architecture



The remote session provides a full TopLink session, complete with a session cache, on the client system. TopLink manages the remote session cache and enables client applications to execute operations on the server.

A remote session offers database access to clients that do not reside on the server. The remote session resides on the client and connects by way of RMI to a corresponding client session, which, in turn, connects to its server session on the server.

This section describes the following:

- [Architectural Overview](#)
- [Remote Session Concepts](#)

For more information, see "[Creating Remote Sessions](#)" on page 76-10.

Architectural Overview

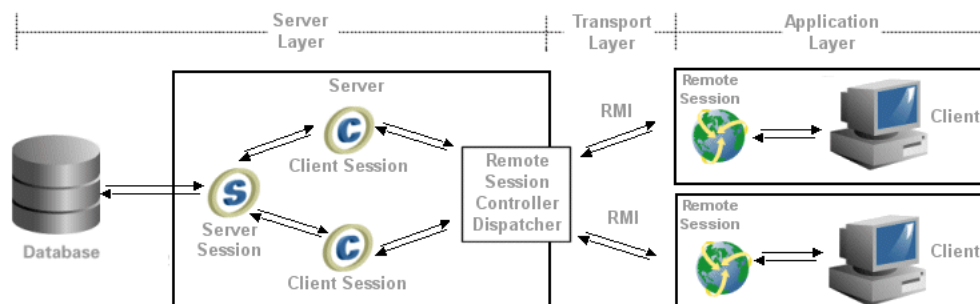
As [Figure 75–10](#) illustrates, the remote session model consists of the following layers:

- The application layer—a client-side application talking to a remote session
- The transport layer—a communication layer, RMI or RMI-IIOP
- The server layer—a TopLink session communicating with a database

The request from the client application to the server travels down through the layers of a distributed system. A client that makes a request to the server session uses the remote session as a conduit to the server session. The client references the remote session, and the remote session forwards a request to the server session through the transport layer.

At run time, the remote session builds its knowledge base by reading descriptors and mappings from the server side as they are needed. These descriptors and mappings are lightweight, because not all information is passed on to the remote session. The information needed to traverse an object tree and to extract primary keys from the given object is passed with the mappings and descriptors.

Figure 75–10 An Architectural Overview of the Remote Session



Application Layer

The application layer includes the application client and the remote session. The remote session is a subclass of `Session` and maintains all the public protocols of the session, giving the appearance of working with the corresponding client session.

The remote session maintains its own identity map and a project of all the descriptors read from the server. If the remote session can handle a request by itself, the request is not passed to the server. For example, a request for an object that is in the remote session cache is processed by the remote session. However, if the object is not in the remote session cache, the request passes to the server session.

Transport Layer

The transport layer is responsible for carrying the semantics of the invocation. It is a layer that hides all the protocol dependencies from the application and server layers.

The transport layer includes a remote connection that is an abstract entity, through which all requests to the server are forwarded. Each remote session maintains a single remote connection that marshals and unmarshals all requests and responses on the client side.

The remote session supports communications over RMI.

Server Layer

The server layer includes a remote session controller dispatcher and a TopLink sessions: [Figure 75–10](#) illustrates a three-tier server and its client sessions. The remote session controller dispatcher is an interface between the session and transport layers: it marshals and unmarshals all responses and requests between the sessions on the server and their corresponding remote sessions on the client.

Remote Session Concepts

When using remote sessions, consider the following:

- [Securing Remote Session Access](#)
- [Queries](#)
- [Refreshing](#)
- [Indirection](#)
- [Cursored Streams](#)
- [Unit of Work](#)

Securing Remote Session Access

The remote session represents a potential security risk because it requires you to register a remote session controller dispatcher as a service that anyone can access. This can expose the entire database to nonprivileged access.

To reduce this threat, run a server manager as a service to hold the remote session controller dispatcher. All the clients must then communicate through the server manager, which implements the security model for accessing the remote session controller dispatcher.

On the client side, the user requests the remote session controller dispatcher. The manager returns a remote session controller dispatcher only if the user has access rights according to the security model built into the server manager.

To access the system, the remote session controller dispatcher on the client side creates a remote connection, and acquires a remote session from the remote connection. The API for the remote session is the same as for the session, and there is no user-visible difference between working on a session or a remote session.

Queries

Read queries are publicly available on the client side, but queries that modify objects must be performed using the unit of work.

Refreshing

Calling refresh methods on the remote session causes database read operations, and may also cause cache updates if the data being refreshed is modified in the database. This can lead to poor performance.

To improve performance, configure refresh methods to run against the server session cache, by configuring the descriptor to always remotely refresh the objects in the cache on all queries. This technique ensures that all queries against the remote session refresh the objects from the server session cache, without the database access.

Cache hits on remote sessions still occur on read object queries based on the primary keys. To avoid this, disable the remote session cache hits on read object queries based on the primary key.

For more information, see ["Configuring Cache Refreshing"](#) on page 28-27.

Indirection

The remote session supports indirection objects. An indirection object is a value holder that can be invoked remotely on the client side. When invoked, the value holder first checks to see if the requested object exists on the remote session. If not, then the associated value holder on the server is instantiated to get the value that is then passed back to the client. Remote value holders are used automatically; the application's code does not change.

Cursored Streams

A remote session supports both cursored streams and scrollable cursors.

For more information, see ["Stream and Cursor Query Results"](#) on page 96-8.

Unit of Work

Use a unit of work acquired from the remote session to modify objects on the database. A unit of work acquired from the remote session offers the user the same functionality as a unit of work acquired from the client session or the database session.

Sessions and the Cache

Server, database, isolated, and historical sessions include an identity map that maintains object identity, and acts as a cache.

This section explains how the cache differs between the following sessions:

- [Server and Database Session Cache](#)
- [Isolated Session Cache](#)
- [Historical Session Cache](#)

For more information, see ["Understanding the Cache"](#) on page 90-1.

Server and Database Session Cache

When a server or database session reads objects from the database, it instantiates them and stores them in its identity map (cache). When the application subsequently queries for the same object, TopLink returns the object in the cache, rather than read the object from the database again.

This cache plays an important role in the performance of your application.

In the case of a server session, all client sessions acquired from it share the server session's cache.

To define how the cache manages objects, specify a strategy for cache management in TopLink Workbench.

Isolated Session Cache

When an isolated session reads an object, whose descriptor is configured as isolated, that object is instantiated and stored in the isolated session's cache only—it is not stored in the parent server session's shared object cache. Objects in the isolated session's cache may reference objects in the parent server session's shared object cache, but objects in the parent server session's shared object cache can never reference objects in the isolated session's cache.

Historical Session Cache

When a historical session reads objects, it does so only from its static, read-only cache, which is populated with all objects as of a specified time.

Understanding the Session API

The session API is defined by the following interfaces:

- `oracle.toplink.sessions.Session`
- `oracle.toplink.sessions.DatabaseSession`
- `oracle.toplink.sessions.UnitOfWork`
- `oracle.toplink.threetier.Server`

These APIs are used at run time to access objects and the data source. Always use the session public interfaces, not the corresponding implementation classes.

You should use the `Session` interface when reading and querying with any of client sessions, session brokers, isolated client sessions, historical client sessions, remote sessions, and database sessions.

You should use the `UnitOfWork` interface for all units of work acquired from any type of session.

You should use the `Server` interface to configure and acquire a client session from a `Server` session.

The `DatabaseSession` interface can be used for a database session.

Typically, you define server sessions, database sessions, and session broker sessions in a `sessions.xml` file and acquire them at run time using the `SessionManager`. You can also acquire a server session or database session from a `Project`. The only session that should ever be instantiated directly is the `SessionBroker`, and only when not using the `SessionManager`.

You acquire a client session from a server session.

You can also acquire a client session broker from a session broker composed of server sessions.

You acquire a unit of work from any session instance, client session broker, or session broker which contains `DatabaseSession` instances.

[Example 75–5](#) illustrates the session interfaces that derive from `oracle.toplink.sessions.Session` interface.

Example 75–5 Session Interface Inheritance Hierarchy

```
oracle.toplink.sessions.Session
    oracle.toplink.sessions.DatabaseSession
        oracle.toplink.threetier.Server
```

`oracle.toplink.sessions.UnitOfWork`

Creating Sessions

This chapter explains how to create TopLink sessions, including the following:

- [Session Creation Overview](#)
- [Creating a Sessions Configuration](#)
- [Configuring a Sessions Configuration](#)
- [Creating a Server Session](#)
- [Creating Session Broker and Client Sessions](#)
- [Creating Database Sessions](#)
- [Creating Remote Sessions](#)

For information on the various types of session available, see ["Session Types"](#) on page 75-1.

Session Creation Overview

Each TopLink session is contained within a sessions configuration (`sessions.xml`) file. You can create a sessions configuration using TopLink Workbench or Java code. Oracle recommends that you use TopLink Workbench to create and manage your sessions (see ["Creating a Sessions Configuration"](#) on page 76-1).

Alternatively, you can create sessions in Java. For more information on creating sessions in Java, see Oracle TopLink API Reference.

After you create a session, you must configure its various options (see ["Configuring a Session"](#) on page 77-1). After configuring the session, you can use it in your application to manage persistence (see ["Acquiring and Using Sessions at Run Time"](#) on page 78-1).

Creating a Sessions Configuration

TopLink Workbench lets you create session instances and save them in the `sessions.xml` file. It represents the `sessions.xml` file as a sessions configuration. Individual session instances are contained within the sessions configuration. You can create multiple sessions configurations, each corresponding to its own uniquely named `sessions.xml` file.

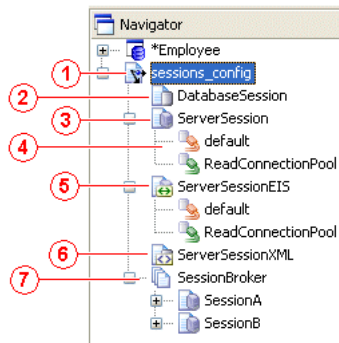
Oracle recommends that you use TopLink Workbench to create and manage sessions. It is the most efficient and flexible approach to session management. For more information about the advantages of this approach, see ["Session Configuration and the sessions.xml File"](#) on page 75-4.

TopLink Workbench displays sessions configurations and their contents in the Navigator window. When you select a session configuration, its attributes are displayed in the Editor window.

Figure 76–1 calls out the following user interface elements:

1. Sessions Configuration
2. Database Session
3. Relational Server Session
4. Connection Pool
5. EIS Server Session
6. XML Session
7. Session Broker

Figure 76–1 Sessions Configurations in Navigator Window



Using TopLink Workbench

To create a TopLink sessions configuration (`sessions.xml` file), use this procedure:



1. Click **New** on the toolbar and select **Sessions Configuration**.

You can also create a new sessions configuration by selecting **File > New > Session Configuration** from the menu, or by clicking **Create New Sessions Configuration** in the standard toolbar.

2. The new sessions configuration element appears in the Navigator window; the Sessions Configuration property sheet appears in the Editor window.

Enter data in each field on the Sessions Configuration property sheet as "[Configuring a Sessions Configuration](#)" on page 76-2 describes.

Configuring a Sessions Configuration

Each TopLink sessions configuration (`sessions.xml` file) can contain multiple sessions and session brokers. In addition, you can specify a classpath for each sessions configuration that applies to all the sessions it contains.

Using TopLink Workbench

To configure a session configuration, use this procedure:

1. Select the session configuration in the **Navigator**. Its properties appear in the Editor.

Figure 76–2 Sessions Configuration Property Sheet

Use the following information to enter data in each field of the Sessions configuration property sheet:

Field	Description
Project Save Location	Click Change and select the directory in which to save the sessions configuration.
Classpath	Lists the JAR or ZIP files that contain the compiled Java classes on which this sessions configuration depends for features that require an external Java class (for example, session event listeners). ■ To add a JAR or ZIP file, click Add Entries or Browse add the file. ■ To remove a JAR or ZIP file, select the file and click Remove. ■ To change the order in which TopLink searches these JAR or ZIP files, select a file and click Up to move it forward or click Down to move it back in the list.
Sessions for <sessions configuration name>	Lists the available sessions defined in this sessions configuration: ■ To add a session, click Add Session. ■ To remove a session, select the session and click Remove. ■ To rename a session, select the session and click Rename. For more information on creating sessions using TopLink Workbench, seen the following: ■ "Creating a Server Session" on page 76-4 ■ "Creating Session Broker and Client Sessions" on page 76-6 ■ "Creating Database Sessions" on page 76-8

Creating a Server Session

Oracle recommends that you create server sessions using TopLink Workbench (see ["Using TopLink Workbench"](#) on page 76-4).

After you create a server session, you create a client session by acquiring it from the server session (see ["Acquiring a Client Session"](#) on page 78-6).

Using TopLink Workbench

Before you create a server session, you must first create a sessions configuration (see ["Creating a Sessions Configuration"](#) on page 76-1).

Creating a Session

To create a new TopLink server session, use this procedure:

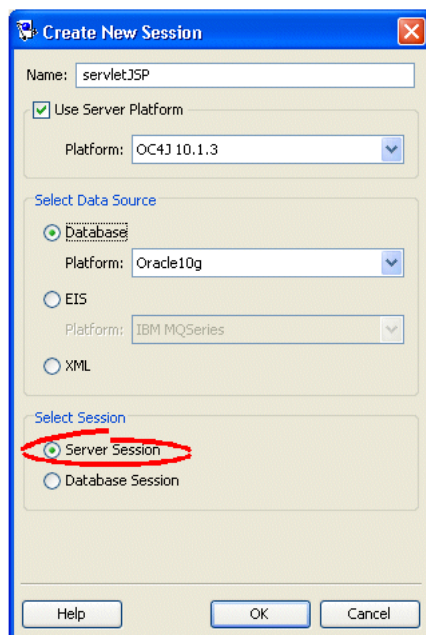
1. Select the sessions configuration in the **Navigator** window in which you want to create a session.



2. Click **Add Session** on the toolbar. The Create New Session dialog box appears.

You can also create a new server session by right-clicking the sessions configuration in the **Navigator** and selecting **New > Session** from the context menu, or by clicking **Add Session** on the Sessions Configuration property sheet.

Figure 76-3 Create New Session Dialog Box, Server Session Option



3. Enter data in each field on the Create New Session dialog box.

Use the following information to enter data in each field of the dialog box:

Field	Description
Name	Specify the name of the new session.

Field	Description
Use Server Platform	Check this field if you intend to deploy your application to a J2EE application server. If you check this field, you must configure the target application server by selecting a Platform.
Platform	This option is only available if you check Use Server Platform. Select the J2EE application server to which you will deploy your application. TopLink supports the following J2EE application servers: ■ OC4J 10.1.3 ■ OC4J 10.1.2 ■ OC4J 9.0.4 ■ OC4J 9.0.3 ■ WebLogic 8.1 ■ WebLogic 7.0 ■ WebLogic 6.1 ■ WebSphere 6.0 ■ WebSphere 5.1 ■ WebSphere 5.0 ■ WebSphere 4.0 ■ Custom The server platform you select is the default server platform for all sessions you create in this sessions configuration. At the session level, you can override this selection or specify a custom server platform class (see "Configuring the Server Platform" on page 77-14).
Select Data Source	Select the data source for this session configuration. Each session configuration must contain one data source. Choose one of the following: ■ Database to create a session for a relational project. ■ EIS to create a session for an EIS project. ■ XML to create a session for an XML project¹. See "TopLink Project Types" on page 20-1 for more information.
Select Session	Select Server Session to create a session for a single data source (including shared object cache and connection pools) for multiple clients in a three-tier application.

¹ You cannot create a server session for an XML project.

Using Java

You can create a server session in Java code using a project. You can create a project in Java code, or read a project from a `project.xml` file.

[Example 76-1](#) illustrates creating an instance (called `serverSession`) of a `Server` class using a `Project` class.

Example 76-1 Creating a Server Session from a Project Class

```
Project myProject = new Project();
Server serverSession = myProject.createServerSession();
```

[Example 76-2](#) illustrates creating an instance (called `serverSession`) of a `Server` class using a `Project` read in from a `project.xml` file.

Example 76-2 Creating a Server Session from a project.xml File Project

```
Project myProject = XMLProjectReader.read("myproject.xml");
Server serverSession = myProject.createServerSession();
```

[Example 76-3](#) illustrates creating a server session with a specified write connection pool minimum and maximum size (when using TopLink internal connection pooling). The default write connection pool minimum size is 5 and maximum size is 10.

Example 76-3 Creating a Server Session with Custom Write Connection Pool Size

```
XMLProjectReader.read("myproject.xml");
Server serverSession = myProject.createServerSession(32, 32);
```

Creating Session Broker and Client Sessions

A session broker may contain both server sessions and database sessions. Oracle recommends that you use the session broker with server sessions because server sessions are the most scalable session type.

Oracle recommends that you create server sessions using TopLink Workbench (see ["Using TopLink Workbench"](#) on page 76-6).

After you create and configure a session broker with server sessions, you can acquire a client session from the session broker at run time to provide a dedicated connection to all the data sources managed by the session broker for each client. For more information, see ["Acquiring a Client Session"](#) on page 78-6.

Using TopLink Workbench

Before you create a session broker session, you must first create a sessions configuration (see ["Creating a Sessions Configuration"](#) on page 76-1) and one or more server sessions (["Creating a Server Session"](#) on page 76-4), or one or more database sessions (["Creating Database Sessions"](#) on page 76-8).

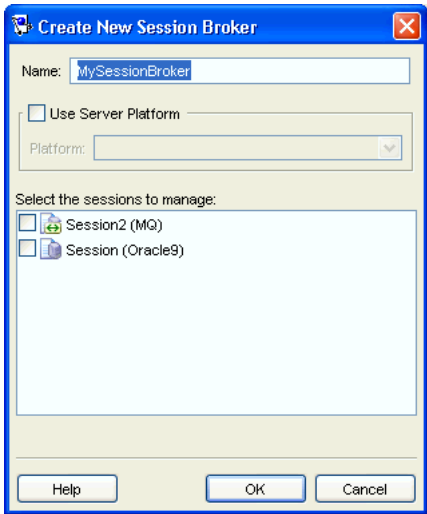
To create a new TopLink session broker, use this procedure:

1. Select the sessions configuration in the **Navigators** window in which you want to create a session broker.
2. Click **Add Session Broker** on the toolbar. The Create New Session Broker dialog box appears.

You can also create a new session broker by right-clicking the sessions configuration in the **Navigators** window and selecting **Add Session Broker** from the context menu or by clicking **Add Session Broker** on the Sessions Configuration property sheet.



Figure 76–4 Create New Session Broker Dialog Box



Use the following information to enter data in each field of the dialog box:

Field	Description
Name	Specify the name of the new session broker.
Use Server Platform	Check this field if you intend to deploy your application to a J2EE application server. If you check this field, you must configure the target application server by selecting a Platform .
Platform	This option is available only if you check Use Server Platform . Select the J2EE application server to which you will deploy your application. TopLink supports the following J2EE application servers: - OC4J 10.1.3 - OC4J 9.0.3¹ - WebLogic 8.1 - WebLogic 7.0 - WebLogic 6.1 - websphere 5.1 - websphere 5.0 - websphere 4.0 - Custom The server platform you select is the default server platform for all sessions you create in this sessions configuration. At the session level, you can override this selection or specify a custom server platform class (see "Configuring the Server Platform" on page 77-14).
Select the sessions to Manage	Select sessions to be managed by this new session broker (from the list of available sessions) and click OK . Note: This field appears only if the configuration contains valid sessions.

¹ Includes support for both 9.0.3 and 9.0.4.

Continue with [Chapter 77, "Configuring a Session"](#).

Using Java

Example 76–4 illustrates how you can create a session broker in Java code by instantiating a `SessionBroker` and registering the brokered sessions with it.

Because the session broker references other sessions, configure these sessions before instantiating the session broker. Add all required descriptors to the session, but do not initialize the descriptors or log the sessions. The session broker manages these issues when you instantiate it.

Example 76–4 Creating a Session Broker

```
Project databaseProject = new MyDatabaseProject();
Server databaseSession = databaseProject.createServerSession();

Project eisProject = new MyEISProject();
Server eisSession = eisProject.createServerSession();

SessionBroker sessionBroker = new SessionBroker();
sessionBroker.registerSession("myDatabase", databaseSession);
sessionBroker.registerSession("myEIS", eisSession);

sessionBroker.login();
```

Creating Database Sessions

Oracle recommends that you create database sessions using TopLink Workbench (see ["Using TopLink Workbench"](#) on page 76-8).

After you create a database session, you can acquire and use it at run time. For more information on acquiring a database session, see ["Acquiring a Session from the Session Manager"](#) on page 78-3.

Using TopLink Workbench

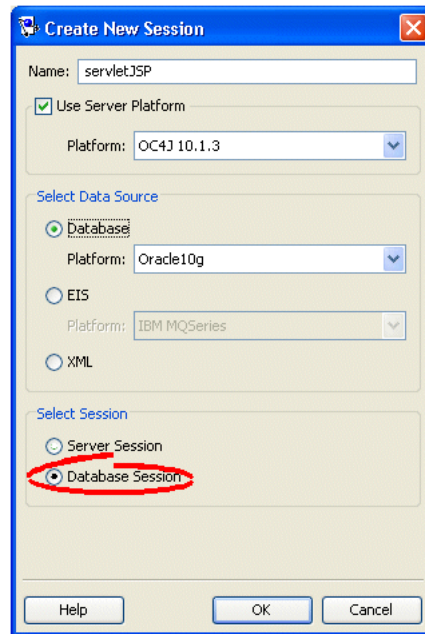
Before you create a database session, you must first create a sessions configuration (see ["Creating a Sessions Configuration"](#) on page 76-1).

To create a new TopLink database session, use this procedure:

1. Select the session configuration in the **Navigators** window in which you want to create a session.
2. Click **Add Session** on the toolbar. The Create New Session dialog box appears.



You can also create a new configuration by right-clicking the sessions configuration in the **Navigators** window and selecting **New > Session** from the context menu.

Figure 76–5 Create New Session Dialog Box, Database Session Option

3. Complete each field on the Create New Session dialog box.

Use the following information to enter data in each field of the dialog box:

Field	Description
Name	Specify the name of the new session.
Use Server Platform	Check this field if you intend to deploy your application to a J2EE application server. If you check this field, you must configure the target application server by selecting a Platform.
Platform	This option is available only if you check Use Server Platform. Select the J2EE application server to which you will deploy your application. TopLink supports the following J2EE application servers: ■ OC4J 10.1.3 ■ OC4J 9.0.3¹ ■ WebLogic 8.1 ■ WebLogic 7.0 ■ WebLogic 6.1 ■ websphere 5.1 ■ websphere 5.0 ■ websphere 4.0 ■ Custom The server platform you select is the default server platform for all sessions you create in this sessions configuration. At the session level, you can override this selection or specify a custom server platform class (see "Configuring the Server Platform" on page 77-14).

Field	Description
Select Data Source	Select the data source for this session configuration. Each session configuration must contain one data source. Choose one of the following: ■ Database to create a session for a relational project. ■ EIS to create a session for an EIS project. ■ XML to create a session for an XML project. See "TopLink Project Types" on page 20-1 for more information.
Select Session	Select Database Session to create a session for a single database (including shared object cache and connection pools) for a single client suitable for simple applications or prototyping.

¹ Includes support for both 9.0.3 and 9.0.4.

Enter the necessary information and click **OK**.

TopLink Workbench window appears, showing the database session in the Navigator window.

Continue with [Chapter 77, "Configuring a Session"](#).

Using Java

You can create an instance of the `DatabaseSession` class in Java code using a `Project`. You can create a project in Java code or read a project from a `project.xml` file.

[Example 76-5](#) illustrates creating a `DatabaseSession` using a `Project` class.

Example 76-5 Creating a Database Session from a Project Class

```
Project myProject = new Project();
DatabaseSession databaseSession = myProject.createDatabaseSession();
```

[Example 76-6](#) illustrates creating a `DatabaseSession` using a `Project` read in from a `project.xml` file.

Example 76-6 Creating a Database Session from a project.xml File Project

```
Project myProject = XMLProjectReader.read("myproject.xml");
DatabaseSession databaseSession = myProject.createDatabaseSession();
```

Creating Remote Sessions

Remote sessions are acquired through a remote connection to their server-side session. Remote sessions are acquired through Java code on the remote client. The server-side session must also be registered with an `oracle.toplink.remote.ejb.RemoteSessionController` and accessible from the RMI naming service.

You create remote sessions entirely in Java code (see "[Using Java](#)" on page 76-10).

Using Java

[Example 76-7](#) and [Example 76-8](#) demonstrate how to create a remote TopLink session on a client that communicates with a remote session controller on a server that uses

RMI. After creating the connection, the client application uses the remote session as it does with any other TopLink session.

Server

[Example 76-7](#) shows the code you add to your application's RMI service (`MyRMIServerManagerImpl`) to create and return an instance of an `RMIRemoteSessionController` to the client. The controller sits between the remote client and the local TopLink session.

The `RMIRemoteSessionController` you create on the server is based on a TopLink server session. You create and configure this server session as described in ["Creating a Server Session"](#) on page 76-4 and ["Configuring Server Sessions"](#) on page 79-1.

Example 76-7 Server Creating RMIRemoteSessionController for Client

```
RMIRemoteSessionController controller = null;
try {
    // Create instance of RMIRemoteSessionControllerDispatcher which implements
    // RMIRemoteSessionController. The constructor takes a TopLink session as a parameter
    controller = new RMIRemoteSessionControllerDispatcher (localTopLinkSession);
} catch (RemoteException exception) {
    System.out.println("Error in invocation " + exception.toString());
}
return controller;
```

Client

The client-side code gets a reference to the application's RMI service (in this example it is called `MyRMIServerManager`) and uses this code to get the `RMIRemoteSessionController` running on the server. The reference to the session controller is then used to create the `RMIConnection` from which it acquires a remote session.

Example 76-8 Client Acquiring RMIRemoteSessionController from Server

```
MyRMIServerManager serverManager = null;
// Set the client security manager
try {
    System.setSecurityManager(new MyRMISecurityManager());
} catch (Exception exception) {
    System.out.println("Security violation " + exception.toString());
}
// Get the remote factory object from the Registry
try {
    serverManager = (MyRMIServerManager) Naming.lookup("SERVER-MANAGER");
} catch (Exception exception) {
    System.out.println("Lookup failed " + exception.toString());
}
// Start RMIRemoteSession on the server and create an RMIConnection
RMIConnection rmiConnection = null;
try {
    rmiConnection = new RMIConnection(
        serverManager.createRemoteSessionController()
    );
} catch (RemoteException exception) {
    System.out.println("Error in invocation " + exception.toString());
}
// Create a remote session which we can use as a normal TopLink session
Session session = rmiConnection.createRemoteSession();
```


Configuring a Session

This chapter describes how to configure TopLink sessions.

[Table 77-1](#) lists the types of TopLink sessions that you can configure and provides a cross-reference to the type-specific chapter that lists the configurable options supported by that type.

Table 77-1 *Configuring TopLink Sessions*

If you are creating...	See...
Server and Client Sessions	Chapter 79, "Configuring Server Sessions"
Unit of Work Sessions	Chapter 100, "Understanding TopLink Transactions"
Isolated Client Sessions	Chapter 80, "Configuring Exclusive Isolated Client Sessions for Virtual Private Database"
Historical Client Sessions	Chapter 81, "Configuring Historical Client Sessions"
Session Broker and Client Sessions	Chapter 82, "Configuring Session Broker and Client Sessions"
Database Sessions	Chapter 83, "Configuring Database Sessions"

[Table 77-2](#) lists the configurable options shared by two or more TopLink sessions types.

For more information, see the following:

- ["Understanding TopLink Sessions"](#) on page 75-1
- ["Creating Sessions"](#) on page 76-1

Configuring Common Session Options

[Table 77-2](#) lists the configurable options shared by two or more TopLink session types. In addition to the configurable options described here, you must also configure the options described for the specific [Session Types](#), as shown in [Table 77-1](#)

Table 77-2 *Configurable Options for Session*

Option	Type	TopLink Workbench	Java
"Configuring a Primary Mapping Project" on page 77-2	Basic	✓	✓
"Configuring a Session Login" on page 77-4	Basic	✓	✓
"Configuring Logging" on page 77-4	Basic	✓	✓
"Configuring Multiple Mapping Projects" on page 77-9	Advanced	✓	✓

Table 77-2 (Cont.) Configurable Options for Session

Option	Type	TopLink Workbench	Java
"Configuring a Performance Profiler" on page 77-10	Advanced	✓	✓
"Configuring an Exception Handler" on page 77-12	Advanced	✓	✓
"Configuring Customizer Class" on page 77-13	Advanced	✓	✓
"Configuring the Server Platform" on page 77-14	Advanced	✓	✓
"Configuring Session Event Listeners" on page 77-17	Advanced	✓	✓
"Configuring a Coordinated Cache" on page 91-1	Advanced	✓	✓
"Configuring the Integrity Checker" on page 77-18	Advanced		✓
"Configuring Connection Policy" on page 77-19	Advanced	✓	✓
"Configuring Named Queries at the Session Level" on page 77-21	Basic		✓

Configuring a Primary Mapping Project

The mapping project contains your TopLink mapping metadata (see ["Understanding Projects" on page 20-1](#)), including descriptors and mappings. Each session is associated with at least one project so that the session can register the descriptors.

[Table 77-3](#) summarizes which sessions support a primary mapping project configuration.

Table 77-3 Session Support for Primary Mapping Project

Session	Using TopLink Workbench	Using Java
Server and Client Sessions	✓	✓
Session Broker and Client Sessions	✓	✓
Database Sessions	✓	✓

Using TopLink Workbench, you can export your mapping metadata as either a deployment XML file or as a Java class. Consequently, in a session, you can specify the mapping project as an XML file or as a Java class.

Oracle recommends that you export your mapping metadata from TopLink Workbench as a deployment XML file (see ["Exporting Project Information" on page 21-13](#)).

If you export your mapping metadata as a Java class, you must compile it and add it to the session configuration classpath (see ["Configuring a Sessions Configuration" on page 76-2](#)) before adding it to a session.

Note: When specifying the mapping project using XML, you can specify the Java resource path. In most applications, the `sessions.xml` and `project.xml` files are deployed inside the JAR file, and the project XML path is specified as a Java resource path.

When specifying the Java resource path, ensure that you are using the forward slash character (/) for directories, not the back slash (\). For example, `com/myapp/mypersistence/my-project.xml`, or `META-INF/my-project.xml`.

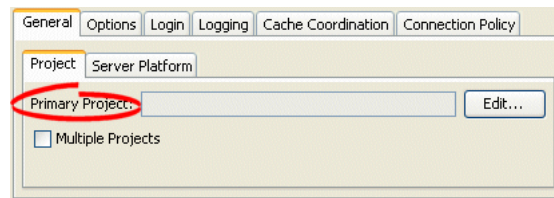
See ["Configuring Multiple Mapping Projects"](#) on page 77-9 for information on configuring additional TopLink projects for the session.

Using TopLink Workbench

To specify the primary TopLink project metadata for your session, use this procedure:

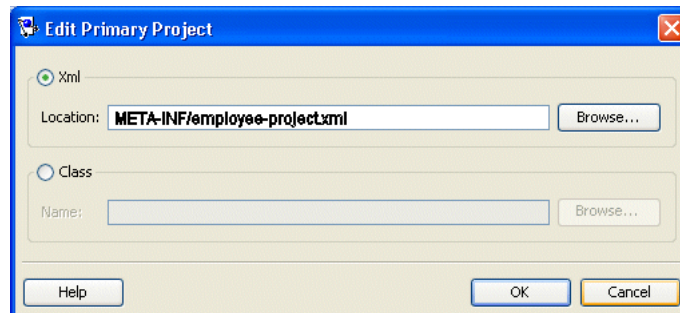
1. Select a server or database session in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.
3. Click the **Project** subtab. The Project subtab appears.

Figure 77-1 General Tab, Project Subtab, Primary Project Option



4. Select the following options:
 - Click **Edit** to define the primary project. The Edit PrimaryProject dialog box appears.
 - Select the **Multiple Projects** option to add additional projects to the session. See ["Configuring Multiple Mapping Projects"](#) for more information.

Figure 77-2 Edit Primary Project Dialog Box



5. Complete each field on the Edit Primary Project dialog box.
Use this information to enter data in each field on the dialog box:

Field	Description
XML	Select XML to add a mapping project as a deployment XML file. Click Browse to select the file.
Class	Select Class to add a mapping project as a <i>compiled</i> Java class file. Click Browse to select the file.

Using Java

Using Java, you can register descriptors with a session using the following API:

- **Project API**—Read your `project.xml` file (or instantiate your project class) and create your session using Project method `createServerSession` or `createDatabaseSession`.
- **Session API**—Add a descriptor or set of descriptors to a session using the DatabaseSession API that [Table 77-4](#) lists. Descriptors should be registered before login, but independent sets of descriptors can be added after login.

Table 77-4 DatabaseSession API for Registering Descriptors

Session Method	Description
<code>addDescriptors(Project)</code>	Add to the session all the descriptors owned by the passed in <code>Project</code> .
<code>addDescriptors(Vector)</code>	Add to the session all the descriptors in the passed in <code>Vector</code> .
<code>addDescriptor(Descriptor)</code>	Add an individual descriptor to the session.

Configuring a Session Login

A session login encapsulates details of data source access for any session that persists to a data source. The session login overrides any other login configuration.

[Table 77-5](#) summarizes which sessions support session login configuration.

Table 77-5 Session Support for Session Login

Session	Session Login
Server and Client Sessions	✓
Session Broker and Client Sessions	✓
Database Sessions	✓

The session login provides access to a variety of features, including the following:

- Connection configuration such as whether or not to use external connection pooling
- Sequencing configuration (that overrides sequencing configuration made at the project level, if any)
- Miscellaneous options specific to your chosen data source
- Properties (arbitrary, application-specific named values)

For more information, see the following:

- ["Data Source Login Types"](#) on page 84-2
- ["Configuring a Data Source Login"](#) on page 85-1

Configuring Logging

Use the TopLink logging framework to record TopLink behavior to a log file or session console.

[Table 77-6](#) summarizes which sessions support logging configuration.

Table 77-6 Session Support for Logging

Session	Using TopLink Workbench	Using Java
Server and Client Sessions	✓	✓
Unit of Work Sessions	✓	✓
Session Broker and Client Sessions	✓	✓
Database Sessions	✓	✓

Note: If the session belongs to a session broker, you must specify the logging information in the session broker—not in the session itself.

For a non-CMP application, you can configure logging using TopLink Workbench (see ["Using TopLink Workbench"](#) on page 77-5). For a CMP application, see ["Configuring Logging in a CMP Application"](#) on page 77-9.

For more information, see ["Logging"](#) on page 75-7.

Using TopLink Workbench

To specify the logging information for a session, use this procedure:

1. Select a database session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Logging** tab. The Logging tab appears.

Figure 77-3 Logging Tab

The screenshot shows the 'Logging' tab in the TopLink Workbench configuration window. The 'General' tab is selected, and the 'Logging' sub-tab is active. The 'No Logging' option is selected. The 'Logging Level' is set to 'fine'. The 'Console' output destination is selected. The 'Log Location' is set to 'standard output'. The 'Options' section includes checkboxes for 'Log Exception Stack Trace' (unchecked), 'Print Connection' (checked), 'Print Date' (checked), 'Print Session' (checked), and 'Print Thread' (unchecked).

Use the following information to enter data in each field of the Logging tab to select the profiler option to use with this session:

Option	Description
No Logging	Select this option to specify that nothing is logged for this session.

Option	Description
Server	Select this option to use logging capabilities of the application server to which you are deploying this application.
Java	Select this option to use <code>java.util.logging</code> package.
Standard	Select this option to use the TopLink logging framework. When selected, you can optionally configure the following options.
Logging Level	Define the amount of logging information to record (in ascending order of information): ▪ Config—Log only login, JDBC connection, and database information. ▪ Info (default)—Log the login/logout per sever session, with user name. After acquiring the session, detailed information is logged. ▪ Warning—Log exceptions that do not force TopLink to stop, including all exceptions not logged with Severe level. This does not include a stack trace. ▪ Severe—Log exceptions indicating TopLink cannot continue, and any exceptions generated during login. This includes a stack trace. ▪ Fine—Log SQL (including thread information). ▪ Finer—Similar to warning. Includes stack trace. ▪ Finest—Includes additional low-level information. ▪ All—Log everything.
Console	Select this option to display logging information to the standard console output.
File	Select this option to record logging information in a file. Click Browse to specify the name and location of the log file.
Options	Select this option to override additional logging option defaults for Java and Standard logging only.
Log Exception Stack Trace	Select this option to include the stack trace with any exception written to the log. Default: For SEVERE messages, log stack trace. For WARNING messages, only log stack trace at log level FINER or lower.
Print Connection	Select this option to include the connection identifier in any connection related log messages. Default: Enabled for all message and log levels.
Print Date	Select this option to include the date and time at which the log message was generated. Default: Enabled for all message and log levels.
Print Session	Select this option to include the session name in any session related log messages. Default: Enabled for all message and log levels.
Print Thread	Select this option to include the thread name in any thread related log messages. Default: Log only at log level FINER or lower.

Using Java

This section describes the following:

- [Using Session Logging API](#)
- [Configuring a Session to use java.util.logging Package](#)
- [Configuring Logging in a CMP Application](#)

Using Session Logging API

If you use TopLink native logging (the default), then at run time, you can configure logging options using `oracle.toplink.sessions.Session` logging API.

The `Session` interface defines the following logging methods:

- `setSessionLog`—specify the type of logging to use (any implementor of `oracle.toplink.logging.SessionLog`)
- `dontLogMessages`—disable logging
- `setLog`—specify the `java.io.Writer` to which the session logs messages
- `setLogLevel`—specify the level at which the session logs using `oracle.toplink.logging.SessionLog` constants:
 - OFF
 - SEVERE
 - WARNING
 - INFO
 - CONFIG
 - FINE
 - FINER
 - FINEST
 - ALL

[Example 77-1](#) illustrates how to configure a session to use `java.util.logging` package.

Example 77-1 Configuring a Session to Use java.util.logging

```
session.setSessionLog(new JavaLog());
```

[Example 77-2](#) illustrates how to configure a session to use the server log that OC4J provides. For more information about server logging, see ["Server Logging"](#) on page 75-9.

Example 77-2 Configuring a Session to Use Application Server Logging

```
session.setSessionLog(new OjdlLog());
```

[Example 77-3](#) illustrates how to configure a session to log to a `java.io.Writer`:

Example 77-3 Configuring a Session to Log to a java.io.Writer

```
session.setLog(myWriter);
```

Configuring a Session to use java.util.logging Package

If you use `java.util.logging` package, then you configure logging options in the `<JRE_HOME>/lib/logging.properties` file. Messages are written to zero or multiple destinations based on this configuration file.

If you configure a session to use `java.util.logging` package, consider the following:

- [logging.properties](#)
- [Formatters](#)
- [Namespace](#)

logging.properties Configure the `logging.properties` file as [Example 77-4](#) illustrates:

Example 77-4 java.util.logging Configuration in logging.properties

```
handlers = java.util.logging.ConsoleHandler
java.util.logging.ConsoleHandler.level = CONFIG
java.util.logging.ConsoleHandler.formatter = oracle.toplink.logging.TopLinkSimpleFormatter
oracle.toplink.LoggingSession.connection.level = CONFIG
```

For information about the types of formatters available, see ["Formatters"](#) on page 77-8.

Formatters TopLink provides two formatters: `TopLinkSimpleFormatter` and `TopLinkXMLFormatter`. They override the corresponding `java.util.logging` formatters and always log session and connection info when available. They also log thread and exception stack trace information at certain levels as specified by the logging level.

Namespace Namespace is supported for `java.util.logging`. [Table 77-7](#) lists the static constants defined in `oracle.toplink.logging.SessionLog` for TopLink components and the corresponding strings in `logging.properties`.

Table 77-7 Logging Property File Names

SessionLog	logging.properties
Not Applicable	<code>oracle.toplink</code>
Not Applicable	<code>oracle.toplink.<sessionname></code>
SQL	<code>oracle.toplink.<sessionname>.sql</code>
TRANSACTION	<code>oracle.toplink.<sessionname>.transaction</code>
EVENT	<code>oracle.toplink.<sessionname>.event</code>
CONNECTION	<code>oracle.toplink.<sessionname>.connection</code>
QUERY	<code>oracle.toplink.<sessionname>.query</code>
CACHE	<code>oracle.toplink.<sessionname>.cache</code>
PROPAGATION	<code>oracle.toplink.<sessionname>.propagation</code>
SEQUENCING	<code>oracle.toplink.<sessionname>.sequencing</code>
EJB	<code>oracle.toplink.<sessionname>.ejb</code>

In the `logging.properties` names listed in [Table 77-7](#), note that `<sessionname>` is the name of the session that the application is running in. For example, if the name

of the session is `MyApplication`, then you would use `oracle.toplink.MyApplication.sql` for the SQL logging property.

An application can also define its own namespace and write to it through the logging API, as long as the logger for that namespace is defined in the logging configuration. Otherwise messages are written to the parent logger, `oracle.toplink.<sessionname>`.

Configuring Logging in a CMP Application

For a CMP project, you do not configure a session directly. In this case, you specify the type of logging by configuring system property `toplink.log.destination` with one of the following values:

- fully qualified file specification (for example, `C:\logs\toplink.log`)—use TopLink native logging to write log messages to the specified file.
- `JAVA`—use `java.util.logging` package to write log messages to any destination you configure in the `<JRE_HOME>/lib/logging.properties` file.
- `SERVER`—use server logging to write log messages to the application server's log file (there is no separate TopLink log file in this case).
- `SYSOUT`—write log messages to `System.out`.

The log level for TopLink standard (default) logging can be set through the `toplink.log.level` system property.

To configure other logging options, use a `customization-class` (see ["Configuring pm-properties"](#) on page 8-10).

Configuring Multiple Mapping Projects

Each session is associated with at least one mapping project (see ["Configuring a Primary Mapping Project"](#) on page 77-2). You can include additional TopLink mapping projects for a session.

[Table 77-8](#) summarizes which sessions support additional mapping project configuration.

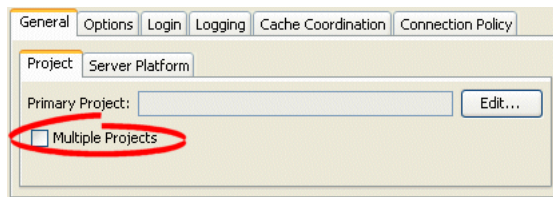
Table 77-8 Session Support for Additional Mapping Project

Session	Using TopLink Workbench	Using Java
Server and Client Sessions	✓	✓
Session Broker and Client Sessions	✓	✓
Database Sessions	✓	✓

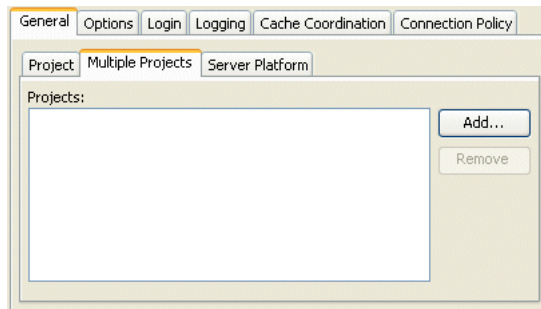
Using TopLink Workbench

To specify additional TopLink projects for your session, use this procedure:

1. Select a server or database session in the **Navigators**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.
3. Click the **Project** subtab. The Project subtab appears.

Figure 77-4 General Tab, Project Subtab, Multiple Projects Options

4. Select **Multiple Projects** option. The Multiple Projects subtab appears.
5. Click the **Multiple Projects** subtab.

Figure 77-5 General Tab, Multiple Projects Subtab

To add an additional mapping project to this session, click **Add**. For more information, see ["Configuring a Primary Mapping Project"](#) on page 77-2.

To remove TopLink mapping projects, select the project file and click **Remove**.

Using Java

Using Java, you can register descriptors from more than one project with a session using the `DatabaseSession` API that [Table 77-9](#) lists. You can register descriptors before login, but you can add independent sets of descriptors after login.

Table 77-9 DatabaseSession API for Registering Descriptors

Session Method	Description
<code>addDescriptors(Project)</code>	Add additional descriptor to the session in the form of a project.
<code>addDescriptors(Vector)</code>	Add a vector of individual descriptor files to the session in the form of a project.
<code>addDescriptor(Descriptor)</code>	Add individual descriptor to the session.

Configuring a Performance Profiler

To successfully improve the performance of a TopLink application, you must measure performance before and after each optimization. TopLink provides a variety of built-in performance measuring features (known as profilers) that you can configure at the session level.

[Table 77-10](#) summarizes which sessions support performance profiler configuration.

Table 77-10 Session Support for Performance Profiler Configuration

Session	Using TopLink Workbench	Using Java
Server and Client Sessions	✓	✓
Session Broker and Client Sessions	✓	✓
Database Sessions	✓	✓

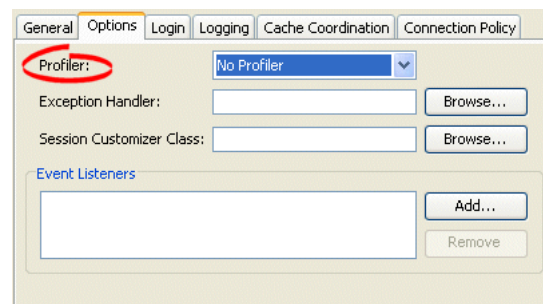
TopLink provides the following profilers:

- TopLink profiler: logs performance statistics for every executed query in a given session (see ["Measuring Performance With the TopLink Profiler"](#) on page 11-2)
- Oracle Dynamic Monitoring System (DMS): includes DMS instrumentation in essential objects to provide efficient Web browser based monitoring of run-time data in TopLink-enabled applications (see ["Measuring Performance With the Oracle Dynamic Monitoring System \(DMS\)"](#) on page 11-4)

Using TopLink Workbench

To specify the type of profiler in a session, use this procedure:

1. Select a session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Options** tab. The Options tab appears.

Figure 77-6 Options Tab, Profiler Options

Use the following information to select the profiler option to use with this session:

Option	Description
No Profiler	Disable all profiling.
DMS	Enable Oracle Dynamic Monitoring (DMS) profiling. For more information, see the following: ■ "Configuring the Oracle DMS Profiler" on page 11-6 ■ "Measuring Performance With the Oracle Dynamic Monitoring System (DMS)" on page 11-4.
Standard (TopLink)	Enable TopLink profiling. For more information, see the following: ■ "Configuring the TopLink Performance Profiler" on page 11-3 ■ "Measuring Performance With the TopLink Profiler" on page 11-2

Using Java

You can use Java to configure a session with a profiler using `Session` method `setProfiler`, as [Example 77-5](#) shows.

Example 77-5 Configuring a Session with a TopLink Profiler

```
session.setProfiler(new PerformanceProfiler());
```

To end a profiling session, use `Session` method `clearProfiler`.

Configuring an Exception Handler

You can associate a single exception handling class with each session. This class must implement the `oracle.toplink.exceptions.ExceptionHandler` interface.

[Table 77-11](#) summarizes which sessions support exception handler configuration.

Table 77-11 Session Support for Exception Handler Configuration

Session	Using TopLink Workbench	Using Java
Server and Client Sessions	✓	✓
Session Broker and Client Sessions	✓	✓
Database Sessions	✓	✓

For an example exception handler implementation, see ["Using Java"](#) on page 77-12.

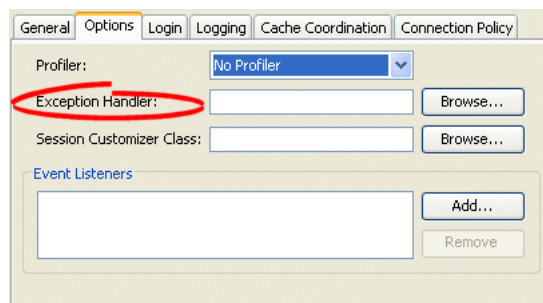
For more information, see ["Exception Handlers"](#) on page 75-12.

Using TopLink Workbench

To specify the exception handler class in a session, use this procedure:

1. Select a session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Options** tab. The Options tab appears.

Figure 77-7 Options Tab, Exception Handler Field



Click **Browse** and select the exception handler class for this session.

Using Java

[Example 77-6](#) shows an example exception handler implementation. In this implementation, the exception handler always tries to reestablish the connection if it has been reset by peer, but only retries a query if it is an instance of `ReadQuery`. Note

that this exception handler either returns the result of the reexecuted `ReadQuery` or throws an exception.

Example 77–6 Implementing an Exception Handler

```
session.setExceptionHandler(
    new ExceptionHandler()
    {
        public Object handleException(RuntimeException exception)
        {
            if (exception instanceof DatabaseException)
            {
                DatabaseException dbex = (DatabaseException) exception;
                if ((dbex.getInternalException() instanceof SQLException) &&
                    (((SQLException) dbex.getInternalException()).getErrorCode() == MyDriver.CONNECTION_RESET_BY_PEER))
                {
                    dbex.getAccessor().reestablishConnection(dbex.getSession());
                    if (dbex.getQuery() instanceof ReadQuery)
                    {
                        return dbex.getSession().executeQuery(dbex.getQuery());
                    }
                    throw exception;
                }
            }
            throw exception;
        }
    }
);
```

Note: Unhandled exceptions must be rethrown by the exception handler code.

Configuring Customizer Class

A session customizer class is a Java class that implements the `oracle.toplink.tools.sessionconfiguration.SessionCustomizer` interface and provides a default (zero-argument) constructor. You can use a session customizer to customize a session at run time on a loaded session before login occurs, similar to how you can use an amendment method to customize a descriptor (see ["Configuring Amendment Methods"](#) on page 28-78). For example, you can use a session customizer class to define and register session event listeners with the session event manager (see ["Configuring Session Event Listeners"](#) on page 77-17).

[Table 77–12](#) summarizes which sessions support customizer class configuration.

Table 77–12 Session Support for Customizer Class Configuration

Session	Using TopLink Workbench	Using Java
Server and Client Sessions	✓	✓
Session Broker and Client Sessions	✓	✓
Database Sessions	✓	✓

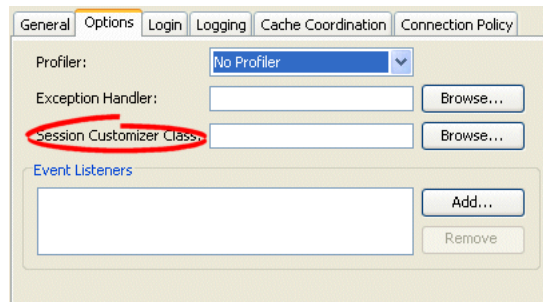
For more information, see ["Session Customization"](#) on page 75-4.

Using TopLink Workbench

To specify the session customizer class in a session, use this procedure:

1. Select a session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Options** tab. The Options tab appears.

Figure 77–8 Options Tab, Session Customizer Class Field



Click **Browse** and select the customizer class for this session.

Configuring the Server Platform

The TopLink server platform defines how a session integrates with a J2EE server including the following:

- **Run-time services:** Enables the deployment of a Java Management Extensions (JMX) MBean that allows monitoring of the TopLink session. Currently, this is only supported for OC4J.
- **External transaction controller:** Integrates the TopLink session with the server's Java Transaction API (JTA) service. This should always be used when using EJB or JTA transactions. You configure TopLink to integrate with the container's external transaction service by specifying a TopLink external transaction controller. For more information on external transaction services, see ["Unit of Work Transaction Demarcation"](#) on page 100-2.

Table 77–8 summarizes which sessions support a server platform.

Table 77–13 Session Support for Server Platform

Session	Using TopLink Workbench	Using Java
Server and Client Sessions	✓	✓
Session Broker and Client Sessions	✓	✓
Database Sessions	✓	✓

If the primary mapping project that you associate with a session has a persistence type of bean-managed persistence (BMP) or Java objects, you may configure a server platform using TopLink Workbench. For more information on primary mapping project, see ["Configuring a Primary Mapping Project"](#) on page 77-2.

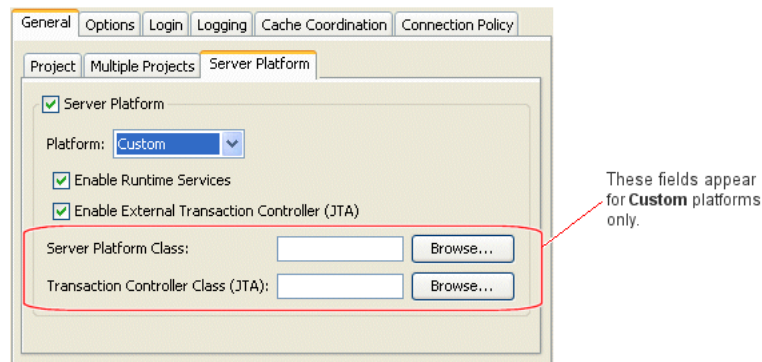
If the primary mapping project you associate with a session has a persistence type of container-managed persistence (CMP), by default, the TopLink runtime automatically configures a server platform to accommodate the application server on which it is deployed.

Using TopLink Workbench

To specify the server platform options for a session, use this procedure:

1. Select a session in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.
3. Click the **Server Platform** subtab. The Server Platform subtab appears.

Figure 77–9 General Tab, Server Platform Subtab



Use the following information to enter data in each field of the Server Platform subtab:

Field	Description
Server Platform	Check this field if you intend to deploy your application to a J2EE application server. If you check this field, you must configure the target application server by selecting a Platform .

Field	Description
Platform	Select the J2EE application server to which you will deploy your application. TopLink supports the following J2EE application servers: ■ OC4J 10.1.3 ■ OC4J 10.1.2 ■ OC4J 9.0.4 ■ OC4J 9.0.3 ■ WebLogic 8.1 ■ WebLogic 7.0 ■ WebLogic 6.1 ■ WebSphere 6.0 ■ WebSphere 5.1 ■ WebSphere 5.0 ■ WebSphere 4.0 ■ Custom For detailed information about supported application server versions and configuration requirements, see "Integrating TopLink With an Application Server" on page 7-1. Select Custom if you have created your own <code>oracle.toplink.platform.server.ServerPlatform</code> class to use an application server not currently supported by TopLink or to override an existing <code>ServerPlatform</code>. If you select Custom, you must specify your custom <code>ServerPlatform</code> class by selecting a Server Platform Class. The server platform you select overrides the default server platform set at the sessions configuration level (see "Creating a Sessions Configuration" on page 76-1).
Enable Runtime Services	Check this field to configure the TopLink runtime to enable the deployment of a JMX MBean that allows monitoring of the TopLink session. Currently, this is only supported for OC4J. To use this feature, you must enable DMS data collection. For more information, see "Configuring the Oracle DMS Profiler" on page 11-6.
Enable External Transaction Controller (JTA)	Check this field if you intend to integrate your application with an external transaction controller. For more information, see "Unit of Work Transaction Demarcation" on page 100-2. If you configure Platform for a J2EE application server that TopLink supports, the TopLink runtime will automatically select the appropriate external transaction controller class. If you configure Platform as Custom, you must specify an external transaction controller class by selecting an External Transaction Controller.
Server Platform Class	This option is only available if you configure Platform as Custom. Click Browse to select your custom <code>ServerPlatform</code> class.
Transaction Controller Class (JTA)	This option is only available if you configure Platform as Custom. If you checked Enable External Transaction Controller (JTA), click Browse to select the transaction controller class that corresponds with your custom <code>ServerPlatform</code> class.

Using Java

When using Java, you must pass the session in a server platform constructor.

[Example 77-7](#) illustrates using a session event listener (see ["Configuring Session Event Listeners"](#) on page 77-17) to configure a session with a server platform from the `oracle.toplink.platform.server` package.

Example 77-7 Configuring a Session with a Server Platform

```
public void preLogin(SessionEvent event) {
    Server server = (Server) event.getSession();
    server.setServerPlatform(new Oc4j_10_1_3_Platform((DatabaseSession) server));
}
```

Configuring Session Event Listeners

As you perform persistence operations with a session, the session produces various events (see ["Session Event Manager Events"](#) on page 75-6) that the TopLink runtime uses to coordinate its various components. You can configure a session with one or more session event listeners (see ["Session Event Listeners"](#) on page 75-7) to customize session behavior and debug session operations. For example, session event listeners play an important role in the configuration of isolated sessions (see ["Configuring Exclusive Isolated Client Sessions for Virtual Private Database"](#) on page 80-1).

[Table 77-14](#) summarizes which sessions support event listeners.

Table 77-14 Session Support for Event Listeners

Session	Using TopLink Workbench	Using Java
Server and Client Sessions	✓	✓
Session Broker and Client Sessions	✓	✓
Database Sessions	✓	✓

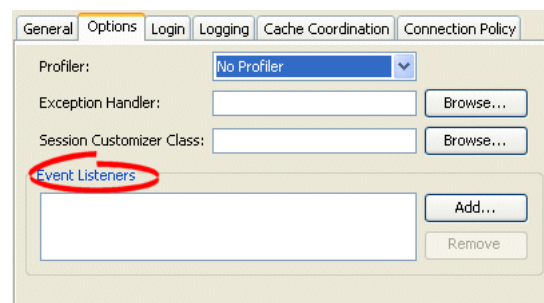
Using TopLink Workbench

Session Event Listeners

To specify the event listener class in a session, use this procedure:

1. Select a session in the **Navigators**. Its properties appear in the Editor.
2. Click the **Options** tab. The Options tab appears.

Figure 77-10 Options Tab, Event Listeners field



To add a new event listener, click **Add**, then select the event listener class for this session.

To remove an existing event listener, select the **Event Listener** and click **Remove**.

Using Java

[Example 77-8](#) illustrates how to use Java to register a session event listener with a session. TopLink provides a `SessionEventAdapter` to simplify creating a `SessionEventListener`. The `SessionEventAdapter` provides a default implementation of all the methods of the `SessionEventListener` interface. You need only override the specific methods of interest. Typically, you would define session event listeners in a session customizer class (see ["Configuring Customizer Class"](#) on page 77-13).

Example 77-8 Using the Session Event Adapter to Create a Session Event Listener

```
...
SessionEventAdapter myEventListener = new SessionEventAdapter() {
    // Listen for PostCommitUnitOfWork events
    public void postCommitUnitOfWork(SessionEvent event) {
        // Call the handler routine
        unitOfWorkCommitted();
    }
};
mySession.getEventManager().addListener(myEventListener);
...
```

For information on how to add logging to your listeners, see ["Logging"](#) on page 75-7.

Configuring the Integrity Checker

When you log into a session, TopLink initializes and validates the descriptors you registered with it. By configuring the integrity checker, you can customize this validation process to do the following:

- [Check Database](#)
- [Catch All Exceptions](#)
- [Catch Instantiation Policy Exceptions](#)

[Table 77-15](#) summarizes which sessions support descriptor integrity checking configuration.

Table 77-15 Session Support for Checking Descriptor Integrity

Session	Using TopLink Workbench	Using Java
Server and Client Sessions		✓
Session Broker and Client Sessions		✓
Database Sessions		✓

Check Database

The `IntegrityChecker` method `setShouldCheckDatabase` specifies whether or not the integrity checker should verify the descriptor's metadata against the database metadata. This will report any errors due to missing or incorrect table or fields

specified in the descriptors. This is turned off by default as it adds a significant overhead to connecting a session.

Catch All Exceptions

By default, the integrity checker catches all exceptions that occur during initialization, and throws a single exception at the end of initialization reporting all of the errors detected. If you only want the first exception encountered, you can disable this feature using `IntegrityChecker` method `setShouldCatchExceptions(false)`.

Catch Instantiation Policy Exceptions

By default, the integrity checker tests the default or configured constructor for each descriptor initialized in the session. To disable this feature, use `IntegrityChecker` method `setShouldCheckInstantiationPolicy(false)`.

Using Java

As [Example 77–9](#) shows, you can configure the integrity checker validation process.

Example 77–9 Configuring the Integrity Checker

```
session.getIntegrityChecker().setShouldCheckDatabase(true);
session.getIntegrityChecker().setShouldCatchExceptions(false);
session.getIntegrityChecker().setShouldCheckInstantiationPolicy(false);
session.login();
```

Configuring Connection Policy

Using a connection policy, you can control how a TopLink session acquires and uses read and write connections, including the following:

- [Exclusive Write Connections](#)
- [Lazy Connection Acquisition](#)

[Table 77–15](#) summarizes which sessions support connection policy configuration.

Table 77–16 Session Support for Connection Policy

Session	Using TopLink Workbench	Using Java
Server and Client Sessions	✓	✓
Session Broker and Client Sessions		
Database Sessions		

Exclusive Write Connections

An exclusive connection is one that TopLink allocates to a client session for reading (of isolated data) and writing for the duration of the client session's life cycle.

By default, exclusive connections are not used and a client session uses the server session's read connection pool for all non-pessimistic read queries. A connection is obtained from the read connection pool for each read query execution and released back to the pool after the query is executed. A connection is only obtained from the write connection pool for the unit of work commit operation, or, potentially, earlier if data modify queries, or read queries using pessimistic locking are used. The connection will be release back to the write connection pool after the unit of work is committed or released.

Exclusive connections are provided for use with database read security or Virtual Private Database (VPD) support. When using an exclusive connection, you will obtain it from the server session's write connection pool. When you acquire the client, the exclusive connection will be used for read queries to isolated classes (see ["Isolated Client Sessions"](#) on page 75-19), exclusive read queries, pessimistic read queries, and for the unit of work commit operation. The exclusive connection will only be released when the client session is released. TopLink still acquires a shared connection from the read connection pool for reading nonisolated data. If you use a JTA-managed external connection pool with exclusive connections, do not reuse a client session across JTA transaction boundaries, as the physical JTA database connection is released and acquired from the connection pool relative to the JTA transaction life cycle. A new client session, or the active unit of work, should be used for each JTA transaction. For more information, see ["Configuring Exclusive Read Connections"](#) on page 89-6.

You can also configure exclusive connections on a client-session-by-client-session basis (see ["Acquiring a Client Session That Uses Exclusive Connections"](#) on page 78-7) and for named queries (see ["Configuring Named Query Advanced Options"](#) on page 28-24).

Note: If any client session contains an exclusive connection, you must release the session (see ["Logging Out of a Session"](#) on page 78-10) when you are finished using it. Relying on the finalizer to release the connection when the session is garbage collected may cause errors when dealing with JTA transactions.

Lazy Connection Acquisition

By default, TopLink acquires write connections lazily, when you perform the first unit of work commit operation, exclusive read query, or pessimistic read query with your client session. The write connection will also be released after each unit of work it committed or released.

Alternatively, you can configure TopLink to acquire the write connection at the time you acquire a client session, and release the connection when you release the client session.

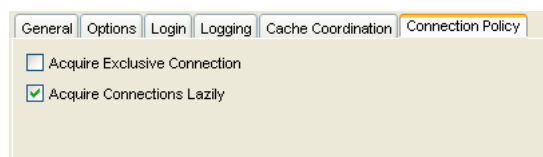
You can also configure lazy connection acquisition on a client-session-by-client-session basis (see ["Acquiring a Client Session that Does Not Use Lazy Connection Allocation"](#) on page 78-9).

Using TopLink Workbench

To specify the connection policy in a session, use this procedure:

1. Select a session in the **Navigators**. Its properties appear in the Editor.
2. Click the **Connection Policy** tab. The Connection Policy tab appears.

Figure 77-11 Connection Policy Tab



Using Java

To configure whether or not an exclusive connection is allocated to a particular isolated session, use `ConnectionPolicy` method `setShouldUseExclusiveConnection`.

To define a map of properties used to support an isolated session, use the following `ConnectionPolicy` methods:

- `setProperty(Object key, Object value)`: Adds the property `value` to the Map under `key`, overwriting the existing value if `key` already exists in the Map.
- `Object getProperty(Object key)`: Returns the value associated with `key` as an `Object`.
- `boolean hasProperties`: Returns `true` if one or more properties exist in the Map; otherwise returns `false`.

The TopLink runtime passes this Map into `SessionEvent` events `PostAcquireExclusiveConnection` and `PreReleaseExclusiveConnection` so that your implementation can make the appropriate PL/SQL calls to the underlying database platform (see ["PostAcquireExclusiveConnection Event Handler"](#) on page 80-1 and ["PreReleaseExclusiveConnection Event Handler"](#) on page 80-2).

To configure the session to use a named connection pool, use the `ConnectionPool` constructor that takes a `String` connection pool name as an argument:

```
Session clientSession = server.acquireClientSession(
    new ConnectionPolicy("myConnectionPool")
);
```

Configuring Named Queries at the Session Level

A **named query** is a TopLink query that you create and store, by name, in a session for later retrieval and execution. Named queries improve application performance, because they are prepared once and they (and all their associated supporting objects) can be efficiently reused thereafter making them well-suited for frequently executed operations.

If a named query is global to a project, configure it at the session level. Alternatively, you can configure a named query at the descriptor level (see ["Configuring Named Queries at the Descriptor Level"](#) on page 28-10).

Use named queries to specify SQL, EJB QL, or TopLink Expression queries to access your data source.

[Table 77-17](#) summarizes which sessions support named query configuration.

Table 77-17 Session Support for Named Queries

Session	Using TopLink Workbench	Using Java
Server and Client Sessions		✓
Session Broker and Client Sessions		✓
Database Sessions		✓

After you create a named query, you can execute it by name on the TopLink session (see ["Using Named Queries"](#) on page 98-17).

For more information about named queries, see ["Named Queries"](#) on page 96-15.

Using Java

You can store a query by name in a `Session` using `Session` method `addQuery(String name, DatabaseQuery query)`.

Acquiring and Using Sessions at Run Time

After you create and configure sessions, you can use the session manager to acquire a session instance at run time.

This section explains the following:

- [Session Acquisition Overview](#)
- [Acquiring the Session Manager](#)
- [Acquiring a Session from the Session Manager](#)
- [Acquiring a Client Session](#)
- [Logging In to a Session](#)
- [Using Session API](#)
- [Logging Out of a Session](#)
- [Storing Sessions in the Session Manager Instance](#)
- [Destroying Sessions in the Session Manager Instance](#)

Session Acquisition Overview

Oracle recommends that you export session instances from TopLink Workbench to one or more uniquely named `sessions.xml` files and then use the session manager to load sessions from these `sessions.xml` files.

The TopLink session manager enables developers to build a series of sessions that are maintained under a single entity. The session manager is a static utility class that loads TopLink sessions from the `sessions.xml` file, caches the sessions by name in memory, and provides a single access point for TopLink sessions.

The session manager supports the following session types:

- Server session
- DatabaseSession
- SessionBroker

See [Chapter 75, "Understanding TopLink Sessions"](#) for detailed information on these available sessions.

The session manager has two main functions: it creates instances of the sessions; it ensures that only a single instance of each named session exists for any instance of a session manager.

This is particularly useful for EJB applications in that EJB can acquire the session manager and acquire the desired session from it.

Understanding the Session Manager

When a client application requires a session, it requests the session from the TopLink session manager. The two main functions of the session manager are to instantiate TopLink sessions for the server, and to hold the sessions for the life of the application. The session manager instantiates database sessions, server sessions, or session brokers based on the configuration information in the `sessions.xml` file.

The session manager instantiates sessions as follows:

- The client application requests a session by name.
- The session manager looks up the session name in the `sessions.xml` file. If the session name exists, the session manager instantiates the specified session; otherwise, it throws an exception.
- After instantiation, the session remains viable until the application is shut down.

Multiple Sessions

Oracle does not recommend instantiating a session and passing it around as a global entity.

Oracle recommends that you acquire sessions from the session manager and perform all persistence operations using the unit of work acquired from the session.

Note that in the case of a server session or a session broker that contains server sessions, after you acquire the session you will acquire a client session from it. From a given server session (or session broker that contains server sessions), you can acquire as many client sessions as you have clients.

Each client can easily manage concurrent access and referential constraints by acquiring a unit of work from its client session and performing all persistence operations using the unit of work.

Acquiring the Session Manager

TopLink maintains only one instance of the session manager class. The singleton session manager maintains all the named TopLink sessions at run time. When an application requests a session by name, the session manager retrieves the specified session from the appropriate configuration file.

As [Example 78–1](#) illustrates, to access the session manager instance, use the `oracle.toplink.tools.sessionmanagement.SessionManager` method `getManager`. You can then use the session manager instance to load TopLink sessions.

Example 78–1 Acquiring a Session Manager Instance

```
import oracle.toplink.tools.sessionmanagement.SessionManager;  
SessionManager sessionManager = SessionManager.getManager();
```


Acquiring a Session from the Session Manager

When the session manager loads a session that is not yet in its cache, the session manager creates an instance of the appropriate session type and configures it according to the `sessions.xml` file configuration.

Note: To best use the methods associated with the session type that is being instantiated, cast the session that is returned from the `getSession` method. This type must match the session type that is defined in the `sessions.xml` file for the named session.

This section explains the following:

- [Loading a Session from sessions.xml Using Defaults](#)
- [Loading a Session from sessions.xml with an Alternative Class Loader](#)
- [Loading a Session from an Alternative Session Configuration File](#)
- [Loading a Session Without Logging In](#)
- [Reloading and Refreshing Session Configuration](#)
- [Refreshing a Session when the Class Loader Changes](#)

Loading a Session from sessions.xml Using Defaults

If you have a single sessions configuration file (`sessions.xml`) that contains all the session instances created from by TopLink Workbench, then you can load a session by name, as [Example 78-2](#) illustrates.

Example 78-2 Acquiring a Named Session from Session Manager Using Defaults

```
/* Load a named session (mysession) defined in the sessions.xml file. */
SessionManager manager = SessionManager.getManager();
Session session = manager.getSession("mysession");
```

In this example, the following session manager default configuration applies:

- **Class loader**—The thread-based class loader is used to find and load the `sessions.xml` resource and resolve any classes referenced in the `sessions.xml` and `project.xml` files.

If you acquire the session in an application class, this will typically be the application's class loader, which is correct. In a J2EE application, it is best to specify this as the class loader from a class in the same JAR file that the `sessions.xml` file is deployed in.
- **File**—By default, the file named `sessions.xml` in the root directory relative to the class loader is used.

If the file is named differently, or not in the root directory, the relative path must be specified. Relative resource paths in Java must use `/`, not `\`.
- **Session name**—The name passed into the `getSession` call.

This name must be unique for the entire application server, not just unique within the application.
- **Login**—`true`. The session will be connected by default.

If you must manually configure the session before login, set this option to `false` (see ["Loading a Session Without Logging In"](#) on page 78-5).

- `Refresh=false`. If already loaded, the same session will be returned.

Refresh should only be used, if it is known that the existing session is not being used, and the configuration has changed, such as in a J2EE environment redeployment scenario.

- `Verify class loader=false`. The session manager will not refresh the session if the class loader changes.

This should normally be set to `true`. It must be set to `true` in a J2EE environment, if hot deployment or redeployment to a running application server is required (see ["Refreshing a Session when the Class Loader Changes"](#) on page 78-6).

Loading a Session from `sessions.xml` with an Alternative Class Loader

You can use an alternative class loader to load sessions. This is common when your TopLink application integrates with a J2EE container. The session manager uses the class loader to find and load the `sessions.xml` resource and resolve any classes referenced in the `sessions.xml` and `project.xml` files.

In most cases, you use the class loader from the current thread context, as [Example 78-3](#) illustrates. In this example, the session named `mysession` is loaded from the first file in the application classpath named `sessions.xml` using the class loader associated with the current thread context.

Example 78-3 Loading a Session Using the Current Thread Context Class Loader

```
/* Use the specified ClassLoader to load a session (mysession) defined in the
sessions.xml file */
SessionManager manager = SessionManager.getManager();
Session session = manager.getSession(
    "mysession", // session name to load
    Thread.currentThread().getContextClassLoader() // ClassLoader instance to use
);
```

However, if your J2EE container does not support using the current thread context class loader, you can use the class loader from the current class, as [Example 78-4](#) illustrates.

Example 78-4 Loading a Session Using the Current Class's Class Loader

```
/* Use the specified ClassLoader to load a session (mysession) defined in the
sessions.xml file */
SessionManager manager = SessionManager.getManager();
Session session = manager.getSession(
    "mysession", // session name to load
    this.getClass().getClassLoader() // ClassLoader instance to use
);
```

Note: Oracle Containers for J2EE supports the use of the class loader from the current thread.

Loading a Session from an Alternative Session Configuration File

If your session instances are contained in multiple, uniquely named session configuration files (`sessions.xml` files), then you must explicitly create an

`XMLSessionConfigLoader` object initialized with the name of the `sessions.xml` file and pass that `XMLSessionConfigLoader` into the `SessionManager` method `getSession`, as [Example 78-5](#) illustrates.

The file path you specify is relative to the class loader root directory. Relative resource paths in Java must use the forward slash (/), not back slash (\).

In this example, the session named `mysession` is loaded by the specified class loader from the first file in the application classpath named `toplink-sessions.xml`.

Example 78-5 Loading a Session from an Alternative Configuration File

```
/* XMLSessionConfigLoader loads the toplinek-sessions.xml file */
SessionManager manager = SessionManager.getManager();
manager.getSession(
    new XMLSessionConfigLoader("toplink-sessions.xml"),
    "mysession",
    this.class.getClassLoader()
);
```

Loading a Session Without Logging In

The `XMLSessionConfigLoader` (see ["Loading a Session from an Alternative Session Configuration File"](#) on page 78-4) lets you call a session using the `SessionManager` method `getSession`, without invoking the `Session` method `login`, as [Example 78-6](#) shows. This lets you prepare a session for use and leave login to the application.

Example 78-6 Open Session with No Login

```
SessionManager manager = SessionManager.getManager();
Session session = manager.getSession(
    new XMLSessionConfigLoader(), // XMLSessionConfigLoader (sessions.xml file)
    "mysession", // session name
    YourApplicationClass.getClassLoader(), // class loader
    false, // do not log in session
    false); // do not refresh session
```

Reloading and Refreshing Session Configuration

You can tell the session manager to refresh an existing session from the `sessions.xml` file. Typically, this would only ever be used in a J2EE environment at redeployment time, or after a reset of a running server. You should only use this option when you know that the existing session is not being used.

Example 78-7 Forcing a Reparse of the sessions.xml File

```
//In this example, XMLSessionConfigLoader loads sessions.xml from the classpath
SessionManager manager = SessionManager.getManager();
Session session = manager.getSession(
    new XMLSessionConfigLoader(), // XMLSessionConfigLoader (sessions.xml file)
    "mysession", // session name
    YourApplicationClass.getClassLoader(), // class loader
    true, // log in session
    true // refresh session
);
```

Refreshing a Session when the Class Loader Changes

In a non-CMP J2EE environment, if you require hot deployment or redeployment to a running application server, you must tell the session manager to refresh your session if the class loader changes, as [Example 78–8](#) shows. This option makes the session manager refresh the session if the class loader changes, which occurs when the application is redeployed. When this option is set to `true`, the same class loader must always be used to retrieve the session.

Example 78–8 Forcing a Reparse of the sessions.xml File

```
//In this example, XMLSessionConfigLoader loads sessions.xml from the classpath
SessionManager manager = SessionManager.getManager();
Session session = manager.getSession(
    new XMLSessionConfigLoader(), // XMLSessionConfigLoader (sessions.xml file)
    "mysession", // session name
    YourApplicationClass.getClassLoader(), // class loader
    true, // log in session
    false, // do not refresh session when loaded
    true // do refresh session if class loader changes
);
```

In a CMP J2EE environment, the TopLink runtime and CMP integration handles this for you automatically.

Acquiring a Client Session

Before you can acquire a client session, you must first use the session manager to acquire a server session or a session broker that contains server sessions (see ["Acquiring a Session from the Session Manager"](#) on page 78-3).

[Table 78–1](#) summarizes the methods used to acquire various types of client sessions from a server session and a session broker session that contains server sessions.

Table 78–1 Method Used to Acquire a Client Session

Client Session	Server Session Method	Session Broker Session Method
Regular or Isolated	<code>acquireClientSession()</code>	<code>acquireClientSessionBroker()</code>
Regular or Isolated	<code>acquireClientSession(ConnectionPolicy)</code>	<i>not applicable</i>
Historical	<code>acquireHistoricalSession(AsOfClause)</code>	<i>not applicable</i>

The `acquireClientSession` method returns a session of type `ClientSession`.

The `acquireClientSessionBroker` method returns a session of type `SessionBroker`.

In both cases, you should cast the returned object to type `Session` and use it as you would any other session.

For more information, see the following:

- ["Acquiring an Isolated Client Session"](#) on page 78-7
- ["Acquiring a Historical Client Session"](#) on page 78-7
- ["Acquiring a Client Session That Uses Exclusive Connections"](#) on page 78-7
- ["Acquiring a Client Session that Uses Connection Properties"](#) on page 78-8

- ["Acquiring a Client Session that Uses a Named Connection Pool"](#) on page 78-8
- ["Acquiring a Client Session that Does Not Use Lazy Connection Allocation"](#) on page 78-9

Acquiring an Isolated Client Session

If in your TopLink project you configure all classes as isolated (see ["Configuring Cache Isolation at the Project Level"](#) on page 22-16), or one or more classes as isolated (see ["Configuring Cache Isolation at the Descriptor Level"](#) on page 28-37), then all client sessions that you acquire from a parent server session will be isolated client sessions (see ["Isolated Client Sessions"](#) on page 75-19).

Using a `ConnectionPolicy`, you can acquire an isolated client session that uses exclusive connections (see ["Acquiring a Client Session That Uses Exclusive Connections"](#) on page 78-7). This isolated client session is configured with connection properties for use with the Oracle Virtual Private Database (VPD) feature, or both (see ["Acquiring a Client Session that Uses Connection Properties"](#) on page 78-8).

For more information about VPD, see ["Isolated Client Sessions and Oracle Virtual Private Database \(VPD\)"](#) on page 75-20.

Acquiring a Historical Client Session

After you configure TopLink to access historical data (see ["Historical Client Session Configuration Overview"](#) on page 81-1), you can query historical data using any session type.

When you query historical data using a regular client session or database session, you must always set `ObjectLevelReadQuery` method `maintainCache` to `false` in order to prevent old (historical) data from corrupting the session cache. However, you can query both current and historical object versions.

As a convenience, TopLink provides a historical session to simplify this process. When you query historical data using a historical session, you do not need to set `ObjectLevelReadQuery` method `maintainCache` to `false`. However, you can query objects only as of the specified time.

Before you can acquire a historical client session, you must first use the session manager to acquire a server session.

To acquire a historical client session, use `Server` method `acquireHistoricalSession` passing in an `AsOfClause`.

The `AsOfClause` specifies a point in time that applies to all queries and expressions subsequently executed on the historical session. The historical session's cache is a read-only snapshot of object versions as of the specified time. Its cache is isolated from its parent server session's shared object cache.

Acquiring a Client Session That Uses Exclusive Connections

[Example 78-9](#) illustrates how to configure a `ConnectionPolicy` and use it to acquire a client session that uses exclusive connections.

Example 78-9 Acquiring a Client Session that Uses Connection Properties

```
ConnectionPolicy connectionPolicy = new ConnectionPolicy();
// Use an exclusive connection for the session
connectionPolicy.setShouldUseExclusiveConnection(true);

Session clientSession = server.acquireClientSession(connectionPolicy);
```

```
// By default, an exclusive connection will be acquired lazily
```

An exclusive connection is not allocated from a shared connection pool. The connection is dedicated to the client session that acquires it.

Note: If you configure a client session to use an exclusive connection, you must release the session when you are finished using it. Relying on the finalizer to release the connection when the session is garbage collected may cause errors when dealing with JTA transactions.

A named query can also use an exclusive connection (see ["Configuring Named Query Advanced Options"](#) on page 28-24).

For more information, see the following:

- ["Acquiring a Client Session that Does Not Use Lazy Connection Allocation"](#) on page 78-9
- ["Configuring Connection Policy"](#) on page 77-19.

Acquiring a Client Session that Uses Connection Properties

[Example 78-10](#) illustrates how to configure a `ConnectionPolicy` and use it to acquire a client session that uses connection properties. In this example, the properties are used by the Oracle VPD feature (see ["Isolated Client Sessions and Oracle Virtual Private Database \(VPD\)"](#) on page 75-20). You can use connection properties for other application purposes.

Example 78-10 Acquiring an Isolated Session Using Connection Properties

```
ConnectionPolicy connectionPolicy = new ConnectionPolicy();  
// Set VPD specific properties to be used in the events  
connectionPolicy.setProperty("userLevel", new Integer(5));  
  
Session clientSession = server.acquireClientSession(connectionPolicy);
```

For more information, see ["Configuring Connection Policy"](#) on page 77-19.

Acquiring a Client Session that Uses a Named Connection Pool

Before you can acquire a client session that uses a named connection pool, you must configure your session with a named connection pool. For more information on named connection pools, see ["Application-Specific Connection Pools"](#) on page 84-9. For more information on creating a named connection pool, see ["Internal Connection Pool Creation Overview"](#) on page 88-1.

To acquire a client session that uses a named connection pool, use `Server` method `acquireClientSession`, passing in a `ConnectionPolicy` configured with the desired connection pool. The acquired `ClientSession` uses connections from the specified pool for writes (reads still go through the `Server` read connection pool).

[Example 78-11](#) illustrates how to configure a `ConnectionPolicy` to specify a named connection pool named `myConnectionPool`.

Example 78-11 Acquiring a Client Session that Uses a Named Connection Pool

```
// Assuming you created a connection pool named "myConnectionPool"  
Session clientSession = server.acquireClientSession(  
    new ConnectionPolicy("myConnectionPool")
```

```
);
```

For more information, see ["Configuring Connection Policy"](#) on page 77-19.

Acquiring a Client Session that Does Not Use Lazy Connection Allocation

By default, the server session does not allocate a data source connection for a client session until a transaction starts (a lazy data source connection). Alternatively, you can acquire a client session that allocates a connection immediately.

[Example 78-12](#) illustrates how to configure a `ConnectionPolicy` to specify that lazy connection allocation is not used.

Example 78-12 Acquiring a Client Session that Does Not Use Lazy Connections

```
ConnectionPolicy connectionPolicy = new ConnectionPolicy();
connectionPolicy.setIsLazy(false);
Session clientSession = server.acquireClientSession(connectionPolicy);
```

For more information, see ["Configuring Connection Policy"](#) on page 77-19.

Logging In to a Session

Before you can use a session, you must first log in to the session using `Session` method `login`.

By default, when you load a session using the session manager, TopLink automatically logs in to the session using the zero-argument `login` method. For information on loading a session without automatically logging into the session, see ["Loading a Session Without Logging In"](#) on page 78-5.

If you load a session without logging in, you can choose from the following signatures of the `login` method:

- `login()`: Use the `Login`, user name, and password defined in the corresponding `sessions.xml` file.
- `login(Login login)`: Override the `Login` defined in the corresponding `sessions.xml` file with the specified `Login`.
- `login(String username, String password)`: Override the user name and password defined in the corresponding `sessions.xml` file with the specified user name and password.

When you log in to a session broker, the session broker logs in all contained sessions and initializes the descriptors in the sessions. After login, the session broker appears and functions as a regular session. TopLink handles the multiple database access transparently.

Using Session API

For more information on using session API, for caching, see ["Understanding the Cache"](#) on page 90-1.

For more information on using session API for queries, see ["Understanding TopLink Queries"](#) on page 96-1.

For more information on using session API for transactions, see ["Understanding TopLink Transactions"](#) on page 100-1.

Logging Out of a Session

When you are finished using a database or server session, you must log out of the session using `Session` method `logout`.

When logging out of a session broker session logs out of all sessions registered with the session broker.

When you are finished using a client session, you must release the session and any connections allocated to it using `Session` method `release`.

Although TopLink provides a finalizer to release sessions, this is a last resort. Oracle recommends that you always log out of your sessions.

Storing Sessions in the Session Manager Instance

Although Oracle recommends that you export all session instances from TopLink Workbench to one or more `sessions.xml` files, alternatively, you can manually create a session in your application and, as [Example 78-13](#) illustrates, manually store it in the session manager using `SessionManager` method `addSession`. Then, you can acquire a session by name using the `SessionManager` method `getSession`.

Note: The `addSession` method is not necessary if you are loading sessions from a session configuration file.

Example 78-13 Storing Sessions Manually in the Session Manager

```
// create and log in to the session programmatically
Session theSession = project.createDatabaseSession();
theSession.login();
// store the session in the SessionManager instance
SessionManager manager = SessionManager.getManager();
manager.addSession("mysession", theSession);
// retrieve the session
Session session = SessionManager.getManager().getSession("mysession");
```

Destroying Sessions in the Session Manager Instance

You can destroy sessions individually by name or destroy all sessions.

Note: You should only do this when a J2EE application is un-deployed, or when the entire application is shut down and only when it is known that the session is no longer in use. You should log out of a session before destroying it (see ["Logging Out of a Session"](#) on page 78-10). If you do not log out of a session, the session manager will at the time you use it to destroy a session.

To destroy one session instance by name, use `SessionManager` method `destroySession`, as [Example 78-14](#) illustrates. If the specified session is not in the session manager cache, a `ValidationException` is thrown.

Example 78-14 Destroying a Session in the Session Manager

```
SessionManager manager = SessionManager.getManager();
Server server = (Server) manager.getSession("myserversession");
```



```
...  
// Destroy session by name. If the session named myserversession is not in the  
// session manager cache, throw a ValidationException  
manager.destroySession("myserversession");
```

To destroy all session instances, use the `SessionManager` method `destroyAllSessions`, as [Example 78–15](#) illustrates.

Example 78–15 Destroying All Sessions in the Session Manager

```
SessionManager manager = SessionManager.getManager();  
Server server = (Server) manager.getSession("myserversession");  
SessionBroker broker = (SessionBroker) manager.getSession("mysessionbroker");  
...  
// Destroy all sessions stored in the session manager  
manager.destroyAllSessions();
```


Configuring Server Sessions

This chapter describes the various components that you must configure to use server and client sessions.

Server Session Configuration Overview

Table 79–1 lists the configurable options for server sessions.

Table 79–1 Configurable Options for Server Sessions

Option	Type	TopLink Workbench	Java
"Configuring Internal Connection Pools" on page 79-1	Basic	✓	✓
"Configuring a Primary Mapping Project" on page 77-2	Basic	✓	✓
"Configuring a Session Login" on page 77-4	Basic	✓	✓
"Configuring Logging" on page 77-4	Basic	✓	✓
"Configuring External Connection Pools" on page 79-2	Advanced	✓	✓
"Configuring Multiple Mapping Projects" on page 77-9	Advanced	✓	✓
"Configuring a Performance Profiler" on page 77-10	Advanced	✓	✓
"Configuring an Exception Handler" on page 77-12	Advanced	✓	✓
"Configuring Customizer Class" on page 77-13	Advanced	✓	✓
"Configuring the Server Platform" on page 77-14	Advanced	✓	✓
"Configuring Session Event Listeners" on page 77-17	Advanced	✓	✓
"Configuring a Coordinated Cache" on page 91-1	Advanced	✓	✓
"Configuring the Integrity Checker" on page 77-18	Advanced	✓	✓
"Configuring Named Queries at the Session Level" on page 77-21	Basic		✓

Configuring Internal Connection Pools

An internal connection pool is a collection of reusable connections to a single data source provided by any session that persists to a data source. By default, such a session provides both an internal read and write connection pool.

In this case, you can do the following:

- Configure read and write connection pool options such as minimum and maximum number of connections, alternate connection configuration, and properties (arbitrary, application-specific named values).

- Create named connection pools for whatever application-specific purpose you choose.
- Create sequence connection pools that TopLink uses exclusively for obtaining object identifiers.

For more information about creating and configuring internal connection pools, see the following:

- ["Creating an Internal Connection Pool"](#) on page 88-1
- ["Configuring an Internal Connection Pool"](#) on page 89-1

For more information about configuring the type of connection pool your session uses, see ["Configuring External Connection Pooling"](#) on page 85-2.

Configuring External Connection Pools

An external connection pool is a collection of reusable connections to a single data source provided by a JDBC driver or J2EE container.

By default, a session uses internal connection pools (see ["Configuring Internal Connection Pools"](#) on page 79-1). For more information about configuring a session to use an external connection pool, see ["Configuring External Connection Pooling"](#) on page 85-2.

Configuring Exclusive Isolated Client Sessions for Virtual Private Database

This chapter describes the various components that you must configure before you can acquire an exclusive isolated client session from a server session.

Exclusive Isolated Client Session Configuration Overview

[Table 80-1](#) lists the configurable options for isolated sessions.

Table 80-1 Configurable Options for Isolated Client Sessions

Option	Type	TopLink Workbench	Java
"Configuring Cache Isolation at the Descriptor Level" on page 28-37	Basic		✓
"Configuring Connection Policy" on page 77-19	Basic		✓
"PostAcquireExclusiveConnection Event Handler" on page 80-1	Basic		✓
"PreReleaseExclusiveConnection Event Handler" on page 80-2	Basic		✓
"NoRowsModifiedSessionEvent Event Handler" on page 80-3	Basic		✓
"ValidationException Handler" on page 80-3	Basic		✓

These options are used throughout the isolated session life cycle (see ["Isolated Client Session Life Cycle"](#) on page 75-22).

PostAcquireExclusiveConnection Event Handler

TopLink raises this event after an exclusive connection is allocated to an isolated session after the user has logged in to the database with it.

If you are using Oracle Database proxy authentication (see ["Oracle Database Proxy Authentication"](#) on page 84-5), then you do not need to implement this session event handler.

If you are not using Oracle Database proxy authentication, then, as part of the isolated session life cycle, you must implement a `SessionEventListener` for `SessionEvent.PostAcquireExclusiveConnection`.

Note: You must add this session event listener to the server session from which you acquire your isolated client session. You cannot add them to the isolated client session itself. For more information, see ["Configuring Session Event Listeners"](#) on page 77-17

Using Java

The `SessionEvent.PostAcquireExclusiveConnection` event listener is your opportunity to authenticate your user and interact with the underlying database platform: for example, to execute PL/SQL to create VPD packages and set VPD context information.

[Example 80–1](#) illustrates a typical session event listener used to handle `postAcquireExclusiveConnection` events for an isolated session.

Example 80–1 Session Event Listener for an Isolated Session

```
class VPDEventListener extends SessionEventAdaptor{
    public void postAcquireExclusiveConnection(SessionEvent event){
        ClientSession session = (ClientSession)event.getSession();
        // Get property set on the ConnectionPolicy prior to acquiring the connection
        String userLevel = session.getConnectionPolicy().getProperty("userLevel");
        // Make the Stored Procedure call for VPD to set up the Context Information
        session.executeNonSelectingSQL("StoreProcSetContextUser(" + userLevel + ")");
    }
}
```

To get the required user credentials, use `ClientSession` method `getConnectionPolicy` to get the associated `ConnectionPolicy`, and then use `ConnectionPolicy` method `getProperty`. The `ConnectionPolicy` associated with the `ClientSession` should contain all required user credentials (see ["Configuring Connection Policy"](#) on page 77-19).

After you implement the required `SessionEventListener`, add it to the parent server session from which you acquire your isolated client session. For more information, see ["Configuring Session Event Listeners"](#) on page 77-17.

PreReleaseExclusiveConnection Event Handler

`TopLink` raises a `SessionEvent.PreReleaseExclusiveConnection` event after you call the isolated session method `release`.

If you are using Oracle Database proxy authentication (see ["Oracle Database Proxy Authentication"](#) on page 84-5), then you do not need to implement this session event handler.

If you are not using Oracle Database proxy authentication, then as part of the isolated session life cycle, you must implement a `SessionEventListener` for `SessionEvent.PreReleaseExclusiveConnection`.

Note: You must add this session event listener to the server session from which you acquire your isolated client session. You cannot add them to the isolated client session itself. For more information, see ["Configuring Session Event Listeners"](#) on page 77-17

Using Java

The `SessionEvent.PreReleaseExclusiveConnection` event listener gives you an opportunity to interact with the underlying database platform: for example, to perform any VPD-specific cleanup such as executing PL/SQL to delete VPD packages or context information.

[Example 80–1](#) illustrates a typical session event listener used to handle `preReleaseExclusiveConnection` events for an isolated session.

Example 80–2 Session Event Listener for an Isolated Session

```

class VPEventListener extends SessionEventAdaptor{
    public void preReleaseExclusiveConnection(SessionEvent event){
        Session session = event.getSession();
        // Make the Stored Procedure call for VPD to reset the Context Information
        session.executeNonSelectingSQL("StoreProcResetContext()");
    }
}

```

To get the required user credentials, use `ClientSession` method `getConnectionPolicy` to get the associated `ConnectionPolicy`, and then use `ConnectionPolicy` method `getProperty`. The `ConnectionPolicy` associated with the `ClientSession` should contain all required user credentials (see ["Configuring Connection Policy"](#) on page 77-19).

After you implement the required `SessionEventListener`, add it to the parent server session, from which you acquire your isolated client session. For more information, see ["Configuring Session Event Listeners"](#) on page 77-17.

NoRowsModifiedSessionEvent Event Handler

As part of your general error handling strategy, you should implement a `SessionEventListener` for `SessionEvent.NoRowsModifiedSessionEvent`.

`TopLink` raises this event when an update or delete query is executed against the database, but no rows are updated, that is, a zero row count is returned.

If optimistic locking is not enabled and you query the database and violate your VPD security configuration, no exception is thrown: the query simply returns zero rows updated.

If optimistic locking is enabled and you query the database and violate your VPD security configuration, an `OptimisticLockException` is thrown even though the root cause of the failure was a security violation, not an optimistic locking issue.

Note: You must add this session event listener to the server session from which you acquire your isolated client session. You cannot add them to the isolated client session itself. For more information, see ["Configuring Session Event Listeners"](#) on page 77-17

Using Java

This event listener gives you an opportunity to determine whether the update failure was due to a security violation (in which case you should not retry the operation), or due to an optimistic lock issue (in which case a retry may be appropriate).

You can use the existing session event API, such as `getQuery().getResult()`, to get the affected object, if any.

After you implement the required `SessionEventListener`, add it to the parent server session, from which you acquire your isolated client session. For more information, see ["Configuring Session Event Listeners"](#) on page 77-17.

ValidationException Handler

As part of your general error handling strategy, your application should be prepared to handle a `ValidationException` of type `ISOLATED_SESSION_IS_NO_LONGER_AVAILABLE`.

TopLink throws an `ISOLATED_SESSION_IS_NO_LONGER_AVAILABLE` when a client triggers the indirection on an isolated object when the isolated session used to load that object is no longer available, that is, after you call the isolated session method `release`.

For more information, see:

- ["Exception Handlers"](#) on page 75-12
- ["Configuring an Exception Handler"](#) on page 77-12

Configuring Historical Client Sessions

This chapter describes the various components that you must configure in order to be able to use historical client sessions.

For more information about historical client sessions, see ["Historical Client Sessions"](#) on page 75-24.

Historical Client Session Configuration Overview

There are two ways to configure TopLink to access the historical versions of objects maintained by your data source:

- [Configuring Historical Client Sessions Using an Oracle Platform](#)
- [Configuring Historical Client Sessions Using a TopLink HistoryPolicy](#)

Configuring Historical Client Sessions Using an Oracle Platform

Oracle9i Database Server (or later) automatically maintains historical versions of objects and extends SQL with an AS_OF clause used to query this historical data. Oracle refers to these as flashback queries.

If you configure your Session with an OraclePlatform (see ["Configuring a Relational Database Platform at the Session Level"](#) on page 86-1) for Oracle9i Database Server (or later), you can query the historical versions of objects automatically maintained by Oracle Database.

No further session configuration is required.

For more information, see the following:

- ["Acquiring a Historical Client Session"](#) on page 78-7
- ["Historical Queries"](#) on page 96-20.

Configuring Historical Client Sessions Using a TopLink HistoryPolicy

If you use a schema that you designed to maintain historical versions of objects and if that schema can be described by TopLink HistoryPolicy, you can query the historical versions of objects maintained by your database in accordance with your schema.

For more information, see the following:

- ["Configuring a History Policy"](#) on page 28-73
- ["Acquiring a Historical Client Session"](#) on page 78-7

- ["Historical Queries"](#) on page 96-20.

Configuring Session Broker and Client Sessions

This chapter describes the various components that you must configure in order to use session broker sessions.

Session Broker and Client Session Configuration Overview

Table 82–1 lists the configurable options for session broker sessions.

Table 82–1 Configurable Options for Session Broker Session

Option	Type	TopLink Workbench	Java
"Removing, Renaming, or Adding Sessions" on page 82-1	Basic	✓	✓
"Configuring a Primary Mapping Project" on page 77-2	Basic	✓	✓
"Configuring a Session Login" on page 77-4	Basic	✓	✓
"Configuring Logging" on page 77-4	Basic	✓	✓
"Configuring Multiple Mapping Projects" on page 77-9	Advanced	✓	✓
"Configuring a Performance Profiler" on page 77-10	Advanced	✓	✓
"Configuring an Exception Handler" on page 77-12	Advanced	✓	✓
"Configuring Customizer Class" on page 77-13	Advanced	✓	✓
"Configuring the Server Platform" on page 77-14	Advanced	✓	✓
"Configuring Session Event Listeners" on page 77-17	Advanced	✓	✓
"Configuring a Coordinated Cache" on page 91-1	Advanced	✓	✓
"Configuring the Integrity Checker" on page 77-18	Advanced	✓	✓
"Configuring Named Queries at the Session Level" on page 77-21	Basic		✓

Removing, Renaming, or Adding Sessions

Using [TopLink Workbench](#), you can manage the sessions contained by a session broker.

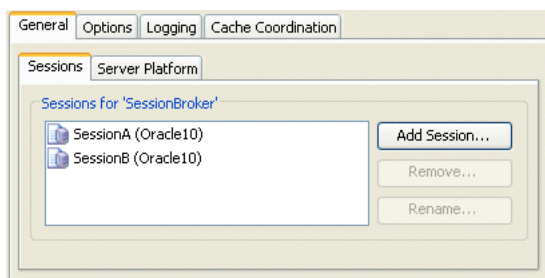
Note: Add only sessions of the same type to any given session broker. Do not mix sessions of different types within a session broker.

Using TopLink Workbench

To add sessions to, remove sessions from, or rename sessions in a session broker, use this procedure:

1. Select a session broker in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.
3. Click the **Sessions** subtab. The Sessions subtab appears.

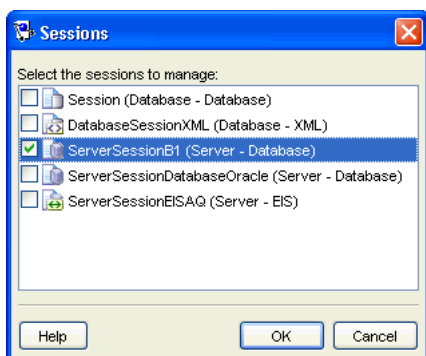
Figure 82–1 General Tab, Sessions Subtab



To manage the sessions in this session broker, choose one of the following:

- To remove a session, select the session in the Sessions tab's list and click **Remove**.
- To rename a session, select the session in the Sessions tab's list and click **Rename**. The Rename dialog box appears. Enter a new name and click **OK**.
- To add a session, click **Add Session**. The Sessions dialog box appears showing a list of all the sessions currently configured in the session configuration that owns this session broker.

Figure 82–2 Sessions Dialog Box



Check the sessions in the Session dialog that you want to add to the session broker and click **OK**.

Configuring Database Sessions

This chapter describes the various components that you must configure in order to use database sessions.

Database Session Configuration Overview

Table 83–1 lists the configurable options for database sessions.

Table 83–1 Configurable Options for Database Session

Option	Type	TopLink Workbench	Java
"Configuring External Connection Pools" on page 83-1	Basic	✓	✓
"Configuring a Primary Mapping Project" on page 77-2	Basic	✓	✓
"Configuring a Session Login" on page 77-4	Basic	✓	✓
"Configuring Logging" on page 77-4	Basic	✓	✓
"Configuring Multiple Mapping Projects" on page 77-9	Advanced	✓	✓
"Configuring a Performance Profiler" on page 77-10	Advanced	✓	✓
"Configuring an Exception Handler" on page 77-12	Advanced	✓	✓
"Configuring Customizer Class" on page 77-13	Advanced	✓	✓
"Configuring the Server Platform" on page 77-14	Advanced	✓	✓
"Configuring Session Event Listeners" on page 77-17	Advanced	✓	✓
"Configuring a Coordinated Cache" on page 91-1	Advanced	✓	✓
"Configuring the Integrity Checker" on page 77-18	Advanced	✓	✓
"Configuring Named Queries at the Session Level" on page 77-21	Basic		✓

Configuring External Connection Pools

Unlike a server session, a database session does not provide internal connection pools. A database session only has a single database connection that it uses for its life cycle.

Oracle recommends that you use a server and client session in a three-tier environment. Alternatively, you can use a database session with an external connection pool (see "[Configuring External Connection Pooling](#)" on page 85-2): in this case, you should allocate a new database session per user/thread or request.

WARNING: Do not allow the concurrent use of a database session by multiple users/threads.

The usage of an external connection pool reduces the number of the database session login and logout attempts to acquire the database connection.

Part XVII

Data Access

This part describes how TopLink defines connections to a data source. It contains the following chapters:

- [Chapter 84, "Understanding Data Access"](#)
This chapter describes each of the different TopLink data source login types and important data access concepts.
- [Chapter 85, "Configuring a Data Source Login"](#)
This chapter explains how to configure TopLink data source login options common to two or more data source login types.
- [Chapter 86, "Configuring a Database Login"](#)
This chapter explains how to configure a TopLink database login for a session used in a relational project.
- [Chapter 87, "Configuring an EIS Login"](#)
This chapter explains how to configure a TopLink EIS login for a session used in an EIS project.
- [Chapter 88, "Creating an Internal Connection Pool"](#)
This chapter explains how to create an internal connection pool.
- [Chapter 89, "Configuring an Internal Connection Pool"](#)
This chapter explains how to configure an internal connection pool.

Understanding Data Access

One of the most important functions of a session is to provide access to a data source. This chapter explains session components specific to accessing a data source.

This chapter explains the following:

- [Data Access Concepts](#)
- [Understanding Data Access API](#)

Data Access Concepts

This section describes concepts unique to TopLink data access, including the following:

- [Externally Managed Transactional Data Sources](#)
- [Data Source Login Types](#)
- [Data Source Platform Types](#)
- [Authentication](#)
- [Connections](#)
- [Connection Pools](#)

Externally Managed Transactional Data Sources

A TopLink transactional data source is *externally managed* if the connection pool is managed by a transaction service (such as an application server controlled transaction or a JTA transaction). A JTA managed data source or connection pool is commonly used in J2EE applications and normally required in EJB applications. When using an externally managed connection pool:

- Configure the session to use an `ExternalTransactionController` to integrate TopLink's unit of work with the external transaction service (see ["Integrating the Unit of Work With an External Transaction Service"](#) on page 102-20).
- Use the `external-transaction-control` option to specify the connection's login and inform TopLink that the connection is maintained by the external controller (see ["Configuring External Connection Pooling"](#) on page 85-2).
- You may need to configure the TopLink read connection pool or sequence connection pool to use a non-JTA connection pool in order to avoid transactional overhead (see ["Default \(Write\) and Read Connection Pools"](#) on page 84-8).

For more information on transactional data sources, see the following:

- ["JTA Controlled Transactions"](#) on page 100-3
- ["OTS Controlled Transactions"](#) on page 100-3
- ["CMP Controlled Transactions"](#) on page 100-3

Refer to [Chapter 100, "Understanding TopLink Transactions"](#) for more information on TopLink transactions.

Data Source Login Types

The login (if any) associated with a session determines how the TopLink runtime connects to the project's data source.

A login includes details of data source access, such as authentication, use of connection pools, and use of external transaction controllers. A `Login` owns a data source platform.

A data source platform includes options specific to a particular data source including such as binding, use of native SQL, use of batch writing, and sequencing. For more information about platforms, see ["Data Source Platform Types"](#) on page 84-3.

For projects that do not persist to a data source, a login is not required. For projects that do persist to a data source, a login is always required.

In TopLink Workbench, the project type determines the type of login that the project uses, if applicable.

You can use a login in a variety of roles. A login's role determines where and how you create it. The login role you choose depends on the type of project you are creating and how you intend to use the login:

- ["Non-CMP Session Role: Session Login"](#) on page 20-3
- ["CMP Deployment Role: Deployment Login"](#) on page 20-3
- ["Development Role: Development Login"](#) on page 20-4

There is a session login type for each project type that persists to a data source:

- `DatabaseLogin`
- `EISLogin`

Note that there is no XML login. TopLink XML projects are used for nonpersistent, in-memory object to XML data transformation and consequently there is no data source to log in to. For more information about persistent and nonpersistent projects, see ["Persistent and Nonpersistent Projects"](#) on page 20-2.

For additional information, see the following:

- ["Projects and Login"](#) on page 20-3
- ["Configuring Common Data Source Login Options"](#) on page 85-1

DatabaseLogin

If you are creating a project that accesses a relational database, you must configure the project with a `DatabaseLogin`. Your choice of `DatabasePlatform` further customizes your project for a particular type of database (see ["Database Platforms"](#) on page 84-3).

For more information, see ["Database Login Configuration Overview"](#) on page 86-1.

EISLogin

If you are creating a project that accesses a nonrelational data source using a J2C adapter, you must configure the project with an `EISLogin`. Your choice of `EISPlatform` further customizes your project for a particular J2C adapter and specifies what record type `TopLink` uses to exchange data with the EIS (see ["EIS Platforms"](#) on page 84-4).

For more information, see ["EIS Login Configuration Overview"](#) on page 87-1.

Data Source Platform Types

`TopLink` abstracts the details of your underlying data source using data source platform classes. `TopLink` provides the following data source platforms:

- [Database Platforms](#)
- [EIS Platforms](#)

A data source platform is owned by your project's `Login`. For more information about logins, see ["Data Source Login Types"](#) on page 84-2.

In general, `TopLink Workbench` provides minimal access to a platform API. To configure most platform options, you must use an amendment method (see ["Configuring Amendment Methods"](#) on page 28-78), or a `preLogin` event listener (see ["Managing Session Events With the Session Event Manager"](#) on page 75-5).

Database Platforms

`TopLink` interacts with databases using structured query language (SQL). Because each database platform uses its own variation on the basic SQL language, `TopLink` must adjust the SQL it uses to communicate with the database to ensure that the application runs smoothly.

The type of database platform you choose determines the specific means by which the `TopLink` runtime accesses the database, including the type of Java Database Connectivity (JDBC) driver to use. JDBC is an application programming interface (API) that gives Java applications access to a database. `TopLink` relational projects rely on JDBC connections to read objects from, and write objects to, the database. `TopLink` applications use either individual JDBC connections or a JDBC connection pool (see ["Connection Pools"](#) on page 84-7), depending on the application architecture.

`TopLink` provides a variety of database-specific platforms that let you customize your project for your target database.

Oracle Database platforms are located in `oracle.toplink.platform.database.oracle` package and include the following:

- `OraclePlatform`
- `Oracle8Platform`
- `Oracle9Platform`

Non-Oracle Database platforms are located in `oracle.toplink.platform.database` package and include the following:

- `AccessPlatform` for Microsoft Access databases
- `AttunityPlatform` for Attunity Connect JDBC drivers
- `CloudscapePlatform`

- `DB2MainframePlatform`
- `DB2Platform`
- `DBasePlatform`
- `HSQLPlatform`
- `InformixPlatform`
- `MySQL4Platform`
- `PointBasePlatform`
- `SQLAnywherePlatform`
- `SQLServerPlatform`
- `SybasePlatform`

Specify your database platform at the project level (see ["Configuring Relational Database Platform at the Project Level"](#) on page 23-2) for all sessions, or override this project-level configuration at the session level (see ["Configuring a Relational Database Platform at the Session Level"](#) on page 86-1).

If you set your database platform in TopLink Workbench, then TopLink Workbench manages the database platform configuration for you automatically.

EIS Platforms

TopLink interacts with an EIS data source indirectly by way of a J2C adapter. TopLink abstracts the details of an EIS data source using the `oracle.toplink.eis.EISPlatform` class.

The type of EIS platform you choose determines the specific means by which the TopLink runtime accesses the EIS, including the type of J2C adapter to use. TopLink EIS projects rely on EIS connections to read objects from, and write objects to, the EIS. TopLink applications use individual EIS connections returned by the EIS connection factory specified by the EIS platform.

TopLink provides a variety of `EISPlatform` classes that let you customize your project for your target EIS.

EIS platforms for production are located in `oracle.toplink.eis.adapters` package and include the following:

- `oracle.toplink.eis.adapters.aq.AQPlatform` to access an EIS using Oracle Advanced Queuing messages.
- `oracle.toplink.eis.adapters.attunity.AttunityPlatform` to access an EIS using an Attunity J2C adapter.
- `oracle.toplink.eis.adapters.jms.JMSPlatform` to access an EIS using JMS messages.
- `oracle.toplink.eis.adapters.mqseries.MQPlatform` to access an EIS using IBM MQSeries messages.

EIS platforms for testing are also located in `oracle.toplink.eis.adapters` and include the following:

- `oracle.toplink.eis.adapters.blackbox.BlackBoxPlatform` for testing your EIS project with the Sun BlackBox reference adapter using indexed records only.

- `oracle.toplink.eis.adapters.xmlfile.XMLFilePlatform` for testing your EIS project with an EIS emulated as one or more XML files in the local file system using XML records.

Specify your EIS platform at the session level (see ["Configuring an EIS Data Source Platform at the Session Level"](#) on page 87-1).

If you set your platform in TopLink Workbench, then TopLink Workbench manages the EIS platform configuration for you automatically.

Authentication

Authentication is the means by which a data source validates a user's identity and determines whether or not the user has sufficient privileges to perform a given action.

For two-tier applications, simple JDBC authentication is usually sufficient (see ["Simple JDBC Authentication"](#) on page 84-5).

For three-tier applications, you can use simple JDBC authentication or, proxy authentication (see ["Oracle Database Proxy Authentication"](#) on page 84-5) when using the Oracle Call Interface (OCI) JDBC driver.

Authentication plays a central role in data security and user accountability and auditing (see ["Auditing"](#) on page 84-6).

Simple JDBC Authentication

When you configure a TopLink database login with a user name and password (["Configuring User Name and Password"](#) on page 85-1), TopLink provides these credentials to the JDBC driver that you configure your application to use (see ["Configuring Database Login Connection Options"](#) on page 86-2).

By default, TopLink writes passwords to and reads them from the `sessions.xml` file in encrypted form using JCE encryption. Optionally, you can configure a different encryption class (see ["Configuring Password Encryption"](#) on page 85-2).

Oracle Database Proxy Authentication

TopLink supports proxy authentication with the Oracle Database in JSE applications and JEE applications using OC4J native or managed data sources with Oracle JDBC driver release 10.1.0.2.0 or later and external connection pools (see ["External Connection Pools"](#) on page 84-8) only.

Note: TopLink does not support Oracle Database proxy authentication with JTA.

Oracle Database proxy authentication delivers the following security benefits:

- A limited trust model, by controlling the users on whose behalf middle tiers can connect, and the roles the middle tiers can assume for the user.
- Scalability, by supporting user sessions through Oracle Call Interface (OCI) and thick JDBC, and eliminating the overhead of reauthenticating clients.
- Accountability, by preserving the identity of the real user through to the database, and enabling auditing of actions taken on behalf of the real user.
- Flexibility, by supporting environments in which users are known to the database, and in which users are merely "application users" of which the database has no awareness.

Note: Oracle Database supports proxy authentication in three-tiers only; it does not support it across multiple middle tiers.

For more information about authentication in an Oracle Database, see "Preserving User Identity in Multitiered Environments" in the *Oracle Database Security Guide*.

Configure your TopLink database login to use proxy authentication (see "[Configuring Oracle Database Proxy Authentication](#)" on page 86-12) to do the following:

- address the complexities of authentication in a three-tier architecture (such as client-to-middle-tier and middle-tier-to-database authentication, and client reauthentication through the middle -tier to the database)
- enhance database audit information (for even triggers and stored procedures) by using a specific user for database operations, rather than the generic pool user
- simplify VPD/OLS configuration (see "[Isolated Client Sessions and Oracle Virtual Private Database \(VPD\)](#)" on page 75-20) by using a proxy user, rather than setting user information directly in the session context with stored procedures

Auditing

Regardless of what type of authentication you choose, TopLink logs the name of the user associated with all database operations. [Example 84-1](#) shows the CONFIG level TopLink logs when a `ServerSession` connects through the main connection for the sample user "scott", and a `ClientSession` uses proxy connection "jeff".

Example 84-1 TopLink Logs with Oracle Database Proxy Authentication

```
[TopLink
Config]--ServerSession(13)--Connection(14)--Thread(Thread[main,5,main])--connecting(DatabaseL
ogin( platform=>Oracle9Platform user name=> "scott" connector=>OracleJDBC10_1_0_
2ProxyConnector datasource name=>DS))
[TopLink Config]--ServerSession(13)--Connection(34)--Thread(Thread[main,5,main])--Connected:
jdbc:oracle:thin:@localhost:1521:orcl
User: SCOTT
[TopLink
Config]--ClientSession(53)--Connection(54)--Thread(Thread[main,5,main])--connecting(DatabaseL
ogin(platform=>Oracle9Platform user name=> "scott" connector=>OracleJDBC10_1_0_
2ProxyConnector datasource name=>DS))
[TopLink Config]--ClientSession(53)--Connection(56)--Thread(Thread[main,5,main])--Connected:
jdbc:oracle:thin:@localhost:1521:orcl
User: jeff
```

For more information on configuring TopLink log level and log options, see "[Configuring Logging](#)" on page 77-4.

Your database server likely provides additional user auditing options. Consult your database server documentation for details.

Alternatively, you may consider using the TopLink unit of work in conjunction with your database schema for auditing purposes (see "[Implementing User and Date Auditing with the Unit of Work](#)" on page 102-20).

Connections

A connection is an object that provides access to a data source by way of the driver you configure your application to use (see "[Configuring Database Login Connection Options](#)" on page 86-2). Relational projects use JDBC to connect to the data source; EIS and XML projects use JCA. TopLink uses the interface

`oracle.toplink.internal.databaseaccess.Accessor` to wrap data source connections. This interface is accessible from certain events (see ["Descriptor Event Manager"](#) on page 26-8).

Typically, when using a server session, TopLink uses a different connection for both reading and writing. This lets you use nontransactional connections for reading and avoid maintaining connections when not required. See ["Reading Through the Write Connection"](#) on page 102-27 and ["Exclusive Write Connections"](#) on page 77-19 for more information.

By default, a TopLink server session acquires connections lazily: that is, only during the commit operation of a unit of work. Alternatively, you can configure TopLink to acquire a write connections at the time you acquire a client sessions (see ["Lazy Connection Acquisition"](#) on page 77-20).

Connections can be allocated from internal or external connection pools (see ["Connection Pools"](#) on page 84-7).

Connection Pools

A **connection pool** is a service that creates and maintains a shared collection (pool) of data source connections on behalf of one or more clients. The connection pool provides a connection to a process on request, and returns the connection to the pool when the process is finished using it. When it is returned to the pool, the connection is available for other processes. Because establishing a connection to a data source can be time-consuming, reusing such connections in a connection pool can improve performance.

TopLink uses connection pools to manage and share the connections used by server and client sessions. This feature reduces the number of connections required and allows your application to support many clients.

You can configure your session to use internal connection pools provided by TopLink or external connection pools provided by a JDBC driver or J2EE container.

You can use connection pools in your TopLink application for a variety of purposes, such as reading, writing, sequencing, and other application-specific functions.

This section describes the following:

- [Internal Connection Pools](#)
- [External Connection Pools](#)
- [Default \(Write\) and Read Connection Pools](#)
- [Sequence Connection Pools](#)
- [Application-Specific Connection Pools](#)

Internal Connection Pools

For non-J2EE applications, you typically use *internal* connection pools. By default, TopLink sessions use internal connection pools.

Using internal connection pools, you can use TopLink Workbench to configure the default (write) and read connection pools (see ["Default \(Write\) and Read Connection Pools"](#) on page 84-8) and you can create additional connection pools for object identity (see ["Sequence Connection Pools"](#) on page 84-8), or any other purpose (see ["Application-Specific Connection Pools"](#) on page 84-9).

Using internal connection pools, you can optimize the creation of read connections for applications that read data only to display it and only infrequently modify data (see ["Configuring a Nontransactional Read Login"](#) on page 89-3).

For information on selecting the type of connection pool to use, see ["Configuring External Connection Pooling"](#) on page 85-2.

For more information on creating and configuring internal connection pools, see:

- ["Internal Connection Pool Creation Overview"](#) on page 88-1
- ["Internal Connection Pool Configuration Overview"](#) on page 89-1

External Connection Pools

For J2EE applications, you typically use *external* connection pools.

If you are using an external transaction controller (JTA), you must use external connection pools to integrate with the JTA (see ["Integrating the Unit of Work With an External Transaction Service"](#) on page 102-20).

Using external connection pools, you can use either TopLink Workbench or Java to configure the default (write) and read connection pools (see ["Default \(Write\) and Read Connection Pools"](#) on page 84-8) and create additional connection pools for object identity (see ["Sequence Connection Pools"](#) on page 84-8), or any other purpose (see ["Application-Specific Connection Pools"](#) on page 84-9).

For more information on selecting the type of connection pool to use, see ["Configuring External Connection Pooling"](#) on page 85-2.

Default (Write) and Read Connection Pools

A server session provides a read connection pool and a write connection pool. These could be different pools, or if you use external connection pooling, the same connection pool.

All read queries use connections from the read connection pool and all queries that write changes to the data source use connections from the write connection pool. You can configure attributes of the default read and write connection pools.

Whenever a new connection is established, TopLink uses the connection configuration you specify in your session's `DatasourceLogin`. Alternatively, when you use an external transaction controller, you can define a separate connection configuration for a read connection pool to avoid the additional overhead, if appropriate (see ["Configuring a Nontransactional Read Login"](#) on page 89-3).

For more information on configuring read and write connection pools, see ["Internal Connection Pool Configuration Overview"](#) on page 89-1.

Sequence Connection Pools

An essential part of maintaining object identity (see ["Cache Type and Object Identity"](#) on page 90-3) is sequencing—managing the assignment of unique values to distinguish one instance from another. For more information, see ["Projects and Sequencing"](#) on page 20-4.

Sequencing involves reading and writing a special sequence resource maintained by your data source.

By default, TopLink includes sequence operations in a separate transaction. This avoids complications during the write transaction, which may lead to deadlocks over the sequence resource. However, when using an external transaction controller (such as a JTA data source or connection pool), TopLink cannot use a different transaction

for sequencing. Use a sequence connection pool to configure a non-JTA transaction pool for sequencing. This is required only for table sequencing—not native sequencing.

In each server session, you can create one connection pool, called a sequence connection pool, that TopLink uses exclusively for sequencing. With a sequence connection pool, TopLink satisfies a request for a new object identifier outside of the transaction from which the request originates. This allows TopLink to immediately commit an update to the sequence resource, which avoids deadlocks.

Note: If you use a sequence connection pool and the original transaction fails, the sequence operation does not roll back.

You should use a sequence connection pool, if the following applies:

- You use table sequencing (that is, non-native sequencing). See ["Table Sequencing"](#) on page 20-16 and ["Unary Table Sequencing"](#) on page 20-17 for more information.
- You use external transaction controller (JTA).

You should not use a sequence connection pool, if the following applies:

- You do not use sequencing, or use the data source's native sequencing (see ["Native Sequencing With an Oracle Database Platform"](#) on page 20-18 and ["Native Sequencing With a Non-Oracle Database Platform"](#) on page 20-19).
- You have configured the sequence table to avoid deadlocks.
- You use non-JTA data sources.

For more information, see

- ["Internal Connection Pool Creation Overview"](#) on page 88-1
- ["Internal Connection Pool Configuration Overview"](#) on page 89-1

Application-Specific Connection Pools

When you use internal TopLink connection pools in a session, you can create one or more connection pools that you can use for any application purpose. These are called named connection pools, as you can give them any name you want and use them for any purpose.

Typically, use these named connection pools to provide pools of different security levels. For example, the "default" connection pool may only allow access to specific tables but the "admin" connection pool may allow access to all tables.

For more information, see the following:

- ["Internal Connection Pool Creation Overview"](#) on page 88-1
- ["Internal Connection Pool Configuration Overview"](#) on page 89-1
- ["Acquiring a Client Session that Uses a Named Connection Pool"](#) on page 78-8

Understanding Data Access API

This section describes the following:

- [Login Inheritance Hierarchy](#)
- [Platform Inheritance Hierarchy](#)

Login Inheritance Hierarchy

[Example 84-2](#) illustrates the login types that are derived from abstract class `oracle.toplink.sessions.DatasourceLogin`.

Example 84-2 Login Inheritance Hierarchy

```
class oracle.toplink.sessions.DatasourceLogin
    class oracle.toplink.sessions.DatabaseLogin
    class oracle.toplink.eis.EISLogin
```

Platform Inheritance Hierarchy

[Example 84-3](#) illustrates the platform type class hierarchy.

Example 84-3 Platform Inheritance Hierarchy

```
oracle.toplink.platform.database
    AccessPlatform
    AttunityPlatform
    CloudscapePlatform
    DatabasePlatform
    DB2MainframePlatform
    DB2Platform
    DBasePlatform
    HSQLPlatform
    InformixPlatform
    PointBasePlatform
    SQLAnywherePlatform
    SQLServerPlatform
    SybasePlatform
oracle.toplink.platform.database.oracle
    Oracle8Platform
    Oracle9Platform
    OraclePlatform
```

Configuring a Data Source Login

This chapter describes how to configure TopLink data source logins.

[Table 85–1](#) lists the types of TopLink data source logins that you can configure and provides a cross-reference to the type-specific chapter that lists the configurable options supported by that type.

Table 85–1 *Configuring TopLink Data Source Logins*

If you are configuring a...	See...
DatabaseLogin	Chapter 86, "Configuring a Database Login"
EISLogin	Chapter 87, "Configuring an EIS Login"

[Table 85–2](#) lists the configurable options shared by two or more TopLink data source login types.

When using the `sessions.xml` file to configure login information, TopLink will override any login information in the `project.xml` and instead use the information from the `sessions.xml` configuration. For more information, see ["Understanding Data Access"](#) on page 84-1.

Configuring Common Data Source Login Options

[Table 85–2](#) lists the configurable options shared by two or more TopLink data source login types. In addition to the configurable options described here, you must also configure the options described for the specific [Data Source Login Types](#), as shown in [Table 85–1](#)

Table 85–2 *Common Data Source Login Options*

Option	Type	TopLink Workbench	Java
"Configuring User Name and Password" on page 85-1	Basic	✓	✓
"Configuring Password Encryption" on page 85-2	Advanced		✓
"Configuring External Connection Pooling" on page 85-2	Advanced	✓	✓
"Configuring Properties" on page 85-4	Advanced	✓	✓
"Configuring a Default Null Value at the Login Level" on page 85-5	Advanced		✓

Configuring User Name and Password

Optionally, you can specify the user name and password of a login.

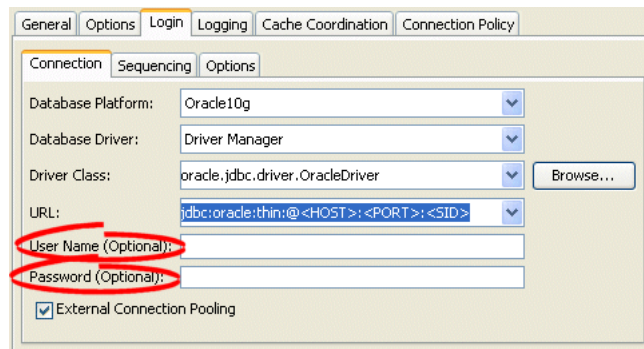
If you specify a password using TopLink Workbench, enter the plain text (not encrypted) value. By default, TopLink Workbench writes passwords to and reads passwords from the `sessions.xml` file in encrypted form using JCE encryption. For information on configuring password encryption, see ["Configuring Password Encryption"](#) on page 85-2.

Using TopLink Workbench

To specify a user name and password, use this procedure:

1. Select a server or database session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Login** tab. The Login tab appears.
3. Click the **Connection** subtab. The Connection subtab appears.

Figure 85–1 Login Tab, Connection Subtab, User Name and Password Fields



Enter a user name and password in plain text (not encrypted).

Configuring Password Encryption

By default, passwords are written to and read from the `sessions.xml` file in encrypted form using JCE encryption.

Currently, TopLink Workbench does not support specifying the encryption class used. To change the encryption class used, you must modify the login in Java using a `preLogin` event listener.

Using Java

To specify the encryption class that TopLink should use to write a password to or read a password from the `sessions.xml` file, use `DatasourceLogin` method `setEncryptionClassName`, passing in the name of the encryption class as a `String`.

To configure a `DatasourceLogin` with an encrypted password, use `DatasourceLogin` method `setEncryptedPassword`, passing in the encrypted password as a `String`.

Configuring External Connection Pooling

For non-J2EE applications, you typically use internal connection pools provided by TopLink (see ["Internal Connection Pools"](#) on page 84-7). In this case, you can use

TopLink Workbench to configure connection pool options and to create a sequence connection pool and application-specific (named) connection pools.

For J2EE applications, you typically use external connection pools provided by a JDBC driver or J2EE container (see ["External Connection Pools"](#) on page 84-8). Optionally, you can configure a read connection pool to use a nontransactional login, and you can configure a sequence connection pool to use a separate (preferably nontransactional) login of its own.

Because JTA external transaction controllers are dependent upon the external transaction service that the application server provides, you must configure TopLink to use external connection pools if you are using an external transaction controller (see ["Integrating the Unit of Work With an External Transaction Service"](#) on page 102-20).

External connection pools enable your TopLink application to do the following:

- Integrate into a J2EE-enabled system.
- Integrate with JTA transactions (JTA transactions require a JTA-enabled data source).
- Leverage a shared connection pool in which multiple applications use the same data source.
- Use a data source configured and managed directly on the server.

For more information about connection pools, see ["Connection Pools"](#) on page 84-7.

Using TopLink Workbench

To specify if the session login uses external connection pooling, use this procedure:

1. Configure a data source on the application server.

If you are using the external connection pool with an external transaction controller (see ["Configuring the Server Platform"](#) on page 77-14), be sure to configure a JTA-enabled data source.

For more information, see your J2EE container documentation.

2. Select a server or database session in the **Navigators**. Its properties appear in the Editor.
3. Click the **Login** tab. The Login tab appears.
4. Click the **Connection** subtab. The Connection subtab appears.

Figure 85–2 Login Tab, Connection Subtab, External Connection Pooling Field, Database Driver

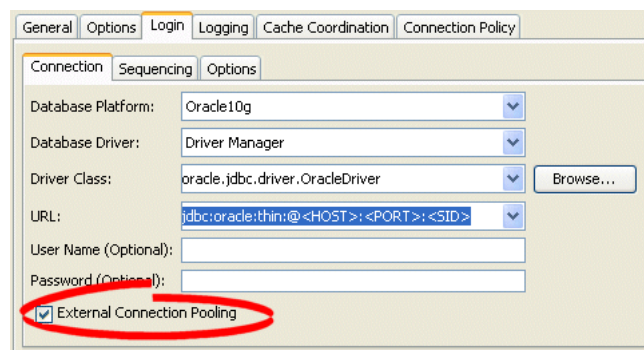


Figure 85-3 Connection Tab, External Connection Pooling Field, J2EE Data Source

The screenshot shows the 'Connection' tab of the Oracle TopLink Workbench configuration window. The 'Database Platform' is set to 'Oracle10g', 'Database Driver' is 'J2EE Data Source', and 'Data Source Name' is 'jdbc/OracleDS'. The 'User Name (Optional)' and 'Password (Optional)' fields are empty. The 'External Connection Pooling' checkbox is checked and highlighted with a red circle.

Specify if this login uses External Connection Pooling. For a database driver, external connection pooling is optional. For a J2EE data source, external connection pooling is mandatory.

Configuring Properties

For all `DatasourceLogin` types, you can specify custom named values, called properties. Some data sources require additional, driver-specific properties not supported in the `DatasourceLogin` API (for example, see ["JDBC Driver Properties Optimization"](#) on page 11-14). Add these properties to the `DatasourceLogin` so that TopLink can pass them to the driver.

For relational sessions, you must first enable advanced option **Use Properties** (see ["Configuring Advanced Options"](#) on page 86-11).

For EIS sessions, properties are always enabled.

Note: Do not set a password as a property. Always use TopLink Workbench or `DatabaseLogin` method `setPassword`. For more information on configuring a password, see ["Configuring User Name and Password"](#) on page 85-1.

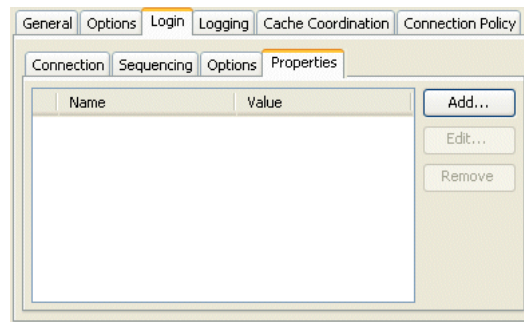
When using TopLink Workbench, you can only set character values, which TopLink returns as `String` objects (see ["Using TopLink Workbench"](#) on page 85-4).

When using Java, you can set any `Object` value (see ["Using Java"](#) on page 85-5).

Using TopLink Workbench

To specify arbitrary named value pairs that TopLink associates with a `DatasourceLogin`, use this procedure:

1. Select a server or database session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Login** tab. The Login tab appears.
3. If necessary, enable support for properties:
 - For relational sessions, you must first enable advanced option **Use Properties** (see ["Configuring Advanced Options"](#) on page 86-11)
 - For EIS sessions, properties are always enabled.
4. Click the **Properties** subtab. The Properties subtab appears.

Figure 85–4 Login Tab, Properties Subtab

To add (or change) a new **Name/Value** property, click **Add** (or **Edit**). Add Property dialog box appears.

Use the following information to add or edit a login property on the Add Property dialog box:

Option	Description
Name	The name by which TopLink retrieves the property value using the <code>DatasourceLogin</code> method <code>getProperty</code> .
Value	The value TopLink retrieves using the <code>DatasourceLogin</code> method <code>getProperty</code> passing in the corresponding property name. Using TopLink Workbench, you can set only character values that TopLink returns as <code>String</code> objects.

To delete an existing property, select the **Name/Value** row and click **Remove**.

Using Java

Using Java, you can set any `Object` value using `DatasourceLogin` method `setProperty`. To remove a property, use `DatasourceLogin` method `removeProperty`.

Configuring a Default Null Value at the Login Level

A default null value is the Java `Object` type and value that TopLink uses instead of `null` when TopLink reads a `null` value from a data source.

When you configure a default null value at the login level, it applies to all mappings used in a session. In this case, TopLink uses it to translate in one direction only: when TopLink reads `null` from the data source, it converts this `null` to the specified type and value.

You can also use TopLink to set a default null value on a per-mapping basis (see ["Configuring a Default Null Value at the Mapping Level"](#) on page 35-12).

Note: A default null value must be an `Object`. To specify a primitive value (such as `int`), you must use the corresponding `Object` wrapper (such as `Integer`).

Using Java

Using Java API, you can configure a default null value for all mappings used in a session with the `DatabaseLogin` method `setDefaultNullValue(Class, Object)`.

For example:

```
// Defaults all null String values read from the database to empty String  
session.getLogin().setDefaultNullValue(String.class, "");
```


Configuring a Database Login

In a relational database project, TopLink retrieves the table information from the database, for each descriptor. Each TopLink Workbench project contains an associated database. You can create multiple logins for each database.

Database Login Configuration Overview

Table 86-1 lists the configurable options for a database login.

Table 86-1 Configurable Options for Database Login

Option	Type	TopLink Workbench	Java
"Configuring a Relational Database Platform at the Session Level" on page 86-1	Basic	✓	✓
"Configuring Database Login Connection Options" on page 86-2	Basic	✓	✓
"Configuring Sequencing at the Session Level" on page 86-4	Basic	✓	✓
"Configuring JDBC Options" on page 86-9	Basic	✓	✓
"Configuring User Name and Password" on page 85-1	Basic	✓	✓
"Configuring a Table Qualifier" on page 86-8	Advanced	✓	✓
"Configuring Advanced Options" on page 86-11	Advanced	✓	✓
"Configuring Password Encryption" on page 85-2	Advanced	✓	✓
"Configuring External Connection Pooling" on page 85-2	Advanced	✓	✓
"Configuring Properties" on page 85-4	Advanced	✓	✓
"Configuring Oracle Database Proxy Authentication" on page 86-12	Advanced		✓

Configuring a Relational Database Platform at the Session Level

For each database session, you must specify the database platform (such as Oracle9i Database Server). This platform configuration overrides the platform at the project level, if configured.

For more information, see the following:

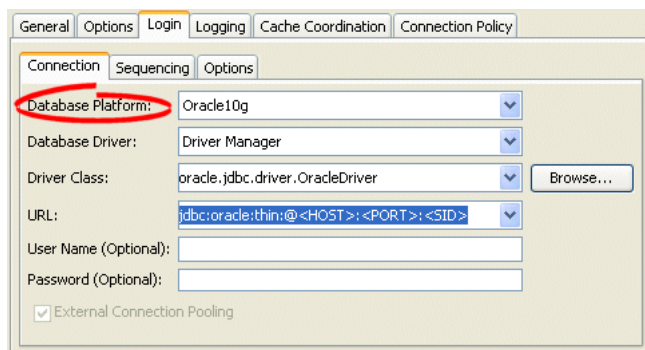
- "Configuring Relational Database Platform at the Project Level" on page 23-2
- "Data Source Platform Types" on page 84-2

Using TopLink Workbench

To specify the database platform options for a relational server (or database) session login, use this procedure:

1. Select a relational server (or database) session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Login** tab. The Login tab appears.
3. Click the **Connection** subtab. The Connection subtab appears.

Figure 86–1 Login Tab, Connection Subtab, Database Platform Option



Select the database platform from the menu of options. This menu includes all instances of `DatabasePlatform` in the TopLink classpath.

Configuring Database Login Connection Options

You configure connection information at the session level for a non-CMP TopLink application. This information is stored in the `sessions.xml` file. The TopLink runtime uses this information whenever you perform a persistence operation using the session in your non-CMP TopLink application.

This connection configuration overrides the connection information at the project level, if configured. For more information about project-level configuration, see ["Configuring Development and Deployment Logins"](#) on page 23-6.

This connection configuration is overridden by the connection information at the connection pool level. For more information, see ["Configuring Connection Pool Connection Options"](#) on page 89-4.

Using TopLink Workbench

To specify the connection options for a relational server (or database) session login, use this procedure:

1. Select a relational server (or database) session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Login** tab. The Login tab appears.
3. Click the **Connection** subtab. The Connection subtab appears.

Figure 86-2 Login Tab, Connection Subtab, Database Driver

The screenshot shows the 'Login' tab with the 'Connection' subtab selected. The 'Database Platform' is set to 'Oracle10g'. The 'Database Driver' is set to 'Driver Manager'. The 'Driver Class' is set to 'oracle.jdbc.driver.OracleDriver'. The 'URL' is set to 'jdbc:oracle:thin:@<HOST>:<PORT>:<SID>'. The 'User Name (Optional)' and 'Password (Optional)' fields are empty. The 'External Connection Pooling' checkbox is checked.

Figure 86-3 Login Tab, Connection Subtab, J2EE Datasource

The screenshot shows the 'Login' tab with the 'Connection' subtab selected. The 'Database Platform' is set to 'Oracle10g'. The 'Database Driver' is set to 'J2EE Data Source'. The 'Data Source Name' is set to 'jdbc/OracleDS'. The 'User Name (Optional)' and 'Password (Optional)' fields are empty. The 'External Connection Pooling' checkbox is checked.

4. Complete each field on the Connection subtab.

Use the following information to enter data in the driver fields on the tab:

Field	Description
Database Driver	Specify the appropriate database driver: ■ Driver Manager: specify this option to configure the driver class and URL used to connect to the database. In this case, you must configure the Driver Class and Driver URL fields. ■ J2EE Data Source: specify this option to use a J2EE data source already configured on your target application server. In this case, you must configure the Datasource Name field. Note: If you select J2EE Datasource, you must use external connection pooling. You cannot use internal connection pools with this Database Driver option (for more information, see "Configuring External Connection Pooling" on page 85-2).
Driver Class ¹	Configure this field when Database Driver is set to Driver Manager. Select from the menu of options. This menu includes all JDBC drivers in the TopLink classpath.
Driver URL ¹	Configure this field when Database Driver is set to Driver Manager. Select from the menu of options relevant to the selected Driver Class, and edit the URL to suit your data source.

Field	Description
Data Source Name ²	Configure this field when Database Driver is set to J2EE Datasource. Specify any valid JNDI name that identifies the J2EE data source preconfigured on your target application server (example: jdbc/EmployeeDB). By convention, all such names should resolve to the JDBC subcontext (relative to the standard java:comp/env naming context that is the root of all provided resource factories).

¹ Applicable only when **Database Driver** is set to **Driver Manager**.

² Applicable only when **Database Driver** is set to **J2EE Datasource**.

Configuring Sequencing at the Session Level

You configure TopLink sequencing at the session or project level to tell TopLink how to obtain sequence values: that is, what type of sequences to use.

In a CMP project, you do not configure a session directly: in this case, you must configure sequences at the project level (see ["Configuring Sequencing at the Project Level"](#) on page 23-3). In a non-CMP project, you can configure a session directly: in this case, you can use session-level sequence configuration to override project-level sequence configuration, on a session-by-session basis, if required.

Using TopLink Workbench (see ["Using TopLink Workbench"](#) on page 86-4), you can configure table sequencing (see ["Table Sequencing"](#) on page 20-16) and native sequencing (["Native Sequencing With an Oracle Database Platform"](#) on page 20-18 and ["Native Sequencing With a Non-Oracle Database Platform"](#) on page 20-19), and you can configure a preallocation size that applies to all sequences (see ["Sequencing and Preallocation Size"](#) on page 20-20).

Using Java (see ["Using Java"](#) on page 86-5), you can configure any sequence type that TopLink supports (["Sequencing Types"](#) on page 20-16). You can create any number and combination of sequences. You can create a sequence object explicitly or use the default sequence that the platform creates. You can associate the same sequence with more than one descriptor and you can configure a separate preallocation size for each descriptor's sequence.

If you are migrating a BEA WebLogic CMP application to OC4J and TopLink persistence (see ["Migrating BEA WebLogic Persistence to OC4J TopLink Persistence"](#) on page 7-16), the TopLink migration tool does not migrate BEA WebLogic single column sequence tables to TopLink unary sequence tables (see ["Unary Table Sequencing"](#) on page 20-17). After migration, you must manually configure your project to use TopLink unary sequence tables if your application originally used single-column sequence tables in BEA WebLogic.

After configuring the sequence type at the session (or project) level, to enable sequencing, you must configure a descriptor with a sequence field and a sequence name (see ["Configuring Sequencing at the Descriptor Level"](#) on page 29-3).

For more information about sequencing, see ["Understanding Sequencing in Relational Projects"](#) on page 20-14.

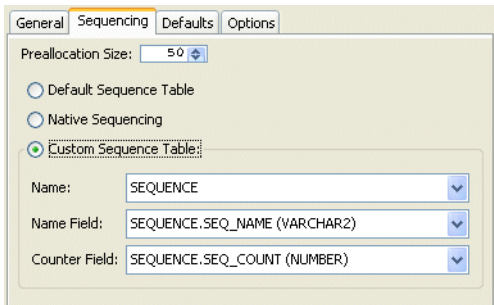
Using TopLink Workbench

To specify the sequencing information for a relational server (or database) session, use this procedure:

1. Select the session object in the **Navigator**.

- 2. Click the **Login** tab in the **Editor**.
- 3. Click the **Sequencing** subtab. The Sequencing subtab appears.

Figure 86–4 Login Tab, Sequencing Subtab



Use the following information to enter data in each field of the Sequencing subtab to configure the persistence type:

Field	Description
Preallocation Size	Select the default preallocation size (see "Sequencing and Preallocation Size" on page 20-20). Default is 50. The preallocation size you configure applies to all sequences.
Default Sequence Table	Select this option to use table sequencing (see "Table Sequencing" on page 20-16) with default sequence table name SEQUENCE, default sequence name field SEQ_NAME, and default sequence count field SEQ_COUNT.
Native Sequencing	Select this option to use a sequencing object (see "Native Sequencing With an Oracle Database Platform" on page 20-18 or "Native Sequencing With a Non-Oracle Database Platform" on page 20-19) created by the database platform. This option applies only to Oracle, Sybase, Microsoft SQL, IBM Informix and IBM DB2 database platforms.
Custom Sequence Table	Select this option to use table sequencing (see "Table Sequencing" on page 20-16) with a sequence table name, sequence name field, and sequence count field name that you specify.
Name	Select the name of the sequence table.
Name Field	Select the name of the column used to store the sequence name.
Counter Field	Select the name of the column used to store the sequence count.

Using Java

Using Java, you can perform the following sequence configurations:

- [Using the Platform Default Sequence](#)
- [Configuring Multiple Sequences](#)
- [Configuring Query Sequencing](#)

Using the Platform Default Sequence

After you configure your login with a platform (see ["Configuring a Relational Database Platform at the Session Level"](#) on page 86-1), you can use the default sequence that the platform provides.

If you associate a descriptor with an unspecified sequence, the TopLink runtime will create an instance of `DefaultSequence` to provide sequencing for that descriptor. For more information, see ["Configuring the Platform Default Sequence"](#) on page 29-6.

You can access the default platform sequence directly as [Example 86-1](#) shows. For example, by default, a `DatabasePlatform` creates a table sequence using the default table and column names (see ["Table Sequencing"](#) on page 20-16).

Example 86-1 Accessing the Platform Default Sequence

```
// assume that dbLogin owns a DatabasePlatform
TableSequence tableSeq2 = ((TableSequence)dbLogin.getDefaultSequence()).clone();
tableSeq2.setName("EMP_SEQ");
tableSeq2.setPreallocationSize(75);
dbLogin.addSequence(tableSeq2);
```

To avoid having to clone the platform default sequence, you can use the `DefaultSequence` class—a wrapper for the platform default sequence—as [Example 86-2](#) shows. The new sequence named `EMP_SEQ` will be of the same type as the platform default sequence.

Example 86-2 Using the DefaultSequence Class

```
login.addSequence(
    new DefaultSequence("EMP_SEQ", 75)
);
```

You can override the default platform sequence as [Example 86-3](#) shows. In this example, `dbLogin` owns a `DatabasePlatform` that provides a default sequence of type `TableSequence`. After setting the default sequence to type `UnaryTableSequence`, when you use the `DefaultSequence` class, it will access the new default sequence type. In this example, the sequence named `EMP_SEQ` will be of type `UnaryTableSequence` and have a preallocation size of 75.

Example 86-3 Overriding the Platform Default Sequence

```
// assume that dbLogin owns a DatabasePlatform
Sequence unaryTableSequence = new UnaryTableSequence();
unaryTableSequence.setPreallocationSize(40);
dbLogin.setDefaultSequence(unaryTableSequence);
dbLogin.addSequence(
    new DefaultSequence("EMP_SEQ", 75) // UnaryTableSequence
);
```

Configuring Multiple Sequences

In addition to using the platform default sequence (see ["Using the Platform Default Sequence"](#) on page 86-5), you can explicitly create sequence instances and configure a `Login` with any combination of sequence types, each with their own preallocation size as [Example 86-4](#) shows. In this example, the sequence named `EMP_SEQ` will provide sequence values exclusively for instances of the `Employee` class and `ADD_SEQ` will provide sequence values exclusively for instances of the `Address` class. The sequence named `PHONE_SEQ` will use the platform default sequence with a preallocation size of 30 to provide sequence values for the `Phone` class.

Example 86-4 Configuring Multiple Sequences Explicitly

```
login.addSequence(new TableSequence("EMP_SEQ", 25));
login.addSequence(new DefaultSequence("PHONE_SEQ", 30));
```

```
login.addSequence(new UnaryTableSequence("ADD_SEQ", 55));
login.addSequence(new NativeSequence("NAT_SEQ", 10));
```

If login owned a DatabasePlatform (whose default sequence type is TableSequence), you could configure your sequences using the platform default sequence type as [Example 86–5](#) shows. In this example, sequences EMP_SEQ and PHONE_SEQ share the same TableSequence table: EMP_SEQ and PHONE_SEQ represent rows in this table.

Example 86–5 Configuring Multiple Sequences Using the Default Sequence Type

```
login.addSequence(new DefaultSequence("EMP_SEQ", 25));
login.addSequence(new DefaultSequence("PHONE_SEQ", 30));
login.addSequence(new UnaryTableSequence("ADD_SEQ", 55));
login.addSequence(new NativeSequence("NAT_SEQ", 10));
```

Configuring Query Sequencing

You can configure the query that TopLink uses to update or read a sequence value for any sequence type that extends QuerySequence.

In most applications, the queries that TopLink automatically uses are sufficient. However, if your application has special sequencing needs—for example, if you want to use stored procedures for sequencing—then you can configure the update and read queries that the TopLink sequence uses.

[Example 86–7](#) illustrates how to specify a stored procedure that updates a sequence and returns the new sequence value with a single SQL select query. In this example, the stored procedure is named UPDATE_SEQ. It contains one input argument—the name of the sequence to update (SEQ_NAME), and one output argument—the value of the sequence after the update (SEQ_COUNT). The stored procedure increments the sequence value associated with the sequence named SEQ_NAME and returns the new sequence value in the output argument named SEQ_COUNT.

Example 86–6 Using a Stored Procedure for both Sequence Update and Select

```
ValueReadQuery seqReadQuery = new ValueReadQuery();
StoredProcedureCall spCall = new StoredProcedureCall();
spCall.setProcedureName("UPDATE_SEQ");
seqReadQuery.addNamedArgument("SEQ_NAME");
seqReadQuery.addNamedOutputArgument("SEQ_COUNT");
seqReadQuery.setCall(spCall);
((QuerySequence)login.getDefaultSequence()).setSelectQuery(seqReadQuery);
```

[Example 86–7](#) and [Example 86–8](#) illustrate how to specify separate stored procedures for sequence update and select actions.

In [Example 86–7](#), the stored procedure is named UPDATE_SEQ and it contains one input argument: the name of the sequence to update (SEQ_NAME). The stored procedure increments the sequence value associated with the sequence named SEQ_NAME.

Example 86–7 Using a Stored Procedure for Sequence Updates Only

```
DataModifyQuery seqUpdateQuery = new DataModifyQuery();
StoredProcedureCall spCall = new StoredProcedureCall();
spCall.setProcedureName("UPDATE_SEQ");
seqUpdateQuery.addNamedArgument("SEQ_NAME");
seqUpdateQuery.setCall(spCall);
```



```
((QuerySequence)login.getDefaultSequence()).setUpdateQuery(seqUpdateQuery);
```

In [Example 86–8](#), the stored procedure is named `SELECT_SEQ` and it takes one argument: the name of the sequence to select from (`SEQ_NAME`). The stored procedure reads one data value: the current sequence value associated with the sequence name `SEQ_NAME`.

Example 86–8 Using a Stored Procedure for Sequence Selects Only

```
ValueReadQuery seqReadQuery = new ValueReadQuery();
StoredProcedureCall spCall = new StoredProcedureCall();
spCall.setProcedureName("SELECT_SEQ");
seqReadQuery.addArgument("SEQ_NAME");
seqReadQuery.setCall(spCall);
login.((QuerySequence)getDefaultSequence()).setSelectQuery(seqReadQuery)
```

You can also create a `QuerySequence` directly and add it to your login, as [Example 86–9](#) shows.

Example 86–9 Using the QuerySequence Class

```
// Use the two-argument constructor: pass in sequence name and preallocation size.
// Alternatively, you can use zero- or one-argument (sequence name) constructor
login.addSequence(new QuerySequence("SEQ1", 75));
```

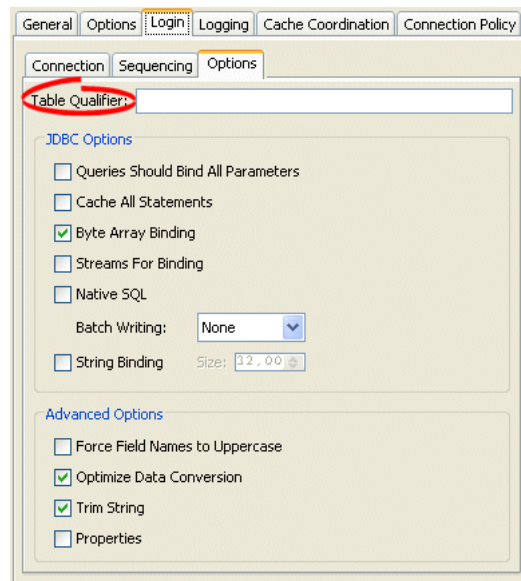
Configuring a Table Qualifier

Some databases (such as Oracle Database and DB2) require that all tables be qualified by an identifier. This can be the creator of the table or database name on which the table exists. When you specify a table qualifier, TopLink uses this qualifier for all of the tables it references. Specify a table qualifier only if required and only if all of the tables have the same qualifier.

Using TopLink Workbench

To specify a table qualifier, use this procedure:

1. Select a relational server (or database) session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Login** tab. The Login tab appears.
3. Click the **Options** subtab. The Options subtab appears.

Figure 86-5 Login Tab, Options Subtab, Table Qualifier Field

In the **Table Qualifier** field enter the identifier used to qualify references to all tables in this database.

Configuring JDBC Options

Most JDBC drivers support the run-time configuration of various options to customize driver operation to meet user needs. TopLink provides direct support (in API and TopLink Workbench) for many of the most important options, as this section describes, as well as more advanced options (see ["Configuring Advanced Options"](#) on page 86-11)

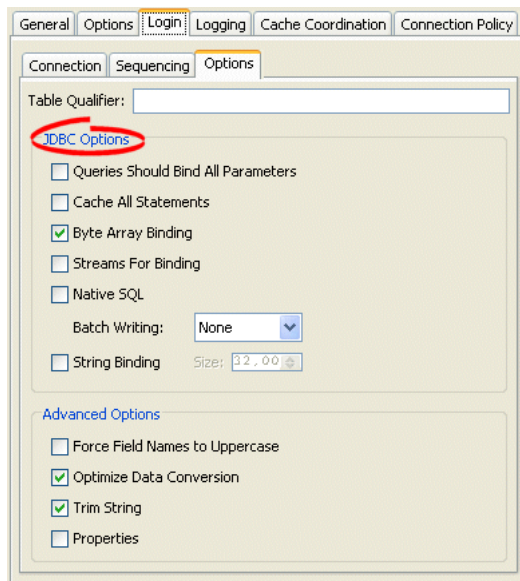
You can also configure additional options by specifying properties (see ["Configuring Properties"](#) on page 85-4).

Note: Not all drivers support all JDBC options. Selecting a combination of options may result in different behavior from one driver to another. Before selecting JDBC options, consult your JDBC driver documentation.

Using TopLink Workbench

To specify the JDBC options for a relational server (or database) session login, use this procedure:

1. Select a relational server (or database) session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Login** tab. The Login tab appears.
3. Click the **Options** subtab. The Options subtab appears.

Figure 86–6 Login Tab, Options Subtab, JDBC Options

Option	Description
Queries Should Bind All Parameters¹	Select this option to bind all of the query's parameters
Cache All Statements¹	When selected, TopLink caches each prepared statement so that when reexecuted, you avoid the SQL preparation time which improves performance.
Byte Array Binding¹	Select this option if you query binary large object (BLOB) data.
Streams for Binding¹	Select this option if you use a JDBC driver that is more efficient at handling BLOB data using <code>java.io.InputStream</code> and <code>java.io.OutputStream</code> .
Native SQL	By default, TopLink generates SQL using JDBC SQL grammar. Select this option if you want TopLink to use database specific SQL grammar, for example, if your database driver does not support the full JDBC SQL grammar.
Batch Writing²	Select this option if you use a JDBC driver that supports sending groups of <code>INSERT</code>, <code>UPDATE</code>, and <code>DELETE</code> statements to the database in a single transaction, rather than individually. Select JDBC to use the batch writing capabilities of your JDBC driver. Select TopLink to use the native batch writing capabilities that TopLink provides. Select this option if your JDBC driver does not support batch writing.
String Binding¹	Select this option if you query large <code>java.lang.String</code> objects. You can configure the maximum <code>String</code> length (default: 32000 characters).

¹ For more information, see "[Parameterized SQL \(Binding\) and Prepared Statement Caching](#)" on page 11-15.

² If you are using the MySQL4 database platform (see "[Data Source Platform Types](#)" on page 84-3), use **JDBC** batch writing (do not use **TopLink** batch writing). For more information, see "[Batch Writing](#)" on page 11-15.

Using Java

To enable parameterized SQL and prepared statement caching for all queries, configure at the **Login** level, as [Example 86–10](#) shows. For more information, see ["Parameterized SQL \(Binding\) and Prepared Statement Caching"](#) on page 11-15.

Example 86–10 *Parameterized SQL and Binding at the Login Level*

```
databaseLogin.bindAllParameters();  
databaseLogin.cacheAllStatements();  
databaseLogin.setStatementCacheSize(100);
```

To enable JDBC batch writing, use **Login** method `useBatchWriting` as [Example 86–11](#) shows:

Example 86–11 *Using JDBC Batch Writing*

```
project.getLogin().useBatchWriting();  
project.getLogin().bindAllParameters();  
project.getLogin().setMaxBatchWritingSize(100);
```

Configuring Advanced Options

Most JDBC drivers support the run-time configuration of various options to customize driver operation to meet user needs. TopLink provides direct support (in API and TopLink Workbench) for many of the most important options (see ["Configuring JDBC Options"](#) on page 86-9), as well as more advanced options, as this section describes.

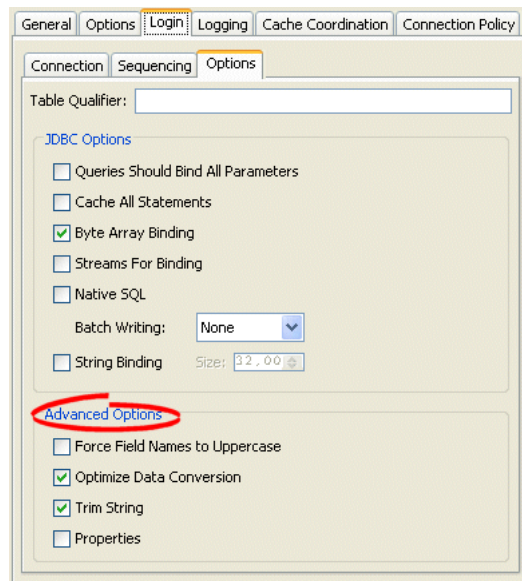
You can also configure additional options by specifying properties (see ["Configuring Properties"](#) on page 85-4).

Note: Not all drivers support all JDBC options. Selecting a combination of options may result in different behavior from one driver to another. Before selecting JDBC options, consult your JDBC driver documentation.

Using TopLink Workbench

To specify the advanced options for a relational server (or database) session login, use this procedure:

1. Select a relational server (or database) session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Login** tab. The Login tab appears.
3. Click the **Options** subtab. The Options subtab appears.

Figure 86–7 Login Tab, Options Subtab, Advanced Options

Option	Description
Force Field Names to Uppercase	By default, TopLink uses the case of field names as returned by the database. If your application expects field names to be uppercase but the database does not return consistent case (for example, if you accessing different databases), enable this option.
Optimize Data Conversion	By default, TopLink optimizes data access by accessing the data from JDBC in the format the application requires. If you are using an older JDBC driver that does not perform data conversion correctly and conflicts with this optimization, disable this optimization.
Trim String	By default, TopLink discards the trailing blanks from CHAR field values. To read and write CHAR field values literally (including any trailing blanks), disable this option.
Properties	Check this option to enable the use of properties for this DatabaseLogin (see "Configuring Properties" on page 85-4).

Configuring Oracle Database Proxy Authentication

You can configure a database login to use Oracle Database proxy authentication with an Oracle Database platform in JSE applications and JEE applications using OC4J native or managed data sources with Oracle JDBC driver release 10.1.0.2.0 or later and external connection pools only.

There is no TopLink Workbench support for this feature. To configure TopLink to use Oracle Database proxy authentication, you must use Java (see ["Using Java"](#) on page 86-13).

For more information, see ["Oracle Database Proxy Authentication"](#) on page 84-5.

You can use TopLink support for Oracle Database proxy authentication by doing the following:

- [Providing Authenticated Reads and Writes of Secured Data Through the Use of an Exclusive Isolated Client Session](#)

- [Providing Authenticated Writes for Database Auditing Purposes With a Client Session](#)
- [Providing Authenticated Reads and Writes With a Database Session](#)

Providing Authenticated Reads and Writes of Secured Data Through the Use of an Exclusive Isolated Client Session

In this configuration, the client Session is an isolated client session (see ["Isolated Client Sessions"](#) on page 75-19) that uses an exclusive proxy connection. You must acquire the client session using a `ConnectionPolicy` that specifies the proxy authentication user credentials.

Reads and writes of secured data are performed through the proxy-authenticated connection. Reads of nonsecured data occur through nonproxy-authenticated connections.

If you are using Oracle Private Virtual Database (VPD) (see ["Isolated Client Sessions and Oracle Virtual Private Database \(VPD\)"](#) on page 75-20), use this configuration to set up VPD support entirely in the database. That is, rather than making the isolated client session execute SQL (see ["PostAcquireExclusiveConnection Event Handler"](#) on page 80-1 and ["PreReleaseExclusiveConnection Event Handler"](#) on page 80-2), the database performs the required set up in an after login trigger using the proxy `session_user`.

Providing Authenticated Writes for Database Auditing Purposes With a Client Session

In this configuration, isolated data or exclusive connections are not required. You must acquire client session using a `ConnectionPolicy` that specifies the proxy authentication user credentials.

Writes are performed through the proxy-authenticated connection. Reads occur through nonproxy-authenticated connections. This enables the database auditing process to access the user that performed the write operations.

Providing Authenticated Reads and Writes With a Database Session

In this configuration, you use a `DatabaseSession` object with a proxy-authenticated login. All reads and writes occur through the proxy-authenticated connection.

Note: Oracle recommends that you use server and client sessions in a three-tier environment. If using database sessions, ensure that each database session is used by a single user and not accessed concurrently.

Using Java

You configure Oracle Database proxy authentication by customizing your session in your Java code, such as through a `SessionCustomizer` when using the `sessions.xml` file. You can wrap a configured `TopLink DataSourceLogin JNDIConnector` with a `TopLink proxy connector` instance (from `oracle.toplink.platform.database.oracle`) appropriate for your JDBC driver and to configure proxy authentication properties.

If you are using the Oracle JDBC OCI driver, use the `OracleOCIProxyConnector` and property constants defined in `oracle.jdbc.pool.OracleOCIConnectionPool`.

If you are using the Oracle JDBC Thin driver, use the `OracleJDBC10_1_0_2ProxyConnector` and the property constants defined in `oracle.jdbc.OracleConnection`.

The properties to set are shown in [Table 86–2](#) through [Table 86–5](#).

Note: Property constant names and values are consistent between the two classes except for `PROXYTYPE_` constants (such as `PROXYTYPE_USER_NAME`). In `OracleOCIConnectionPool` these are of type `String` and in `OracleConnection` they are of type `int`. If you are using the Oracle JDBC Thin driver and `OracleJDBC10_1_0_2ProxyConnector`, you must always set these properties as a `String`. For example:

```
login.setProperty(
    "proxytype", Integer.toString(OracleConnection.PROXYTYPE_USER_NAME));
```

To configure TopLink to use Oracle Database proxy authentication, do the following:

1. Decide on the proxy type you want to use and create appropriate users and roles.

- a. User Name Authentication:

To authenticate a proxy user `sarah` by user name only, create the user account on the Oracle Database using the following:

```
alter user sarah grant connect through dbadminuser
    with roles clerk, reports;
```

In this case, you will need to set the proxy properties shown in [Table 86–2](#).

Table 86–2 Proxy Properties for User Name Authentication

Property Name	Property Value
"proxytype"	PROXYTYPE_USER_NAME
PROXY_USER_NAME	"sarah"
PROXY_ROLES	String[] {"role1", "role2", ...}

- b. User Name and Password Authentication:

To authenticate a proxy user `sarah` by user name and password, create the user account on the Oracle Database using the following:

```
alter user sarah grant connect through dbadminuser
    authenticated using password
    with roles clerk, reports;
```

In this case, you will need to set the proxy properties shown in [Table 86–3](#).

Table 86–3 Proxy Properties for User Name and Password Authentication

Property Name	Property Value
"proxytype"	PROXYTYPE_USER_NAME
PROXY_USER_NAME	"sarah"
PROXY_PASSWORD	"passwordforsarah"
PROXY_ROLES	String[] {"role1", "role2", ...}

c. Distinguished Name Authentication:

To authenticate a proxy user sarah by globally unique distinguished name, create the user account on the Oracle Database using the following:

```
create user sarah identified globally as
  'CN=sarah,OU=americas,O=oracle,L=city,ST=ca,C=us';
alter user sarah grant connect through dbadminuser
  authenticated using distinguished name
  with roles clerk, reports;
```

In this case, you will need to set the proxy properties shown in [Table 86–4](#).

Table 86–4 Proxy Properties for Distinguished Name Authentication

Property Name	Property Value
"proxytype"	PROXYTYPE_DISTINGUISHED_NAME
PROXY_DISTINGUISHED_NAME	"CN=sarah,OU=americas,O=oracle,L=city,ST=ca,C=us"
PROXY_ROLES	String[] {"role1", "role2", ...}

d. Certificate Authentication:

To authenticate a proxy user sarah by encrypted distinguished name, create the user account on the Oracle Database using the following:

```
alter user sarah grant connect through dbadminuser
  authenticated using certificate
  with roles clerk, reports;
```

In this case, you will need to set the proxy properties shown in [Table 86–2](#).

Table 86–5 Proxy Properties for User Name Authentication

Property Name	Property Value
"proxytype"	PROXYTYPE_CERTIFICATE
PROXY_CERTIFICATE	byte[] {<EncryptedCertificate>}
PROXY_ROLES	String[] {"role1", "role2", ...}

2. Configure your session login using Java code. Do this through a SessionCustomizer when using the sessions.xml file.

The following example demonstrates how you can wrap the already specified JNDIConnector with the appropriate TopLink proxy authentication connector. You can set the server session's default connection policy to the same proxy authenticated login.

If you use Oracle VPD (ref VPD), you should set the connection policy to use exclusive connections, and the descriptor for secured data to isolated (ref isolated).

```
login login = server.getDatasourceLogin();
// Make sure that external connection pooling is used
login.setUsesExternalConnectionPooling(true);
// Wrap JNDIConnector with either
// OracleOCIProxyConnector or OracleJDBC10_1_0_2ProxyConnector
login.setConnector(
  new OracleOCIProxyConnector(
    ((JNDIConnector)login.getConnector()).getName());
ConnectionPolicy policy = server.getDefaultConnectionPolicy();
```

```
policy.setPoolName(null);
policy.setLogin(login);
// If using Oracle VPD support, set the connection policy to exclusive
policy.setShouldUseExclusiveConnection(true);
```

3. Acquire a proxy-authenticated client session through specifying a **ConnectionPolicy** with this user's credentials.

```
ConnectionPolicy policy =
    (ConnectionPolicy)server.getDefaultConnectionPolicy().clone();
Login login = (Login)policy.getLogin().clone();
// Set proxy properties into connection policy's login
login.setProperty("proxytype" , OracleOCIConnectionPool.PROXYTYPE_USER_NAME);
login.setProperty(OracleOCIConnectionPool.PROXY_USER_NAME , "sarah");
policy.setLogin(login);
Session session = server.acquireClientSession(policy);
```

Configuring an EIS Login

This chapter describes the various components that you must configure to use an EIS login.

EIS Login Configuration Overview

Table 87-1 lists the configurable options for an EIS login.

Table 87-1 Configurable Options for EIS Login

Option	Type	TopLink Workbench	Java
"Configuring an EIS Data Source Platform at the Session Level" on page 87-1	Basic	✓	✓
"Configuring EIS Connection Specification Options at the Session Level" on page 87-2	Basic	✓	✓
"Configuring User Name and Password" on page 85-1	Basic	✓	✓
"Configuring Password Encryption" on page 85-2	Advanced	✓	✓
"Configuring External Connection Pooling" on page 85-2	Advanced	✓	✓
"Configuring Properties" on page 85-4	Advanced	✓	✓

Configuring an EIS Data Source Platform at the Session Level

For each EIS session, you must specify the platform (such as AQ). This platform configuration overrides the platform at the project level, if configured.

For more information, see the following:

- ["Configuring Relational Database Platform at the Project Level" on page 23-2](#)
- ["Data Source Login Types" on page 84-2](#)

Using TopLink Workbench

To specify the database platform options for an EIS session login, use this procedure:

1. Select an EIS session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Login** tab. The Login tab appears.
3. Click the **Connection** subtab. The Connection subtab appears.

Figure 87-1 Login Tab, Connection Subtab, Platform Options

Use the following information to enter data in the Platform field on the Connection tab to configure the platform:

Field	Description
Platform	The EIS platform for the session. Select from the menu of options. This menu includes all instances of <code>EISPlatform</code> in the TopLink classpath.

Configuring EIS Connection Specification Options at the Session Level

You can configure connection information at the session level for an EIS application. This information is stored in the `sessions.xml` file. The Oracle TopLink runtime uses this information whenever you perform a persistence operation using the session in your EIS application.

This connection configuration overrides the connection information at the project level, if configured. For more information about project-level configuration, see ["Configuring Development and Deployment Logins"](#) on page 23-6 and ["Configuring EIS Connection Specification Options at the Project Level"](#) on page 24-2.

This connection configuration is overridden by the connection information at the connection pool level. For more information about connection pool-level configuration, see ["Configuring Connection Pool Connection Options"](#) on page 89-4.

Using TopLink Workbench

Use this procedure to specify the connection options for an EIS session login.

1. Select an EIS session in the Navigator window. Its properties appear in the Editor window.
2. Click the **Login** tab. The Login tab appears.
3. Click the **Connection** subtab. The Connection tab appears.

Figure 87-2 Login Tab, Connection Subtab

The screenshot shows a configuration window with tabs: General, Options, Login, Logging, Cache Coordination, and Connection Policy. The 'Login' tab is active, and the 'Connection' subtab is selected. The 'Platform' dropdown is set to 'Oracle AQ'. The 'Connection Specification Class' field contains 'oracle.toplink.eis.adapters.aq.AQEISConnectionSpec'. The 'Connection Factory URL' field is empty. The 'User Name (Optional)' and 'Password (Optional)' fields are also empty. The 'External Connection Pooling' checkbox is unchecked. Red circles highlight the 'Connection Specification Class' and 'Connection Factory URL' fields.

Use the following information to enter data in the driver fields on the tab:

Field	Description
Connection Specification Class	Specify the appropriate connection specification class for the selected Platform . Click Browse to choose from all the classes in the TopLink classpath. (For example: if Platform is <code>oracle.toplink.eis.aq.AQPlatform</code> , use <code>oracle.toplink.eis.aq.AQEISConnectionSpec</code>). For more information on platform configuration, see "Configuring an EIS Data Source Platform at the Session Level" on page 87-1.
Connection Factory URL	Specify the appropriate connection factory URL for the selected Connection Specification Class (For example: <code>jdbc:oracle:thin@:localhost:1521:orcl</code>).

Creating an Internal Connection Pool

This chapter explains how to create TopLink internal connection pools, including the following:

- [Internal Connection Pool Creation Overview](#)

For information, see "[Internal Connection Pools](#)" on page 84-7.

Internal Connection Pool Creation Overview

You can create internal connection pools only for server sessions (not for any other session type, including database sessions).

You can create an internal connection pool using TopLink Workbench or Java code. Oracle recommends that you use TopLink Workbench to create and manage your internal connection pools. For more information, see "[Using TopLink Workbench](#)" on page 88-1.

Alternatively, you can create internal connection pools in Java. For more information on creating sessions in Java, see the Oracle TopLink API Reference.

After you create an internal connection pool, you must configure its various options (see "[Internal Connection Pool Configuration Overview](#)" on page 89-1).

After you create and configure a sequence connection pool, TopLink uses it whenever it needs to assign an identifier to a new object.

After you create and configure a named connection pool, you use it in your application by passing in a `ConnectionPolicy` when you acquire a client session (see "[Acquiring a Client Session that Uses a Named Connection Pool](#)" on page 78-8).

Using TopLink Workbench

Before you create a connection pool, you must first create a server session (see "[Creating a Server Session](#)" on page 76-4).

To create a new TopLink internal connection pool, use this procedure:

1. Select the server session in the **Navigators** in which you want to create a connection pool.
2. Click the appropriate button on the toolbar to create the type of connection pool you want:
 - To create a named connection pool, select **Create a New Named Connection Pool**, enter a name, and click **OK**.



- To create a sequence connection pool, select **Add the Sequence Connection Pool**.
- To create a write connection pool, select **Add the Write Connection Pool**.

You can also create a new internal connection pool by right-clicking the server session configuration in the **Navigator** and selecting **New > Named Connection Pool, Sequence Connection Pool, or Write Connection Pool** from the context menu.

Configuring an Internal Connection Pool

This chapter describes the various components that you must configure to use an internal connection pool.

Internal Connection Pool Configuration Overview

When you are using server sessions, you can configure the default read connection pool and write connection pool. You can also configure the optional named connection pools and sequence connection pool you may have created (see ["Internal Connection Pool Creation Overview"](#) on page 88-1).

[Table 89-1](#) lists the configurable options for an internal connection pool.

Table 89-1 Configurable Options for Connection Pool

Option	Type	TopLink Workbench	Java
"Configuring Connection Pool Sizes" on page 89-1	Basic	✓	✓
"Configuring Exclusive Read Connections" on page 89-6 ¹	Advanced	✓	✓
"Configuring a Nontransactional Read Login" on page 89-3 ¹	Advanced	✓	✓
"Configuring Properties" on page 89-2	Advanced	✓	✓
"Configuring Connection Pool Connection Options" on page 89-4 ²	Advanced	✓	✓

¹ Read connection pools only.

² Not applicable to write connection pools.

Configuring Connection Pool Sizes

By default, if using TopLink internal connection pooling, the TopLink write connection pool maintains a minimum of five connections and a maximum of ten. The read connection pool maintains a minimum and maximum of two connections.

Connection pool size can significantly influence the concurrency of your application and should be set to be large enough to handle your expected application load.

Tip: To maintain compatibility with JDBC drivers that do not support many connections, the default number of connections is small. If your JDBC driver supports it, use a larger number of connections for reading and writing.

The smallest value you can enter is 0. Setting the maximum number of connections to 0 will make it impossible for TopLink to allocate any connections.

The minimum number of connections should always be less than or equal to the maximum number of connections.

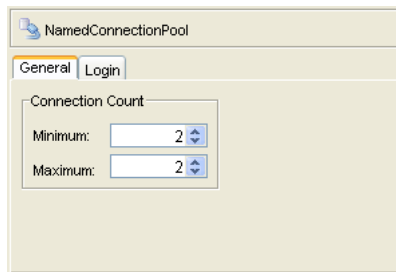
If the maximum number of connections is in use, the next connection request will be blocked until a connection is available.

Using TopLink Workbench

To specify the minimum and maximum number of connections in a TopLink internal connection pool, use this procedure:

1. Expand a server session to reveal its connection pools in the **Navigator**.
2. Select a connection pool in the **Navigator**. Its properties appear in the Editor.
3. Click the **General** tab. The General tab appears.

Figure 89–1 General Tab, Connection Count Options



Enter the desired minimum and maximum number of connections and press **Enter** or use the increment and decrement arrows.

Configuring Properties

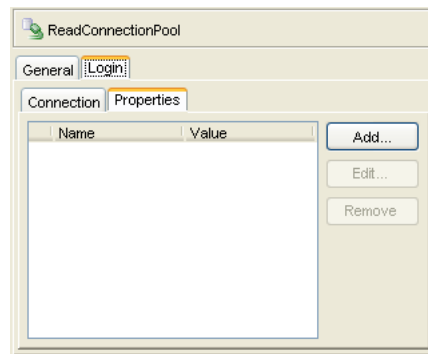
For all connection pools, except write connection pools, you can specify arbitrary named values, called properties.

Some data sources require additional, driver-specific properties not supported in the `ConnectionPool` API. Add these properties to the `ConnectionPool` so that TopLink can pass them to the driver.

Using TopLink Workbench

To specify arbitrary named value pairs that TopLink associates with a `ConnectionPool`, use this procedure:

1. Expand a server session to reveal its connection pools in the **Navigator**.
2. Select a read, named, or sequence connection pool in the **Navigator**. Its properties appear in the Editor.
3. Click the **Login** tab. The Login tab appears.
4. Click the **Properties** subtab. The Properties subtab appears.

Figure 89-2 Login Tab, Properties Subtab

Use the following information to add or edit a login property on the Add Property dialog box to add or edit a login property:

Option	Description
Name	The name by which TopLink retrieves the property value using the <code>DatasourceLogin</code> method <code>getProperty</code> .
Value	The value TopLink retrieves using the <code>DatasourceLogin</code> method <code>getProperty</code> passing in the corresponding property name. Using TopLink Workbench, you can set only character values which TopLink returns as <code>String</code> objects.

To add (or change) a new **Name/Value** property, click **Add** (or **Edit**).

To delete an existing property, select the **Name/Value** row and click **Remove**.

Using Java

Using Java, you can set any `Object` value using the `DatasourceLogin` method `setProperty`. To remove a property, use the `DatasourceLogin` method `removeProperty`.

Configuring a Nontransactional Read Login

When you use an external transaction controller (see "[Configuring the Server Platform](#)" on page 77-14), establishing a connection requires not only the usual connection setup overhead, but also transactional overhead. If your application reads data only to display it and only infrequently modifies data, you can configure an internal read connection pool to use its own connection specification that does not use the external transaction controller. This may improve performance by reducing the time it takes to establish a new read connection.

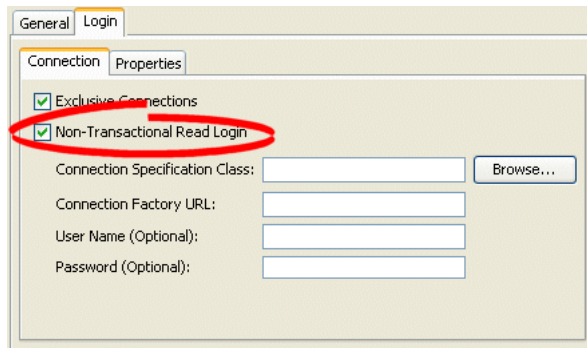
Using TopLink Workbench

To enable the configuration of nontransactional connection information for a TopLink read connection pool, use this procedure:

1. Expand a server session to reveal its connection pools in the **Navigator**.
2. Select a read connection pool in the **Navigator**. Its properties appear in the Editor.
3. Click the **Login** tab. The Login tab appears.

- Click the **Connection** subtab. The Connection subtab appears.

Figure 89-3 Login Tab, Connection Subtab



To enable a nontransactional read login, select the **Use Non-Transactional Read Login** option (see ["Externally Managed Transactional Data Sources"](#) on page 84-1). Continue with ["Configuring Connection Pool Connection Options"](#) on page 89-4 to specify the connection information.

Configuring Connection Pool Connection Options

By default, connection pools use the login configuration specified for their session (see ["Configuring Database Login Connection Options"](#) on page 86-2 and ["Configuring EIS Connection Specification Options at the Session Level"](#) on page 87-2).

For read, named, and sequence connection pools, you can override the session login configuration on a per-connection pool basis.

To configure login configuration for a read connection pool, you must first enable it for a nontransactional read login (see ["Configuring a Nontransactional Read Login"](#) on page 89-3).

Using TopLink Workbench

To configure connection information for a TopLink read, named, or sequence connection pool, use this procedure:

- Expand a server session to reveal its connection pools in the **Navigator**.
- Select a read, named, or sequence connection pool in the **Navigator**. Its properties appear in the Editor.
- Click the **Login** tab. The Login tab appears.
- Click the **Connection** subtab. The Connection subtab appears.

Figure 89–4 Login Tab, Connection Subtab, Relational Session Connection Pool Options

General Login

Connection Properties

☐ Exclusive Connections

☒ Non-Transactional Read Login

Database Driver: Driver Manager

Driver Class: Browse...

URL:

User Name (Optional):

Password (Optional):

Figure 89–5 Login Tab, Connection Subtab, EIS Session Connection Pool Options

General Login

Connection Properties

☒ Exclusive Connections

☒ Non-Transactional Read Login

Connection Specification Class: Browse...

Connection Factory URL:

User Name (Optional):

Password (Optional):

5. Ensure the **Use Non-Transaction Read Login** option is selected.

Use the following information to complete fields on the Connection subtab:

Field	Description
Database Driver ¹	Specify the appropriate database driver: ■ Driver Manager: Specify this option to configure the driver class and URL used to connect to the database. In this case, you must configure the Driver Class and Driver URL fields. ■ J2EE Datasource: Specify this option to use a J2EE data source already configured on your target application server. In this case, you must configure the Datasource Name field. Note: If you select J2EE Datasource, you must use external connection pooling. You cannot use internal connection pools with this Database Driver option (for more information, see "Configuring External Connection Pooling" on page 85-2).
Driver Class ¹	Configure this field when Database Driver is set to Driver Manager. Select from the menu of options. This menu includes all JDBC drivers in the TopLink application classpath.
URL ¹	Configure this field when Database Driver is set to Driver Manager. Select from the menu of options relevant to the selected Driver Class and edit the URL to suit your data source.

Field	Description
Datasource Name ¹	Configure this field when Database Driver is set to J2EE Datasource. Specify any valid JNDI name that identifies the J2EE data source preconfigured on your target application server (For example: jdbc/EmployeeDB). By convention, all such names should resolve to the JDBC subcontext (relative to the standard java:comp/env naming context that is the root of all provided resource factories).
Connection Specification Class ²	Specify the appropriate connection specification class for the selected Platform. Click Browse to choose from all the classes in the TopLink classpath. (For example: if Platform is <code>oracle.toplink.eis.aq.AQPlatform</code>, use <code>oracle.toplink.eis.aq.AQEISConnectionSpec</code>). For more information on platform configuration, see "Configuring an EIS Data Source Platform at the Session Level" on page 87-1.
Connection Factory URL ²	Specify the appropriate connection factory URL for the selected Connection Specification Class (For example: <code>jdbc:oracle:thin@:localhost:1521:orcl</code>).

¹ For sessions that contain a DatabaseLogin.

² For sessions that contain an EISLogin.

Configuring Exclusive Read Connections

An exclusive connection is one that TopLink allocates specifically to a given session and one that is never used by any other session.

Allowing concurrent reads on the same connection reduces the number of read connections required and reduces the risk of having to wait for an available connection. However, many JDBC drivers do not support concurrent reads.

If you are using internal connection pools (see ["Internal Connection Pools"](#) on page 84-7), you can configure TopLink to acquire an exclusive connection from the read connection pool.

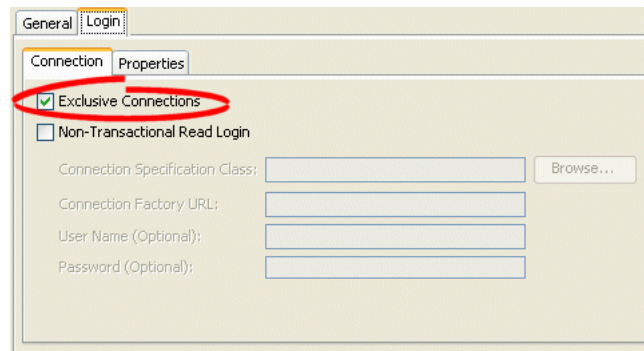
By default, TopLink acquires exclusive read connections.

If you are using external connection pools, read connections are always exclusive.

Using TopLink Workbench

To configure a TopLink read connection pool to allocate exclusive connections, use this procedure:

1. Expand a server session to reveal its connection pools in the **Navigator**.
2. Select a read connection pool in the **Navigator**. Its properties appear in the Editor.
3. Click the **Login** tab. The Login tab appears.
4. Click the **Connection** subtab. The Connection subtab appears.

Figure 89–6 Login Tab, Connection Subtab, Exclusive Connections OptionThe image shows a software configuration window with two tabs: 'General' and 'Login'. The 'Login' tab is active. Inside the 'Login' tab, there are two sub-tabs: 'Connection' and 'Properties'. The 'Connection' sub-tab is selected. Under the 'Connection' sub-tab, there are two checkboxes: 'Exclusive Connections' (which is checked and circled in red) and 'Non-Transactional Read Login' (which is unchecked). Below these checkboxes, there are four text input fields labeled 'Connection Specification Class:', 'Connection Factory URL:', 'User Name (Optional):', and 'Password (Optional):'. To the right of the 'Connection Specification Class:' field is a 'Browse...' button.

Select the **Exclusive Connections** option to configure TopLink to acquire an exclusive connection from the read connection pool.

Deselect the **Exclusive Connections** option to configure TopLink to share read connections and allow concurrent reads. Before selecting this option, ensure that your JDBC driver supports concurrent reads.

Part XVIII

Cache

This part describes using the TopLink object cache in both distributed and nondistributed applications. It contains the following chapters:

- [Chapter 90, "Understanding the Cache"](#)
This chapter describes each of the different TopLink cache types and important cache concepts.
- [Chapter 91, "Configuring a Coordinated Cache"](#)
This chapter explains how to configure TopLink coordinated cache options common to two or more coordinated cache types.
- [Chapter 92, "Configuring a JMS Coordinated Cache"](#)
This chapter explains how to configure a TopLink JMS coordinated cache.
- [Chapter 93, "Configuring an RMI Coordinated Cache"](#)
This chapter explains how to configure a TopLink RMI coordinated cache.
- [Chapter 94, "Configuring a CORBA Coordinated Cache"](#)
This chapter explains how to configure a TopLink CORBA coordinated cache.

Understanding the Cache

The TopLink cache is an in-memory repository that stores recently read or written objects based on class and primary key values. TopLink uses the cache to do the following:

- Improve performance by holding recently read or written objects and accessing them in-memory to minimize database access
- Manage locking and isolation level
- Manage object identity

This section describes the following:

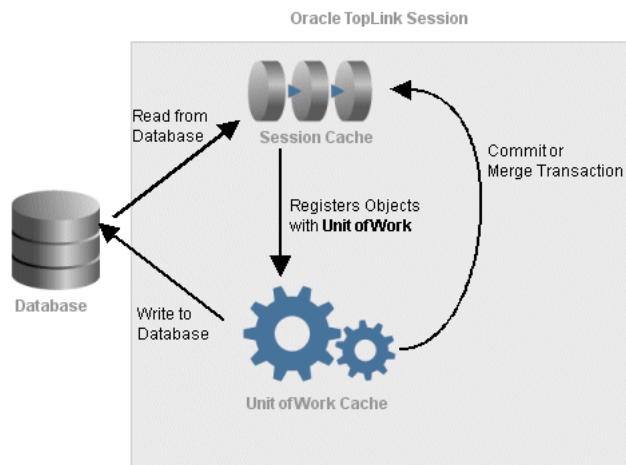
- [Cache Architecture](#)
- [Cache Concepts](#)
- [Understanding the Cache API](#)

Cache Architecture

TopLink uses two types of cache: the **session cache** maintains objects retrieved from and written to the data source; and the **unit of work cache** holds objects while they participate in transactions. When a unit of work successfully commits to the data source, TopLink updates the session cache accordingly.

Note: You can also configure a query to cache its results (see ["Caching Results in a ReadQuery"](#) on page 99-21)

As [Figure 90-1](#) shows, the session cache and the unit of work cache work together with the data source connection to manage objects in a TopLink application. The object life cycle relies on these three mechanisms.

Figure 90–1 Object Life Cycle and the TopLink Caches

Session Cache

The session cache is a shared cache that services clients attached to a given session. When you read objects from or write objects to the data source using a client session, TopLink saves a copy of the objects in the parent server session's cache and makes them accessible to all other processes in the session.

TopLink adds objects to the session cache from the following:

- The data store, when TopLink executes a read operation
- The unit of work cache, when a unit of work successfully commits a transaction

An isolated client session is a special type of client session that provides its own session cache isolated from the shared object cache of its parent server session. The isolated client session cache can be used to improve user-based security or to avoid caching highly volatile data. For more information, see ["Isolated Client Sessions"](#) on page 75-19.

Unit of Work Cache

The unit of work cache services operations within the unit of work. It maintains and isolates objects from the session cache, and writes changed or new objects to the session cache after the unit of work commits changes to the data source.

Cache Concepts

This section describes concepts unique to the TopLink cache, including the following:

- [Cache Type and Object Identity](#)
- [Querying and the Cache](#)
- [Handling Stale Data](#)
- [Explicit Query Refreshes](#)
- [Cache Invalidation](#)
- [Cache Coordination](#)
- [Cache Isolation](#)

- [Cache Locking and Transaction Isolation](#)
- [Cache Optimization](#)

Cache Type and Object Identity

TopLink preserves object identity through its cache using the primary key attributes of a persistent entity. These attributes may or may not be assigned through sequencing (see ["Projects and Sequencing"](#) on page 20-4). In a Java application, object identity is preserved if each object in memory is represented by one, and only one, object instance. Multiple retrievals of the same object return references to the same object instance—not multiple copies of the same object.

Maintaining object identity is extremely important when the application's object model contains circular references between objects. You must ensure that the two objects are referencing each other directly, rather than copies of each other. Object identity is important when multiple parts of the application may be modifying the same object simultaneously.

Oracle recommends that you always maintain object identity. Disable object identity only if absolutely necessary, for example, for read-only objects (see ["Configuring Read-Only Descriptors"](#) on page 28-4).

You can configure how object identity is managed on a class-by-class basis. The `Descriptor` object provides the cache and identity map options described in [Table 90-1](#).

Table 90-1 *Cache and Identity Map Options*

Option (Identity Map)	Caching	Guaranteed Identity	Memory Use	Client/Server Transaction Save
Full Identity Map	Yes	Yes	High	Yes
Weak Identity Map	Yes	Yes	Low	No
Soft and Hard Cache Weak Identity Maps	Yes	Yes	Lower	Yes
No Identity Map	No	No	None	No

For more information, see ["Guidelines for Configuring the Cache and Identity Maps"](#) on page 90-4.

Full Identity Map

This option provides full caching and guaranteed identity: objects are never flushed from memory unless they are deleted.

It caches all objects and does not remove them. Cache size doubles whenever the maximum size is reached. This method may be memory-intensive when many objects are read. Do not use this option on batch operations.

Oracle recommends using this identity map when the data set size is small and memory is in large supply.

Weak Identity Map

This option is similar to the full identity map, except that the map holds the objects by using weak references. This method allows full garbage collection and provides full caching and guaranteed identity.

The weak identity map uses less memory than full identity map but also does not provide a durable caching strategy across client/server transactions. Objects are available for garbage collection when the application no longer references them on the server side (that is, from within the server JVM).

Oracle recommends using this identity map for transactions that, once started, stay on the server side. Do not use this option for applications that expect objects to remain cached across client/server invocations.

Soft and Hard Cache Weak Identity Maps

This option is similar to the weak identity map except that it maintains a most frequently used subcache. The subcache uses soft or hard references to ensure that these objects are garbage-collected only if the system is low on memory.

The soft cache weak identity map and hard cache weak identity map provide more efficient memory use. They release objects as they are garbage-collected, except for a fixed number of most recently used objects. Note that weakly cached objects might be flushed if the transaction spans multiple client/server invocations. The size of the subcache is proportional to the size of the identity map as specified by the `Descriptor` method `setIdentityMapSize`. You should set this cache size to be as large as the maximum number of objects (of the same type) referenced within a transaction (see ["Configuring Cache Type and Size at the Descriptor Level"](#) on page 28-35).

Oracle recommends using this identity map in most circumstances as a means to control memory used by the cache.

For more information, see ["Understanding the Internals of Soft and Hard Cache Weak Identity Map"](#) on page 90-5.

No Identity Map

This option does not preserve object identity and does not cache objects.

Oracle does not recommend using the no identity map option.

Guidelines for Configuring the Cache and Identity Maps

You can configure the cache at the project (["Configuring Cache Type and Size at the Project Level"](#) on page 22-13) or descriptor (["Configuring Cache Type and Size at the Descriptor Level"](#) on page 28-35) level.

Use the following guidelines when configuring your cache and identity map:

- If objects with a long life span and object identity are important, use a `SoftCacheWeakIdentityMap` or `HardCacheWeakIdentityMap` policy. For more information on when to choose one or the other, see ["Understanding the Internals of Soft and Hard Cache Weak Identity Map"](#) on page 90-5.
- If object identity is important, but caching is not, use a `WeakIdentityMap` policy.
- If an object has a long life span or requires frequent access, or object identity is important, use a `FullIdentityMap` policy.

WARNING: Use the `FullIdentityMap` only if the class has a small number of finite instances. Otherwise, a memory leak will occur.

- If an object has a short life span or requires frequent access, and identity is not important, use a `CacheIdentityMap` policy.
- If objects are discarded immediately after being read from the database, such as in a batch operation, use a `NoIdentityMap` policy. The `NoIdentityMap` does not preserve object identity.

Note: Oracle does not recommend the use of `CacheIdentityMap` and `NoIdentityMap` policies.

Understanding the Internals of Soft and Hard Cache Weak Identity Map

The `SoftCacheWeakIdentityMap` and `HardCacheWeakIdentityMap` types of identity map contain two caches:

- Reference cache: implemented as a `LinkedList` that contains soft or hard references, respectively
- Weak cache: implemented as a `HashMap` that contains weak references

When you create a `SoftCacheWeakIdentityMap` or `HardCacheWeakIdentityMap` with a specified size, the reference cache `LinkedList` is exactly this size. The weak cache `HashMap` is initialized to 100 percent of the specified size: the weak cache will grow when more objects than the specified size are read in. Because `TopLink` does not control garbage collection, the JVM can reap the weakly held objects whenever it sees fit.

Because the reference cache is implemented as a `LinkedList`, new objects are added to the end of the list. Because of this, it is by nature a least recently used (LRU) cache: fixed size, object at the top of the list is deleted, provided the maximum size has been reached.

The `SoftCacheWeakIdentityMap` and `HardCacheWeakIdentityMap` are essentially the same type of identity map. The `HardCacheWeakIdentityMap` was constructed to work around an issue with some JVMs.

If your application reaches a low system memory condition frequently enough, or if your platform's JVM treats weak and soft references the same, the objects in the reference cache may be garbage-collected so often that you will not benefit from the performance improvement provided by it. If this is the case, Oracle recommends that you use the `HardCacheWeakIdentityMap`. It is identical to the `SoftCacheWeakIdentityMap` except that it uses hard references in the reference cache. This guarantees that your application will benefit from the performance improvement provided by it.

When an object in a `HardCacheWeakIdentityMap` or `SoftCacheWeakIdentityMap` is pushed out of the reference cache, it gets put in the weak cache. Although it is still cached, `TopLink` cannot guarantee that it will be there for any length of time because the JVM can decide to garbage-collect weak references at anytime.

`TopLink` cleans up dead cache keys after every n^{th} access to a `HardCacheWeakIdentityMap` or `SoftCacheWeakIdentityMap`. For example, if you set the size to 100, then `TopLink` locks the cache and cleans up these cache keys on the 100th access, 200th access, 300th access, and so on. Although this may appear to be a memory leak, it is not: it is a compromise that provides improved performance at the expense of holding on to memory until the n^{th} access. While reducing n frees memory more frequently, it does so at the unacceptable performance cost of performing an enumeration of the cache too frequently.

If you are querying in memory and you absolutely need the guarantee that the objects are in memory, then you should have those classes set to `FullIdentityMap` (see ["Full Identity Map"](#) on page 90-3). Querying in memory with `SoftCacheWeakIdentityMap` or `HardCacheWeakIdentityMap` does not 100 percent guarantee that the query will retrieve the objects that you expect.

When you query in memory, TopLink does not move the objects around to maintain a strict LRU. This behavior occurs only when you merge or query the database. Not supporting in-memory LRU functionality is a compromise that enhances performance. To support in-memory LRU functionality, the identity map would have to be locked for the duration of the move, resulting in performance degradation.

Querying and the Cache

A query that is run against the shared session cache is known as an **in-memory query**. Careful configuration of in-memory querying can improve performance (see ["Using In-Memory Queries"](#) on page 96-30).

By default, a query that looks for a single object based on primary key attempts to retrieve the required object from the cache first, searches the data source only if the object is not in the cache. All other query types search the database first, by default. You can specify whether a given query runs against the in-memory cache, the database, or both.

For more information, see ["Queries and the Cache"](#) on page 96-29.

Handling Stale Data

Stale data is an artifact of caching, in which an object in the cache is not the most recent version committed to the data source. To avoid stale data, implement an appropriate cache locking strategy.

By default, TopLink optimizes concurrency to minimize cache locking during read or write operations. Use the default TopLink isolation level, unless you have a very specific reason to change it. For more information on isolation levels in TopLink, see ["Database Transaction Isolation Levels"](#) on page 102-25.

Cache locking regulates when processes read or write an object. Depending on how you configure it, cache locking determines whether a process can read or write an object that is in use within another process.

A well-managed cache makes your application more efficient. There are very few cases in which you turn the cache off entirely, because the cache reduces database access, and is an important part of managing object identity.

To make the most of your cache strategy and to minimize your application's exposure to stale data, Oracle recommends the following:

- [Configure a Locking Policy](#)
- [Configure the Cache on a Per-Class Basis](#)
- [Force a Cache Refresh When Required on a Per-Query Basis](#)
- [Configure Cache Invalidation](#)
- [Configure Cache Coordination](#)

Configure a Locking Policy

Make sure you configure a locking policy so that you can prevent or at least identify when values have already changed on an object you are modifying. Typically, this is

done using optimistic locking. TopLink offers several locking policies such as numeric version field, time-stamp version field, and some or all fields.

For more information, see ["Configuring Locking Policy"](#) on page 28-62.

Configure the Cache on a Per-Class Basis

If other applications can modify the data used by a particular class, use a weaker style of cache for the class. For example, the `SoftCacheWeakIdentityMap` or `WeakIdentityMap` minimizes the length of time the cache maintains an object whose reference has been removed.

For more information, see ["Configuring Cache Type and Size at the Descriptor Level"](#) on page 28-35.

Force a Cache Refresh When Required on a Per-Query Basis

Any query can include a flag that forces TopLink to go to the data source for the most up-to-date version of selected objects and update the cache with this information.

For more information, see the following:

- ["Cache Refresh API"](#) on page 90-14
- ["Using DatabaseQuery Queries"](#) on page 98-4
- ["Using Named Queries"](#) on page 98-17

Configure Cache Invalidation

Using descriptor API, you can designate an object as invalid: when any query attempts to read an invalid object, TopLink will go to the data source for the most up to date version of that object and update the cache with this information. You can manually designate an object as invalid or use a `CacheInvalidationPolicy` to control the conditions under which an object is designated invalid.

For more information, see ["Cache Invalidation"](#) on page 90-8.

Configure Cache Coordination

If your application is primarily read-based and the changes are all being performed by the same Java application operating with multiple, distributed sessions, you may consider using the TopLink cache coordination feature. Although this will not prevent stale data, it should greatly minimize it.

For more information, see ["Cache Coordination"](#) on page 90-9.

Explicit Query Refreshes

Some distributed systems require only a small number of objects to be consistent across the servers in the system. Conversely, other systems require that several specific objects must always be guaranteed to be up-to-date, regardless of the cost. If you build such a system, you can explicitly refresh selected objects from the database at appropriate intervals, without incurring the full cost of distributed cache coordination.

To implement this type of strategy, do the following:

1. Configure a set of queries that refresh the required objects.
2. Establish an appropriate refresh policy.
3. Invoke the queries as required to refresh the objects.

Refresh Policy

When you execute a query, if the required objects are in the cache, TopLink returns the cached objects without checking the database for a more recent version. This reduces the number of objects that TopLink must build from database results, and is optimal for noncoordinated cache environments. However, this may not always be the best strategy for a coordinated cache environment.

To override this behavior, set a refresh policy that specifies that the objects from the database always take precedence over objects in the cache. This updates the cached objects with the data from the database.

You can implement this type of refresh policy on each TopLink descriptor, or just on certain queries, depending upon the nature of the application.

For more information, see the following:

- ["Configuring Cache Refreshing"](#) on page 28-27
- ["Refreshing the Cache"](#) on page 96-35

Note: Refreshing does not prevent phantom reads from occurring. See ["Refreshing Finder Results"](#) on page 96-38.

EJB Finders and Refresh Policy

When you invoke a `findByPrimaryKey` finder, if the object exists in the cache, TopLink returns that copy. This is the default behavior, regardless of the refresh policy. To force a database query, you can configure the query to refresh by calling `refreshIdentityMapResult` method on it.

For more information, see ["Queries and the Cache"](#) on page 96-29.

Cache Invalidation

By default, objects remain in the session cache until they are explicitly deleted (see ["Deleting Objects"](#) on page 101-7) or garbage collected when using a weak identity map (see ["Configuring Cache Type and Size at the Project Level"](#) on page 22-13).

Alternatively, you can configure any object with a `CacheInvalidationPolicy` that lets you specify, either automatically or manually, under what circumstances a cached object is invalid: when any query attempts to read an invalid object, TopLink will go to the data source for the most up-to-date version of that object, and update the cache with this information.

You can use any of the following `CacheInvalidationPolicy` instances:

- `DailyCacheInvalidationPolicy`: the object is automatically flagged as invalid at a specified time of day.
- `NoExpiryCacheInvalidationPolicy`: the object can only be flagged as invalid by explicitly calling `oracle.toplink.sessions.IdentityMapAccessor.invalidateObject`.
- `TimeToLiveCacheInvalidationPolicy`: the object is automatically flagged as invalid after a specified time period has elapsed since the object was read.

You can configure a cache invalidation policy in the following ways:

- At the project level that applies to all objects (["Configuring Cache Expiration at the Project Level"](#) on page 22-19)

- At the descriptor level to override the project level configuration on a per-object basis ("[Configuring Cache Expiration at the Descriptor Level](#)" on page 28-40)
- At the query level that applies to the results returned by the query ("[Configuring Cache Expiration at the Query Level](#)" on page 99-22)

If you configure a query to cache results in its own internal cache (see "[Caching Query Results](#)" on page 96-36), the cache invalidation policy you configure at the query level applies to the query's internal cache in the same way it would apply to the session cache.

If you are using a coordinated cache (see "[Cache Coordination](#)" on page 90-9), you can customize how TopLink communicates the fact that an object has been declared invalid. For more information, see "[Configuring Cache Coordination Change Propagation at the Descriptor Level](#)" on page 28-38.

Cache Coordination

The need to maintain up-to-date data for all applications is a key design challenge for building a distributed application. The difficulty of this increases as the number of servers within an environment increases. TopLink provides a distributed cache coordination feature that ensures data in distributed applications remains current.

Cache coordination reduces the number of optimistic lock exceptions encountered in a distributed architecture, and decreases the number of failed or repeated transactions in an application. However, cache coordination in no way eliminates the need for an effective locking policy. To effectively ensure working with up-to-date data, cache coordination must be used with optimistic or pessimistic locking. Oracle recommends that you use cache coordination with an optimistic locking policy (see "[Configuring Locking Policy](#)" on page 28-62).

You can use cache invalidation to improve cache coordination efficiency. For more information, see "[Cache Invalidation](#)" on page 90-8.

For more information, see "[Understanding Cache Coordination](#)" on page 90-10.

Cache Isolation

Isolated client sessions provide a mechanism for disabling the shared server session cache. Any classes marked as isolated only cache objects relative to the life cycle of their client session. These classes never utilize the shared server session cache. This is the best mechanism to prevent caching as it is configured on a per-class basis allowing caching for some classes, and denying for others.

For more information, see "[Isolated Client Sessions](#)" on page 75-19.

Cache Locking and Transaction Isolation

By default, TopLink optimizes concurrency to minimize cache locking during read or write operations. Use the default TopLink transaction isolation configuration unless you have a very specific reason to change it.

For more information, see "[Database Transaction Isolation Levels](#)" on page 102-25.

Cache Optimization

Tune the TopLink cache for each class to help eliminate the need for distributed cache coordination. Always tune these settings before implementing cache coordination.

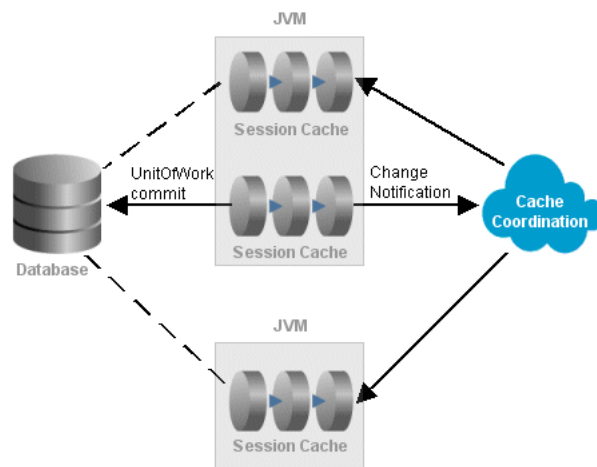
For more information, see "[Cache Optimization](#)" on page 11-13.

Understanding Cache Coordination

As [Figure 90–2](#) shows, cache coordination is a session feature that allows multiple, possibly distributed, instances of a session to broadcast object changes among each other so that each session's cache is either kept up-to-date or notified that the cache must update an object from the data source the next time it is read.

Note: You cannot use isolated client sessions (see ["Isolated Client Sessions"](#) on page 75-19) with cache coordination.

Figure 90–2 Cache Coordination



When sessions are distributed, that is, when an application contains multiple sessions (in the same JVM, in multiple JVMs, possibly on different servers), as long as the servers hosting the sessions are interconnected on the network, sessions can participate in cache coordination. Coordinated cache types that require discovery services also require the servers to support User Datagram Protocol (UDP) communication and multicast configuration (for more information, see ["Coordinated Cache Architecture"](#) on page 90-11).

This section describes the following:

- [When to use Cache Coordination](#)
- [Coordinated Cache Architecture](#)
- [Coordinated Cache Types](#)
- [Custom Coordinated Cache](#)

For more information, see ["Configuring a Coordinated Cache"](#) on page 91-1.

When to use Cache Coordination

Cache coordination can enhance performance and reduce the likelihood of stale data for applications that have the following characteristics:

- Changes are all being performed by the same Java application operating with multiple, distributed sessions
- Primarily read-based
- Regularly requests and updates the same objects

To maximize performance, avoid cache coordination for applications that do not have these characteristics. For more information about alternatives to cache coordination, see ["Cache Optimization"](#) on page 11-13.

Cache coordination enhances performance mainly by avoiding data source access.

Cache coordination reduces the occurrence of stale data by increasing the likelihood that distributed caches are kept up-to-date with changes and are notified when one of the distributed caches must update an object from the data source the next time it is read.

Cache coordination reduces the number of optimistic lock exceptions encountered in a distributed architecture, and decreases the number of failed or repeated transactions in an application. However, cache coordination in no way eliminates the need for an effective locking policy. To effectively ensure working with up-to-date data, cache coordination must be used with optimistic or pessimistic locking. Oracle recommends that you use cache coordination with an optimistic locking policy (see ["Configuring Locking Policy"](#) on page 28-62).

For other options to reduce the likelihood of stale data, see ["Handling Stale Data"](#) on page 90-6.

Coordinated Cache Architecture

TopLink provides coordinated cache implementations that perform discovery and message transport services using various technologies including the following:

- Java Message Service (JMS)–See ["JMS Coordinated Cache"](#) on page 90-12
- Remote Method Invocation (RMI)–See ["RMI Coordinated Cache"](#) on page 90-12
- Common Object Request Broker Architecture (CORBA)–See ["CORBA Coordinated Cache"](#) on page 90-13

Regardless of the type of discovery and message transport you choose to use, the following are the principal objects that provide coordinated cache functionality:

- [Session](#)
- [Descriptor](#)
- [Unit of Work](#)

Session

When you enable a session for change propagation, the session provides discovery and message transport services using either JMS, RMI, or CORBA.

Discovery services ensure that sessions announce themselves to other sessions participating in cache coordination. Discovery services use UDP communication and multicast configuration to monitor sessions as they join and leave the coordinated cache. All coordinated cache types (except JMS) require discovery services.

Message transport services allow the session to broadcast object change notifications to other sessions participating in cache coordination when a unit of work from this session commits a change.

Descriptor

You can configure how object changes are broadcast on a descriptor-by-descriptor basis. This lets you fine-tune the type of notification to make.

For example, for an object with few attributes, you can configure its descriptor to send object changes. For an object with many attributes, it may be more efficient to configure its descriptor so that the object is flagged as invalid (so that other sessions will know to update the object from the data source the next time it is read).

Unit of Work

Only changes committed by a unit of work are subject to propagation when cache coordination is enabled. The unit of work computes the appropriate change set based on the descriptor configuration of affected objects.

Coordinated Cache Types

You can create the following types of coordinated cache:

- [JMS Coordinated Cache](#)
- [RMI Coordinated Cache](#)
- [CORBA Coordinated Cache](#)

JMS Coordinated Cache

For a JMS coordinated cache, when a particular session's coordinated cache starts up, it uses its JNDI naming service information to locate and create a connection to the JMS server. The coordinated cache is ready when all participating sessions are connected to the same topic on the same JMS server. At this point, sessions can start sending and receiving object change messages. You can then configure all sessions that are participating in the same coordinated cache with the same JMS and JNDI naming service information.

Because you must supply the necessary information to connect to the JMS Topic, a JMS coordinated cache does not use a discovery service.

If you do use cache coordination, Oracle recommends that you use JMS cache coordination: JMS is robust, easy to configure, and provides efficient support for asynchronous change propagation.

For more information, see [Chapter 92, "Configuring a JMS Coordinated Cache"](#).

RMI Coordinated Cache

For an RMI coordinated cache, when a particular session's coordinated cache starts up, the session binds its connection in its naming service (either an RMI registry or JNDI), creates an announcement message (that includes its own naming service information), and broadcasts the announcement to its multicast group (see ["Configuring a Multicast Group Address"](#) on page 91-4 and ["Configuring a Multicast Port"](#) on page 91-5). When a session that belongs to the same multicast group receives this announcement, it uses the naming service information in the announcement message to establish bidirectional connections with the newly announced session's coordinated cache. The coordinated cache is ready when all participating sessions are interconnected in this way, at which point, sessions can start sending and receiving object change messages. You can then configure each session with naming information that identifies the host on which the session is deployed.

If you do use cache coordination, Oracle recommends that you use RMI cache coordination only if you require synchronous change propagation (see ["Configuring the Synchronous Change Propagation Mode"](#) on page 91-2).

TopLink also supports cache coordination using RMI over the Internet Inter-ORB Protocol (IIOP). An RMI/IIOP coordinated cache uses RMI (and a JNDI naming service) for discovery and message transport services.

Note: If you use an RMI coordinated cache, Oracle recommends that you use RMI/IIOP only if absolutely necessary.

For more information, see [Chapter 93, "Configuring an RMI Coordinated Cache"](#).

CORBA Coordinated Cache

For a CORBA coordinated cache, when a particular session's coordinated cache starts up, the session binds its connection in JNDI, creates an announcement message (that includes its own JNDI naming service information), and broadcasts the announcement to its multicast group (see ["Configuring a Multicast Group Address"](#) on page 91-4 and ["Configuring a Multicast Port"](#) on page 91-5). When a session that belongs to the same multicast group receives this announcement, it uses the naming service information in the announcement message to establish bidirectional connections with the newly announced session's coordinated cache. The coordinated cache is ready when all participating sessions are interconnected in this way, at which point, sessions can start sending and receiving object change messages. You can then configure each session with naming information that identifies the host on which the session is deployed.

Currently, TopLink provides support for the Sun Object Request Broker.

For more information on configuring a CORBA coordinated cache, see [Chapter 94, "Configuring a CORBA Coordinated Cache"](#).

Custom Coordinated Cache

Using the classes in `oracle.toplink.remotecommand` package, you can define your own coordinated cache for custom solutions. For more information, contact your TopLink support representative.

Once you have created the required cache coordination classes, for more information on configuring a user-defined coordinated cache, see [Chapter 95, "Configuring a Custom Coordinated Cache"](#).

Understanding the Cache API

To configure the TopLink cache, you use the appropriate API in the following objects:

- [Object Identity API](#)
- [Cache Refresh API](#)
- [Cache Invalidation API](#)
- [Cache Coordination API](#)

Object Identity API

You configure object identity using the `Descriptor` API summarized in [Example 90-1](#).

For more information, see ["Configuring Cache Type and Size at the Descriptor Level"](#) on page 28-35.

Example 90–1 Object Identity Descriptor API

```
useCacheIdentityMap()  
useFullIdentityMap()  
useHardCacheWeakIdentityMap()  
useNoIdentityMap()  
useSoftCacheWeakIdentityMap()  
useWeakIdentityMap()
```

Cache Refresh API

You configure cache refresh using the Descriptor API summarized in [Example 90–2](#).

Example 90–2 Cache Refresh Descriptor API

```
alwaysRefreshCache()  
alwaysRefreshCacheOnRemote()  
disableCacheHits()  
disableCacheHitsOnRemote()  
onlyRefreshCacheIfNewerVersion()
```

You can also configure cache refresh using the following API calls:

- **Session:** `refreshObject` method
- **DatabaseSession and UnitOfWork:** `refreshAndLockObject` methods
- **ObjectLevelReadQuery:** `refreshIdentityMapResult` and `refreshRemoteIdentityMapResult` methods

For more information, see "[Configuring Cache Refreshing](#)" on page 28-27.

Cache Invalidation API

You configure cache invalidation using Descriptor methods `getCacheInvalidationPolicy` and `setCacheInvalidationPolicy` to configure an `oracle.toplink.descriptors.invalidation.CacheInvalidationPolicy`.

You can use any of the following `CacheInvalidationPolicy` instances:

- **DailyCacheInvalidationPolicy:** The object is automatically flagged as invalid at a specified time of day.
- **NoExpiryCacheInvalidationPolicy:** The object can only be flagged as invalid by explicitly calling `oracle.toplink.sessions.IdentityMapAccessor.invalidateObject`.
- **TimeToLiveCacheInvalidationPolicy:** The object is automatically flagged as invalid after a specified time period has elapsed since the object was read.

You can also configure cache invalidation using a variety of API calls accessible through the **Session**. The `oracle.toplink.sessions.IdentityMapAccessor` provides the following methods:

- **getRemainingValidTime:** Returns the remaining life of the specified object. This time represents the difference between the next expiry time of the object and its read time.

- `invalidateAll`: Sets all objects for all classes to be invalid in TopLink identity maps.
- `invalidateClass(Class klass)` and `invalidateClass(Class klass, boolean recurse)`: Set all objects of a specified class to be invalid in TopLink identity maps.
- `invalidateObject(Object object)`, `invalidateObject(Record rowWithPrimaryKey, Class klass)` and `invalidateObject(Vector primaryKey, Class klass)`: Set an object to be invalid in TopLink identity maps.
- `invalidateObjects(Expression selectionCriteria)` and `invalidateObjects(Vector collection)`: Set all objects from the specified Expression/collection to be invalid in TopLink identity maps.
- `isValid(DatabaseRow rowContainingPrimaryKey, Class theClass)`, `isValid(Object object)` and `isValid(java.util.Vector primaryKey, Class theClass)`: Return true if the object is valid in TopLink identity maps.

For more information, see the following:

- ["Configuring Cache Expiration at the Project Level"](#) on page 22-19
- ["Configuring Cache Expiration at the Descriptor Level"](#) on page 28-40
- ["Configuring Cache Expiration at the Query Level"](#) on page 99-22

Cache Coordination API

You configure cache coordination using the `Session` methods summarized in [Example 90-3](#).

You configure how object changes are propagated using the `Descriptor` methods summarized in [Example 90-4](#).

For more information, see ["Configuring Common Coordinated Cache Options"](#) on page 91-1.

Example 90-3 Cache Coordination Session API

```
Session.getCommandManager()
    setShouldPropagateAsynchronously(boolean)
Session.getCommandManager().getDiscoveryManager()
    setAnnouncementDelay()
    setMulticastGroupAddress()
    setMulticastPort()
    setPacketTimeToLive()
Session.getCommandManager().getTransportManager()
    setEncryptedPassword()
    setInitialContextFactoryName()
    setLocalContextProperties(Hashtable)
    setNamingServiceType() passing in one of:
        TransportManager.JNDI_NAMING_SERVICE
        TransportManager.REGISTRY_NAMING_SERVICE
    setPassword()
    setRemoteContextProperties(Hashtable)
    setShouldRemoveConnectionOnError()
    setUsername()
```

Example 90–4 Cache Coordination Descriptor API

setCacheSynchronizationType() passing in one of:
ClassDescriptor.DO_NOT_SEND_CHANGES
ClassDescriptor.INVALIDATE_CHANGED_OBJECTS
ClassDescriptor.SEND_NEW_OBJECTS_WITH_CHANGES
ClassDescriptor.SEND_OBJECT_CHANGES

Configuring a Coordinated Cache

This chapter describes how to configure a TopLink coordinated cache.

Table 91–1 *Configuring TopLink Coordinated Caches*

If you are configuring a...	See...
JMS Coordinated Cache	Chapter 92, "Configuring a JMS Coordinated Cache"
RMI Coordinated Cache	Chapter 93, "Configuring an RMI Coordinated Cache"
CORBA Coordinated Cache	Chapter 94, "Configuring a CORBA Coordinated Cache"
Custom Coordinated Cache	Chapter 95, "Configuring a Custom Coordinated Cache"

Table 91–1 lists the configurable options shared by two or more TopLink coordinated cache types.

For more information, see ["Understanding Cache Coordination"](#) on page 90-10.

Configuring Common Coordinated Cache Options

Table 91–1 lists the configurable options shared by two or more TopLink coordinated cache types. In addition to the configurable options described here, you must also configure the options described for the specific [Coordinated Cache Types](#), as shown in **Table 91–2**.

Table 91–2 *Common Coordinated Cache Options*

Option	Type	TopLink Workbench	Java
"Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 28-38	Basic	✓	✓
"Configuring the Synchronous Change Propagation Mode" on page 91-2	Basic	✓	✓
"Configuring a Service Channel" on page 91-3	Basic	✓	✓
"Configuring a Multicast Group Address" on page 91-4	Basic	✓	✓
"Configuring a Multicast Port" on page 91-5	Basic	✓	✓
"Configuring a Naming Service Type" on page 91-7	Basic	✓	✓
"Configuring an Announcement Delay" on page 91-11	Basic	✓	✓
"Configuring Connection Handling" on page 91-13	Advanced	✓	✓
"Configuring Context Properties" on page 91-14	Advanced	✓	✓
"Configuring a Packet Time-to-Live" on page 91-15	Advanced	✓	✓

Configuring the Synchronous Change Propagation Mode

You can configure whether the coordinated cache should propagate object changes asynchronously or synchronously.

[Table 91–3](#) summarizes which coordinated caches support propagation mode configuration.

Table 91–3 Coordinated Cache Support for Propagation Mode Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
JMS Coordinated Cache (asynchronous only)		
RMI Coordinated Cache	✓	✓
CORBA Coordinated Cache	✓	✓
Custom Coordinated Cache		

Synchronous propagation mode forces the session to wait for an acknowledgement before sending the next object change notification: this reduces the likelihood of stale data at the expense of performance.

Asynchronous propagation mode allows the session to create separate threads to propagate changes to remote servers. TopLink returns control to the client immediately after the local commit operation, whether or not the changes merge successfully on the remote servers. This offers superior performance for applications that are somewhat tolerant of stale data.

For more information, "[Handling Stale Data](#)" on page 90-6.

Using TopLink Workbench

To specify the change propagation mode, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.
3. Ensure the **Enable Cache Coordination** option is selected, then select the appropriate coordinated cache **Type** (see [Table 91–3](#)). The cache coordination options appear on the tab.

Figure 91–1 Cache Coordination Tab, Synchronous Field

The screenshot shows the 'Cache Coordination' tab of a configuration window. The 'Enable Cache Coordination' checkbox is checked. The 'Type' is set to 'RMI'. The 'Channel' is 'TopLinkCommandChannel'. The 'Multicast Group Address' is '226.10.12.64'. The 'Multicast Port' is '3,121'. The 'Packet Time to Live' is '2'. The 'Announcement Delay' is '1,000'. The 'Remove Connection on Error' checkbox is checked. The 'Synchronous' checkbox is checked and highlighted with a red circle. The 'Registry Naming Service' is not selected. The 'JNDI Naming Service' is selected, with 'URL' empty, 'User Name' 'admin', 'Password' 'password', and 'Initial Context Factory' 'com.evermind.server.rmi.RMIInitialContextFactory'. There are 'Properties...' and 'Browse...' buttons.

4. Select the **Synchronous** option to use synchronous change propagation. Do not select this option to use asynchronous change propagation.

Configuring a Service Channel

The **service channel** is the name of the TopLink coordinated cache channel to which sessions subscribe in order to participate in the same coordinated cache. Such sessions use the service channel to exchange messages with each other. Messages sent on other service channels will not be exchanged with this coordinated cache.

Table 91–4 summarizes which coordinated caches support service channel configuration.

Table 91–4 Coordinated Cache Support for Service Channel Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
JMS Coordinated Cache		
RMI Coordinated Cache	✓	✓
CORBA Coordinated Cache	✓	✓
Custom Coordinated Cache	✓	✓

Using TopLink Workbench

To specify the service channel, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.

3. Ensure the **Enable Cache Coordination** option is selected, then select the appropriate coordinated cache **Type** (see [Table 91–4](#)). The cache coordination options appear on the tab.

Figure 91–2 Cache Coordination Tab, Channel Field

The screenshot shows the 'Cache Coordination' tab of a configuration window. The 'Enable Cache Coordination' checkbox is checked. Below it, the 'Type' dropdown menu is set to 'RMI'. The 'Channel' text field is highlighted with a red circle and contains the value 'TopLinkCommandChannel'. Other configuration fields include 'Multicast Group Address' (226.10.12.64), 'Multicast Port' (3,121), 'Packet Time to Live' (2), and 'Announcement Delay' (1,000). There are also checkboxes for 'Remove Connection on Error' and 'Synchronous', and radio buttons for 'Registry Naming Service' and 'JNDI Naming Service'. The 'JNDI Naming Service' section includes fields for 'URL', 'User Name' (admin), 'Password' (password), and 'Initial Context Factory' (com.evermind.server.rmi.RMIInitialContextFactory).

4. In the **Channel** field, enter the name of the service channel for this coordinated cache.

Configuring a Multicast Group Address

A multicast group address is an Internet Protocol (IP) address in the range 224.0.0.0 to 239.255.255.255 that identifies the members of an IP multicast group. To efficiently broadcast the same message to all members of an IP multicast group, you configure each recipient with the same multicast group address and send the message to that address.

[Table 91–5](#) summarizes which coordinated caches support multicast group address configuration.

Table 91–5 Coordinated Cache Support for Multicast Group Address Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
JMS Coordinated Cache		
RMI Coordinated Cache	✓	✓
CORBA Coordinated Cache	✓	✓
Custom Coordinated Cache		

Note: Ensure your host and network are configured to support multicast operation before configuring this option.

In addition to configuring the multicast group address, you must also configure the multicast port (see ["Configuring a Multicast Port"](#) on page 91-5) for the coordinated cache types shown in [Table 91-5](#).

Using TopLink Workbench

To specify the multicast group address, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.
3. Ensure the **Enable Cache Coordination** option is selected, then select the appropriate coordinated cache **Type** (see [Table 91-5](#)). The cache coordination options appear on the tab.

Figure 91-3 *Cache Coordination Tab, Multicast Group Address Field*

The screenshot shows the 'Cache Coordination' tab in the TopLink Workbench. The 'Enable Cache Coordination' checkbox is checked. The 'Type' dropdown is set to 'RMI'. The 'Channel' text box contains 'TopLinkCommandChannel'. The 'Multicast Group Address' text box, which is circled in red, contains '226.10.12.64'. The 'Multicast Port' spinner is set to 3,121. The 'Packet Time to Live' spinner is set to 2. The 'Announcement Delay' spinner is set to 1,000. There are two checked checkboxes: 'Remove Connection on Error' and 'Synchronous'. Below these are two sections: 'Registry Naming Service' (with an unchecked radio button and an empty URL field) and 'JNDI Naming Service' (with a checked radio button, empty fields for URL, User Name, and Password, and a text box for Initial Context Factory containing 'com.evermind.server.rmi.RMIInitialContextFactory' with a 'Browse...' button next to it). A 'Properties...' button is at the bottom left.

4. Enter the multicast group address in the range 224.0.0.0 to 239.255.255.255 to subscribe this session to a given coordinated cache.

Configuring a Multicast Port

The multicast port is the port on which multicast messages are received. Members of a multicast group (see ["Configuring a Multicast Group Address"](#) on page 91-4) rely on messages broadcast to their multicast group address to communicate with one another.

Table 91–6 summarizes which coordinated caches support multicast port configuration.

Table 91–6 Coordinated Cache Support for Multicast Port Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
JMS Coordinated Cache		
RMI Coordinated Cache	✓	✓
CORBA Coordinated Cache	✓	✓
Custom Coordinated Cache		

Note: Ensure your host and network are configured to support multicast operation before configuring this option

Using TopLink Workbench

To specify the multicast port, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.
3. Ensure the **Enable Cache Coordination** option is selected, then select the appropriate coordinated cache **Type** (see Table 91–6). The cache coordination options appear on the tab.

Figure 91–4 Cache Coordination Tab, Multicast Port Field

The screenshot shows the 'Cache Coordination' tab in the TopLink Workbench. The 'Enable Cache Coordination' checkbox is checked. The 'Type' dropdown is set to 'RMI'. The 'Channel' is 'TopLinkCommandChannel'. The 'Multicast Group Address' is '226.10.12.64'. The 'Multicast Port' is '3121', which is circled in red. Other settings include 'Packet Time to Live' (2), 'Announcement Delay' (1.000), 'Remove Connection on Error' (checked), 'Synchronous' (checked), and 'Registry Naming Service' (selected). The 'JNDI Naming Service' section is also visible with fields for URL, User Name, Password, and Initial Context Factory.

4. Enter the multicast port on which messages broadcast to the multicast group address are received.

Configuring a Naming Service Type

The session's message transport service uses a naming service when it looks up connections to other sessions in the coordinated cache. You can configure the message transport service to look up remote objects using an RMI registry or Java Naming and Directory Interface (JNDI). By default, JNDI is used.

[Table 91-7](#) summarizes which coordinated caches support naming service configuration.

Table 91-7 Coordinated Cache Support for Naming Service Configuration

Coordinated Cache	JNDI Naming Service	RMI Registry Naming Service
JMS Coordinated Cache	✓	
RMI Coordinated Cache	✓	✓
CORBA Coordinated Cache	✓	
Custom Coordinated Cache		

For information, see:

- ["Configuring RMI Registry Naming Service Information"](#) on page 91-9
- ["Configuring JNDI Naming Service Information"](#) on page 91-7

Configuring JNDI Naming Service Information

The session's message transport service uses a naming service when it looks up connections to other sessions in the coordinated cache. If you choose to use a JNDI naming service, you must configure JNDI naming service information.

[Table 91-8](#) summarizes which coordinated caches support JNDI naming service configuration.

Table 91-8 Coordinated Cache Support for JNDI Naming Service Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
JMS Coordinated Cache	✓	✓
RMI Coordinated Cache	✓	✓
CORBA Coordinated Cache	✓	✓
Custom Coordinated Cache		

TopLink uses JNDI naming service information differently, depending on the type of coordinated cache.

For a JMS coordinated cache, when a particular session's coordinated cache starts up, it uses its JNDI naming service information to locate and create a connection to the JMS server. The coordinated cache is ready when all participating sessions are connected to the JMS server. At this point, sessions can start sending and receiving object change messages. You can then configure all sessions that are participating in the same coordinated cache with the same JNDI naming service information.

For an RMI or CORBA coordinated cache, when a particular session's coordinated cache starts up, the session binds its connection in JNDI, creates an announcement message (that includes its own JNDI naming service information), and broadcasts the announcement to its multicast group (see ["Configuring a Multicast Group Address"](#) on page 91-4 and ["Configuring a Multicast Port"](#) on page 91-5). When a session that belongs to the same multicast group receives this announcement, it uses the JNDI naming service information in the announcement message to establish bidirectional connections with the newly announced session's coordinated cache. The coordinated cache is ready when all participating sessions are interconnected in this way, at which point, sessions can start sending and receiving object change messages. You can then configure each session with JNDI naming information that identifies the host on which the session is deployed.

Using TopLink Workbench

To specify the sessions's JNDI naming service, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.
3. Ensure the **Enable Cache Coordination** option is selected, then select the appropriate coordinated cache **Type** (see [Table 91-8](#)). The cache coordination options appear on the tab.

Figure 91-5 Cache Coordination Tab, JNDI Naming Service Options

The screenshot shows the 'Cache Coordination' tab in the TopLink Workbench. The 'Enable Cache Coordination' checkbox is checked. The 'Type' dropdown is set to 'RMI'. The 'Channel' is 'TopLinkCommandChannel'. The 'Multicast Group Address' is '226.10.12.64'. The 'Multicast Port' is '3,121'. The 'Packet Time to Live' is '2'. The 'Announcement Delay' is '1,000'. The 'Remove Connection on Error' and 'Synchronous' checkboxes are checked. The 'Registry Naming Service' radio button is selected, but the 'JNDI Naming Service' radio button is circled in red. Below the 'JNDI Naming Service' section, the 'URL' is empty, 'User Name' is 'admin', 'Password' is 'password', and 'Initial Context Factory' is 'com.evermind.server.rmi.RMIInitialContextFactory'. There are 'Properties...' and 'Browse...' buttons.

4. Complete the Naming Service options.

Field	Description
URL	The location of the JNDI naming service. For a JMS coordinated cache: If you are using the Oracle Containers for J2EE (OC4J) JNDI naming service and all the hosts in your coordinated cache can communicate using the OC4J proprietary RMI protocol ORMI, use a URL similar to the following: <pre>ormi://<JMS-host-IP>:<JMS-host-port></pre> where <code>JMS-host-IP</code> is the IP address of the host on which the JMS service provider is running and <code>JMS-host-port</code> is the port on which the JMS service provider is listening for JMS requests. For an RMI or CORBA coordinated cache: If you are using the OC4J JNDI naming service and all the hosts in your coordinated cache can communicate using the OC4J proprietary RMI protocol ORMI on OC4J default port 23791, use a URL similar to the following: <pre>ormi://<session-host-IP>:23791</pre> where <code>session-host-IP</code> is the IP address of the host on which this session is deployed.
Username	The user name required to log in to the JNDI naming service. The value you enter defines the <code>Context.SECURITY_PRINCIPAL</code> environment property.
Password	The plain text (unencrypted) password required to log in to the JNDI naming service. The password appears in plain text in TopLink Workbench, but it is encrypted when written to the <code>sessions.xml</code> file The value you enter defines the <code>Context.SECURITY_CREDENTIALS</code> environment property.
Initial Context Factory	The name of the factory class, provided by your JNDI naming service provider, that implements the <code>javax.naming.spi.InitialContextFactory</code> interface. This factory class is used to create a <code>javax.naming.Context</code> instance that can access the JNDI naming service provider's context implementation. The value you enter defines the <code>Context.INITIAL_CONTEXT_FACTORY</code> environment property.
Properties	The JNDI context properties. Click Properties to configure custom JNDI context properties (see "Configuring Context Properties" on page 91-14).

Configuring RMI Registry Naming Service Information

The session's message transport service uses a naming service when it looks up connections to other sessions in the coordinated cache. If you choose to use an RMI registry naming service, you can configure RMI registry naming service options.

[Table 91-8](#) summarizes which coordinated caches support RMI registry naming service configuration.

Table 91–9 Coordinated Cache Support for RMI Registry Naming Service Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
JMS Coordinated Cache		
RMI Coordinated Cache	✓	✓
CORBA Coordinated Cache		
Custom Coordinated Cache		

For an RMI coordinated cache, when a particular session's coordinated cache starts up, the session binds its connection in its RMI registry, creates an announcement message (that includes its own naming service information), and broadcasts the announcement to its multicast group (see ["Configuring a Multicast Group Address"](#) on page 91-4 and ["Configuring a Multicast Port"](#) on page 91-5). When a session that belongs to the same multicast group receives this announcement, it uses the JNDI naming service information in the announcement message to establish bidirectional connections with the newly announced session's coordinated cache. The coordinated cache is ready when all participating sessions are interconnected in this way, at which point, sessions can start sending and receiving object change messages. You can then configure each session with RMI registry naming information that identifies the host on which the session is deployed.

Using TopLink Workbench

To specify the sessions's registry naming service, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor window.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.
3. Ensure the **Enable Cache Coordination** option is selected, then select the appropriate coordinated cache **Type** (see [Table 91–9](#)). The cache coordination options appear on the tab.

Figure 91–6 Cache Coordination Tab, Naming Service Options

The screenshot shows the 'Cache Coordination' tab of a configuration window. The 'Enable Cache Coordination' checkbox is checked. The 'Type' is set to 'RMI'. The 'Channel' is 'TopLinkCommandChannel'. The 'Multicast Group Address' is '226.10.12.64'. The 'Multicast Port' is '3,121'. The 'Packet Time to Live' is '2'. The 'Announcement Delay' is '1,000'. Under 'Naming Service Options', the 'Registry Naming Service' radio button is selected and circled in red. Below it, the 'JNDI Naming Service' section is visible with fields for URL, User Name (admin), Password (password), and Initial Context Factory (com.evermind.server.rmi.RMIInitialContextFactory). There are 'Properties...' and 'Browse...' buttons.

Use the following information to configure the naming service options:

Field	Description
URL	Assuming that you are using the OC4J JNDI naming service and that all the hosts in your coordinated cache can communicate using the OC4J proprietary RMI protocol ORMI on OC4J default port 23791, use a URL similar to the following: <code>ormi://<session-host-IP>:23791</code> where <code>session-host-IP</code> is the IP address of the host on which this session is deployed.

Configuring an Announcement Delay

Use the announcement delay option to set the amount of time (in milliseconds) that a session should wait between the time that it is available and the time that it broadcasts its announcement message to the members of the coordinated cache. This additional delay may be necessary to give some systems more time to post their connections into the naming service (see ["Configuring a Naming Service Type"](#) on page 91-7).

[Table 91–10](#) summarizes which coordinated caches support announcement delay configuration.

Table 91–10 Coordinated Cache Support for Announcement Delay Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
JMS Coordinated Cache		
RMI Coordinated Cache	✓	✓

Table 91–10 (Cont.) Coordinated Cache Support for Announcement Delay Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
CORBA Coordinated Cache	✓	✓
Custom Coordinated Cache		

In addition to announcement delay, you may also need to consider packet time-to-live configuration (see ["Configuring a Packet Time-to-Live"](#) on page 91-15).

Using TopLink Workbench

To specify the announcement delay (in milliseconds) for an RMI coordinated cache, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.
3. Ensure the **Enable Cache Coordination** option is selected, then select the appropriate coordinated cache **Type** (see [Table 91–10](#)). The cache coordination options appear on the tab.

Figure 91–7 Cache Coordination Tab, Announcement Delay Field

The screenshot shows the 'Cache Coordination' tab in the TopLink Workbench. The 'Enable Cache Coordination' checkbox is checked. The 'Type' dropdown is set to 'RMI'. The 'Channel' is 'TopLinkCommandChannel'. The 'Multicast Group Address' is '226.10.12.64'. The 'Multicast Port' is '3,121'. The 'Packet Time to Live' is '2'. The 'Announcement Delay' is '1,000' and is circled in red. Below these fields are checkboxes for 'Remove Connection on Error' (checked) and 'Synchronous' (checked). There are two sections for naming services: 'Registry Naming Service' (disabled) and 'JNDI Naming Service' (selected). The 'JNDI Naming Service' section includes fields for 'URL', 'User Name' (admin), 'Password' (password), and 'Initial Context Factory' (com.evermind.server.rmi.RMIInitialContextFactory). A 'Browse...' button is next to the 'Initial Context Factory' field. A 'Properties...' button is at the bottom left.

4. Select the amount of time (in milliseconds) that this session should wait between the time that it is available and the time that it broadcasts its announcement message to the members of the coordinated cache.

Configuring Connection Handling

The session's transport manager creates connections to the various members of the coordinated cache. If a communication error occurs on one of these connections, you can configure the session to either ignore the error or remove the connection.

[Table 91–11](#) summarizes which coordinated caches support connection handling configuration.

Table 91–11 Coordinated Cache Support for Connection Handling Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
JMS Coordinated Cache	✓	✓
RMI Coordinated Cache	✓	✓
CORBA Coordinated Cache	✓	✓
Custom Coordinated Cache	✓	✓

If you configure the session to remove the connection on error, the next time the session tries to communicate with that coordinated cache member, it will construct a new connection.

If you configure the session to ignore the error, the next time the session tries to communicate with that coordinated cache member, it will continue to use the same connection.

Using TopLink Workbench

To specify how TopLink handles session connections in the event of an error, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.
3. Ensure the **Enable Cache Coordination** option is selected, then select the appropriate coordinated cache **Type** (see [Table 91–11](#)). The cache coordination options appear on the tab.

Figure 91–8 Cache Coordination Tab, Remove Connection on Error Option

4. Select the **Remove Connection on Error** option to configure the session to remove the data source connection in the event of an error.

Configuring Context Properties

When you configure a coordinated cache to use a JNDI naming service (see ["Configuring a Naming Service Type"](#) on page 91-7), you can add new environment properties to the environment of the initial JNDI context.

[Table 91–12](#) summarizes which coordinated caches support context properties.

Table 91–12 Coordinated Cache Support for Context Properties

Coordinated Cache	Using TopLink Workbench	Using Java
JMS Coordinated Cache	✓	✓
RMI Coordinated Cache ¹	✓	✓
CORBA Coordinated Cache ¹	✓	✓
Custom Coordinated Cache		

¹ When JNDI naming service is selected (see ["Configuring a Naming Service Type"](#) on page 91-7).

Using TopLink Workbench, TopLink uses the new environment properties you add to create the initial context for both local and remote JNDI access.

Using Java, you can configure different properties for local and remote JNDI access using a session customizer class to call `TransportManager` methods `setLocalContextProperties` and `setRemoteContextProperties` (for more information, see ["Configuring Customizer Class"](#) on page 77-13).

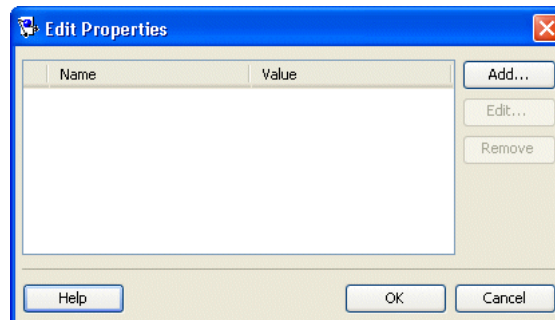
Using TopLink Workbench

To define JNDI context properties, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor.

2. Ensure the **Enable Cache Coordination** option is selected, then select the appropriate coordinated cache **Type** (see [Table 91-12](#)). The cache coordination options appear on the tab.
3. Ensure the **JNDI Naming Service** option is selected. See ["Configuring a Naming Service Type"](#) on page 91-7.
4. In the JNDI Naming Service area, click **Properties**. The Edit Properties dialog box appears.

Figure 91-9 Edit Properties Dialog Box



5. Click **Add** to create a new property. The Add New Property dialog box appears.
To change (or delete) an existing property, select the property and click **Edit** (or **Remove**).

Use this table to enter data in the following fields on the dialog box.

Field	Description
Name	The name of the property.
Value	The value of the property.

Configuring a Packet Time-to-Live

The **packet time-to-live** is the number of hops that session data **packets** can take before expiring. The default is 2. This allows for a **hub** and an interface card, and prevents the data packets from leaving the local network. If sessions are hosted on different local area networks (LANs) that are part of wide area network (WAN), or if a firewall configuration prevents it, the announcement sent by one session may not reach the other sessions in the coordinated cache. In this case, consult your network administrator for the correct time-to-live value.

[Table 91-13](#) summarizes which coordinated caches support packet time-to-live configuration.

Table 91-13 Coordinated Cache Support for Packet Time-to-Live Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
JMS Coordinated Cache		
RMI Coordinated Cache	✓	✓

Table 91-13 (Cont.) Coordinated Cache Support for Packet Time-to-Live Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
CORBA Coordinated Cache	✓	✓
Custom Coordinated Cache		

In addition to configuring packet time-to-live, you may also need to configure announcement delay (see ["Configuring an Announcement Delay"](#) on page 91-11).

Using TopLink Workbench

To specify the number of hops that session data packets can take before expiring, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor.
2. Ensure the **Enable Cache Coordination** option is selected, then select the appropriate coordinated cache **Type** (see [Table 91-12](#)). The cache coordination options appear on the tab.

Figure 91-10 Cache Coordination Tab, Packet Time to Live Field

The screenshot shows the 'Cache Coordination' tab in the TopLink Workbench. The 'Enable Cache Coordination' checkbox is checked. The 'Type' dropdown is set to 'RMI'. The 'Channel' text box contains 'TopLinkCommandChannel'. The 'Multicast Group Address' text box contains '226.10.12.64'. The 'Multicast Port' spinner is set to '3,121'. The 'Packet Time to Live' spinner is set to '2' and is circled in red. The 'Announcement Delay' spinner is set to '1,000'. Below these are checkboxes for 'Remove Connection on Error' (checked) and 'Synchronous' (checked). There are two sections for naming services: 'Registry Naming Service' (disabled) and 'JNDI Naming Service' (selected). The 'JNDI Naming Service' section has fields for 'URL', 'User Name' (admin), 'Password' (password), and 'Initial Context Factory' (com.evermind.server.rmi.RMIInitialContextFactory). There are 'Properties...' and 'Browse...' buttons at the bottom.

In the **Packet Time to Live** field, specify the number of hops (default = 2) that session data packets can take before expiring.

Configuring a JMS Coordinated Cache

This chapter describes the various components that you must configure in order to use a JMS coordinated cache.

JMS Coordinated Cache Configuration Overview

Table 92–1 lists the configurable options for a JMS coordinated cache.

Table 92–1 Configurable Options for a JMS Coordinated Cache

Option	Type	TopLink Workbench	Java
"Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 28-38	Basic	✓	✓
"Configuring the Synchronous Change Propagation Mode" on page 91-2	Basic	✓	✓
"Configuring JNDI Naming Service Information" on page 91-7	Basic	✓	✓
"Configuring a Topic Name" on page 92-1	Basic	✓	✓
"Configuring a Topic Connection Factory Name" on page 92-2	Basic	✓	✓
"Configuring a Topic Host URL" on page 92-2	Advanced	✓	✓
"Configuring Connection Handling" on page 91-13	Advanced	✓	✓
"Configuring Context Properties" on page 91-14	Advanced	✓	✓
"Configuring a Packet Time-to-Live" on page 91-15	Advanced		✓

Configuring a Topic Name

A JMS topic identifies a publish/subscribe destination for a JMS server. JMS users who wish to share messages subscribe to the same JMS topic.

The topic name you configure is the name that TopLink uses to look up the `javax.jms.Topic` instance from the JNDI service. You must provide a fully qualified JNDI name, such as `jms/<topic_name>`.

All the members of the same JMS coordinated cache must use the same JMS topic.

Using TopLink Workbench

To specify the topic name for JMS cache coordination, use this procedure:

1. Select a server session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.

3. Ensure **Enable Cache Coordination** is selected and the **Type** is **JMS** (see ["Understanding Cache Coordination"](#) on page 90-10 for more information).

Figure 92–1 Cache Coordination Tab, Topic Name Field, JMS

4. Enter the topic name to use with the JMS coordinated cache for this session. This must be a fully qualified JNDI name, such as `jms/<topic_name>`.

Configuring a Topic Connection Factory Name

A JMS topic connection factory creates connections with the JMS provider for a specific JMS destination. Each connection factory contains the specific configuration information to create a connection to a JMS destination.

The topic connection factory name you configure is the name that TopLink uses to look up the `javax.jms.TopicConnectionFactory` instance from the JNDI service. This must be a fully qualified JNDI name, such as `jms/<resource_name>`.

Using TopLink Workbench

To specify the topic connection factory for a JMS coordinated cache, use this procedure:

1. Select a server session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.
3. Ensure **Enable Cache Coordination** is selected and the **Type** is **JMS** (see ["Understanding Cache Coordination"](#) on page 90-10 for more information).

Figure 92-2 Cache Coordination Tab, Topic Connection Factory Name Field

The screenshot shows the 'Cache Coordination' tab of a configuration window. The 'Enable Cache Coordination' checkbox is checked. The 'Type' is set to 'JMS'. The 'Topic Name' is 'jms/TopLinkTopic'. The 'Topic Connection Factory Name' is 'jms/TopLinkTopicConnectionFactory' and is circled in red. The 'Topic Host URL' is empty. The 'Remove Connection on Error' checkbox is checked. Below, the 'JNDI Naming Service' section has fields for 'URL', 'User Name' (admin), 'Password' (password), and 'Initial Context Factory' (com.evermind.server.rmi.RMIInitialContextFactory). There are 'Properties...' and 'Browse...' buttons.

4. Enter the topic connection factory name to use with the JMS coordinated cache for this session. This must be a fully qualified JNDI name, such as `jms/<resource_name>`.

Configuring a Topic Host URL

The JMS topic host URL is the URL of the machine on the network that hosts the JMS topic (see ["Configuring a Topic Name"](#) on page 92-1).

Using TopLink Workbench

To specify the topic host URL for a JMS coordinated cache, use this procedure:

1. Select a server session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.
3. Ensure **Enable Cache Coordination** is selected and the **Type** is **JMS** (see ["Understanding Cache Coordination"](#) on page 90-10 for more information).

Figure 92-3 Cache Coordination Tab, Topic Host URL Field

The screenshot shows a configuration window with several tabs: General, Options, Login, Logging, Cache Coordination (selected), and Connection Policy. In the Cache Coordination tab, the 'Enable Cache Coordination' checkbox is checked. Below it, the 'Type' is set to 'JMS'. The 'Topic Name' is 'jms/TopLinkTopic', and the 'Topic Connection Factory Name' is 'jms/TopLinkTopicConnectionFactory'. The 'Topic Host URL' field is empty and highlighted with a red circle. Below this, the 'Remove Connection on Error' checkbox is checked. A section titled 'JNDI Naming Service' contains fields for 'URL', 'User Name' (set to 'admin'), and 'Password' (set to 'password'). The 'Initial Context Factory' is set to 'com.evermind.server.rmi.RMIInitialContextFactory', with a 'Browse...' button next to it. A 'Properties...' button is at the bottom left of the JNDI section.

Enter the URL of the machine on the network that hosts the JMS topic (see ["Configuring a Topic Name"](#) on page 92-1) to use with the JMS coordinated cache for this session.

Configuring an RMI Coordinated Cache

This chapter describes the various components that you must configure in order to use an RMI coordinated cache.

RMI Coordinated Cache Configuration Overview

Table 93–1 lists the configurable options for an RMI coordinated cache.

Table 93–1 Configurable Options for an RMI Coordinated Cache

Option	Type	TopLink Workbench	Java
"Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 28-38	Basic	✓	✓
"Configuring the Synchronous Change Propagation Mode" on page 91-2	Basic	✓	✓
"Configuring a Service Channel" on page 91-3	Basic	✓	✓
"Configuring a Multicast Group Address" on page 91-4	Basic	✓	✓
"Configuring a Multicast Port" on page 91-5	Basic	✓	✓
"Configuring a Naming Service Type" on page 91-7	Basic	✓	✓
"Configuring an Announcement Delay" on page 91-11	Basic	✓	✓
"Configuring Connection Handling" on page 91-13	Advanced	✓	✓
"Configuring Context Properties" on page 91-14	Advanced	✓	✓
"Configuring a Packet Time-to-Live" on page 91-15	Advanced	✓	✓

Configuring a CORBA Coordinated Cache

This chapter describes the various components that you must configure to use a CORBA coordinated cache.

CORBA Coordinated Cache Configuration Overview

Table 94–1 lists the configurable options for a CORBA coordinated cache.

Table 94–1 Configurable Options for a CORBA Coordinated Cache

Option	Type	TopLink Workbench	Java
"Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 28-38	Basic	✓	✓
"Configuring the Synchronous Change Propagation Mode" on page 91-2	Basic	✓	✓
"Configuring a Service Channel" on page 91-3	Basic	✓	✓
"Configuring a Multicast Group Address" on page 91-4	Basic	✓	✓
"Configuring a Multicast Port" on page 91-5	Basic	✓	✓
"Configuring a Naming Service Type" on page 91-7	Basic	✓	✓
"Configuring an Announcement Delay" on page 91-11	Basic	✓	✓
"Configuring Connection Handling" on page 91-13	Advanced	✓	✓
"Configuring Context Properties" on page 91-14	Advanced	✓	✓
"Configuring a Packet Time-to-Live" on page 91-15	Advanced		✓

Configuring a Custom Coordinated Cache

This chapter describes the various components that you must configure to use a custom, user-defined coordinated cache. For more information, see "[Custom Coordinated Cache](#)" on page 90-13.

Custom Coordinated Cache Configuration Overview

[Table 95-1](#) lists the configurable options for a custom coordinated cache.

Table 95-1 Configurable Options for a Custom Coordinated Cache

Option	Type	Using TopLink Workbench	Java
"Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 28-38	Basic	✓	✓
"Configuring a Service Channel" on page 91-3	Basic	✓	✓
"Configuring Transport Class" on page 95-1	Basic	✓	✓
"Configuring Connection Handling" on page 91-13	Advanced	✓	✓

Configuring Transport Class

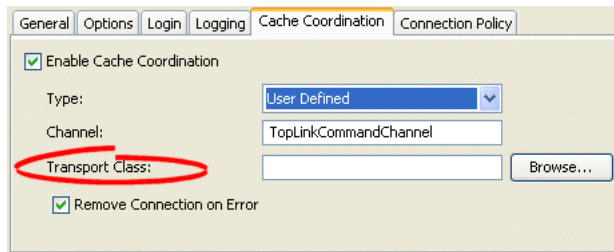
To configure a custom coordinated cache, you must specify your custom instance of `oracle.toplink.remotecommand.TransportManager`.

Using TopLink Workbench

To select the transport class for the user defined coordinated cache, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.
3. Ensure the **Enable Cache Coordination** option is selected and the **Type** is **User Defined** (see "[Understanding Cache Coordination](#)" on page 90-10).

Figure 95–1 Cache Coordination, Transport Class Option



4. Click **Browse** and select the transport class for the user-defined coordinated cache.

Part XIX

Queries

This part describes building TopLink queries and using them to create, read, update, and delete objects. It contains the following chapters.:

- [Chapter 96, "Understanding TopLink Queries"](#)

This chapter describes each of the different TopLink query types and important query concepts.

- [Chapter 98, "Using Basic Query API"](#)

This chapter explains how to use basic TopLink query options.

- [Chapter 99, "Using Advanced Query API"](#)

This chapter explains how to use advanced TopLink query options.

Understanding TopLink Queries

TopLink enables you to create, read, update, and delete persistent objects or data using queries in both J2EE and non-J2EE applications for both relational and nonrelational data sources.

This chapter explains the following:

- [Query Types](#)
- [Query Concepts](#)
- [Building Queries](#)
- [Executing Queries](#)
- [Handling Query Results](#)
- [Queries and the Cache](#)
- [Understanding the Query API](#)

Query Types

[Table 96–1](#) lists the query types that you can build in TopLink, and classifies them as basic or advanced.

Table 96–1 *TopLink Query Types*

Query Type	Description	Type	TopLink Workbench	Java
Session Queries	A query implicitly constructed and executed by a <code>Session</code> based on input parameters used to perform the most common data source actions on objects.	Basic		✓
Database Queries	A query also known as a <i>query object query</i> . An instance of <code>DatabaseQuery</code> that you create and then execute to perform any data source action on either objects or data. You can further refine a <code>DatabaseQuery</code> by also creating and configuring its <code>Call</code> (see "Call Queries" on page 96-17).	Basic		✓
Named Queries	An instance of <code>DatabaseQuery</code> stored by name in a <code>Session</code> or a descriptor's <code>DescriptorQueryManager</code> where it is constructed and prepared once. Such a query can then be repeatedly executed by name.	Basic	✓	✓
Call Queries	An instance of <code>Call</code> that you create and then either execute directly, using a special <code>Session</code> API to perform limited data source actions on data only, or execute indirectly in the context of a <code>DatabaseQuery</code> . TopLink supports <code>Call</code> instances for custom SQL, stored procedures, and EIS interactions.	Basic	✓	✓

Table 96-1 (Cont.) TopLink Query Types

Query Type	Description	Type	TopLink Workbench	Java
Redirect Queries	An instance of <code>MethodBasedQueryRedirector</code> (taking the name of a static method and the <code>Class</code> in which it is defined as parameters) set on a named query. When the query is executed, the static method is invoked.	Advanced		✓
Historical Queries	Any query executed in the context of a historical session using the time-aware features of the TopLink Expression framework.	Advanced		✓
Interface and Inheritance Queries	Any query that references an interface type or super and subclasses of an inheritance hierarchy.	Advanced		✓
Descriptor Query Manager Queries	The <code>DescriptorQueryManager</code> defines a default <code>DatabaseQuery</code> for each basic data source operation (create, read, update, and delete), and provides an API with which you can customize either the <code>DatabaseQuery</code> or its <code>Call</code> .	Advanced	✓	✓
EJB Finders	A query defined on the home interface of an EJB that returns EJB. You can implement finders using any TopLink query type, including <code>EJBQLCall</code> , a <code>Call</code> that takes EJB QL.	Advanced	✓	✓

For more information, see the following:

- ["Using Basic Query API"](#) on page 98-1
- ["Using Advanced Query API"](#) on page 99-1
- ["Query Concepts"](#) on page 96-2

Query Concepts

In general, querying a data source means performing an action on or interacting with the contents of the data source. To do this, you must be able to perform the following:

- Define an action in a syntax native to the data source being queried
- Apply the action in a controlled fashion
- Manage the results returned by the action (if any)

Specific to TopLink, you must also consider how the query affects the TopLink cache. For more information, see ["Queries and the Cache"](#) on page 96-30.

This section introduces query concepts unique to TopLink, including the following:

- [Call](#)
- [DatabaseQuery](#)
- [Data-Level and Object-Level Queries](#)
- [Summary Queries](#)
- [Descriptor Query Manager](#)
- [TopLink Expressions](#)
- [Query Keys](#)
- [Query Languages](#)

Call

In TopLink, the `Call` object encapsulates an operation or action on a data source. TopLink provides a variety of `Call` types such as structured query language (SQL), Enterprise Java Beans Query Language (EJB QL), Extensible Markup Language (XML), and enterprise information system (EIS).

You can execute a `Call` directly or in the context of a `DatabaseQuery`.

DatabaseQuery

A `DatabaseQuery` object is an abstraction that associates additional customization and optimization options with the action encapsulated by a `Call`. By separating these options from the `Call`, TopLink can provide sophisticated query capabilities across all `Call` types.

For more information, see ["Database Queries"](#) on page 96-10.

Data-Level and Object-Level Queries

In TopLink, queries can be defined for objects or data.

- **Object-level** queries (see ["Object-Level Read Query"](#) on page 96-11 and ["Object-Level Modify Query"](#) on page 96-14) are object-specific and return data as objects in your domain model. They are the preferred type of query for mapped data. By far, object-level `DatabaseQuery` queries are the most common query used in TopLink.
- **Data-level** queries (see ["Data-Level Read Query"](#) on page 96-13 and ["Data-Level Modify Query"](#) on page 96-15) are used to query database tables directly, and are an appropriate way to work with unmapped data, such as foreign keys and object version fields.

Summary Queries

While data-level queries return raw data and object-level queries return objects in your domain model, summary queries return data about objects. TopLink provides partial object queries (see ["Partial Object Queries"](#) on page 96-11) to return a set of objects with only specific attributes populated, and report queries (see ["Report Query"](#) on page 96-15) to return summarized (or rolled-up) data for specific attributes of a set of objects.

Descriptor Query Manager

In addition to storing named queries applicable to a particular class (see ["Named Queries"](#) on page 96-16), you can also use the `DescriptorQueryManager` to override the default action that TopLink defines for common data source operations. For more information, see ["Descriptor Query Manager Queries"](#) on page 96-23.

TopLink Expressions

TopLink expressions let you specify query search criteria based on your domain object model. When you execute the query, TopLink translates these search criteria into the appropriate query language for your platform.

TopLink provides two public classes to support expressions:

- The `Expression` class represents an expression that can be anything from a simple constant to a complex clause with boolean logic. You can manipulate, group, and integrate expressions.
- The `ExpressionBuilder` class is the factory for constructing new expressions.

You can specify a selection criterion as an `Expression` with `DatabaseQuery` method `setSelectionCriteria` (see ["Database Queries"](#) on page 96-10), and in a finder that takes an `Expression` (see ["Expression Finders"](#) on page 96-28).

For more information about using TopLink expressions, see ["Understanding TopLink Expressions"](#) on page 97-1.

Query Keys

A query key is a schema-independent alias for a database field name. Using a query key, you can refer to a field using a schema-independent alias. In relational projects only, TopLink automatically creates query keys for all mapped attributes. The name of the query key is the name of the class attribute specified in your object model.

You can configure query keys in a class descriptor (see ["Configuring Query Keys"](#) on page 28-35) or interface descriptor (see ["Configuring Interface Query Keys"](#) on page 28-39).

You can use query keys in expressions (see ["Query Keys and Expressions"](#) on page 97-10) and to query variable one-to-one mappings (see ["Using Queries on Variable One-to-One Mappings"](#) on page 99-5).

Query Languages

Using TopLink, you can express a query using any of the following query languages:

- [SQL Queries](#)
- [EJB QL Queries](#)
- [XML Queries](#)
- [EIS Interactions](#)
- [Query by Example](#)

In most cases, you can compose a query directly in a given query language or, preferably, you can construct a `DatabaseQuery` with an appropriate `Call` and specify selection criteria using a TopLink `Expression`. Although composing a query directly appears to be the simplest approach (and for simple operations or operations on unmapped data, it is), using the `DatabaseQuery` approach offers the compelling advantage of confining your query to your domain object model and avoiding dependence on data source schema implementation details. Oracle recommends that you compose your queries using `DatabaseQuery`, `Call`, and `Expression`.

SQL Queries

SQL is the most common query language for applications that use a relational database data source.

You can execute custom SQL directly using `Session` methods `executeSelectingCall` and `executeNonSelectingCall`, or you can construct a `DatabaseQuery` with an appropriate `Call`.

TopLink provides a variety of SQL `Call` objects for use with stored procedures and, with Oracle databases, stored functions. For more information, see ["SQL Calls"](#) on page 96-17.

EJB QL Queries

Like SQL, EJB QL is a query language; but unlike SQL, it presents queries from an object model perspective, allowing you to declare queries using the attributes of each abstract entity bean in your object model. It also includes path expressions that enable navigation over the relationships defined between entity beans and dependent objects.

Using EJB QL offers the following advantages:

- You do not need to know the database structure (such as tables and fields).
- You can construct queries using the attributes of the entity beans instead of using database tables and fields.
- You can use relationships in a query to provide navigation from attribute to attribute.
- EJB QL queries are portable because they are database-independent.
- You can specify the reference class in the `SELECT` clause.

The disadvantage of EJB QL queries is that it is difficult to use when you construct complex queries.

TopLink provides the full support for the EJB QL specification.

Note: TopLink supports the `LOCATE` string function and will generate the correct SQL with it. However, not all data sources support `LOCATE`. Before using the `LOCATE` string function, consult your data source documentation.

EJB QL is the standard query language defined in the EJB 2.0 specification. Consequently, TopLink lets you specify selection criteria using EJB QL in an EJB finder (see ["EJB QL Finders"](#) on page 96-28).

Although EJB QL is usually associated with EJB, TopLink also lets you specify selection criteria using EJB QL in queries for regular Java objects as well. TopLink provides an EJB QL `Call` that you can execute directly or in the context of a `DatabaseQuery`. For more information, see ["EJB QL Calls"](#) on page 96-19 and ["DatabaseQuery"](#) on page 96-3.

XML Queries

You can use TopLink XML to query XML data stored in an Oracle Database XMLType field. For more information, see ["Direct to XMLType Mapping"](#) on page 36-5 and ["XMLType Functions"](#) on page 97-4.

EIS Interactions

When you execute a TopLink query using an EIS `Call` (see ["Enterprise Information System \(EIS\) Interactions"](#) on page 96-19), TopLink converts your selection criteria into an XML format appropriate for your J2C adapter.

If supported by your J2C adapter, you can use the XQuery language by executing an XQuery interaction (see ["XQueryInteraction"](#) on page 96-20) either directly or in the context of a `DatabaseQuery`.

Query by Example

Query by example is a simple and intuitive way to express a query. To specify a query by example, provide a sample instance of the persistent object to query, and set appropriate values on only the data members on which you wish to query.

You can use any valid constructor to create a sample instance or example object.

Query by example lets you query for an object based on any attribute that uses a direct mapping or a one-to-one relationship (including those with nesting).

Note: Query by example does not support any other relationship mapping types nor does it support EJB 2.0 beans.

Set only the attributes on which you base the query; set all other attributes to `null`. By default, TopLink ignores attributes in the sample instance that contain `null`, zero (0), empty strings, and `FALSE`. You can modify this list of values (and define other query by example options) by specifying a `QueryByExamplePolicy` (see ["Defining a QueryByExamplePolicy"](#) on page 98-7).

Query by example uses the AND operator to tie the attribute comparisons together. By default, attribute values in the sample instance are compared against corresponding values of candidate objects using EQUALS operator. You can modify this behaviour using the `QueryByExamplePolicy`.

Both `ReadAllQuery` and `ReadObjectQuery` provide a `setExampleObject` method and `setQueryByExamplePolicy` method that you can use to specify selection criteria based on an example object instance.

For more information, see ["Reading Objects Using Query By Example"](#) on page 98-6.

Building Queries

You can build queries using TopLink Workbench or Java, using the TopLink API.

Some queries are implicitly constructed for you based on passed in arguments and executed in one step (for example, session queries, as described in ["Session Queries"](#) on page 96-9) and others you explicitly create, configure, and then execute (for example, ["Database Queries"](#) on page 96-10).

For more information, see the following:

- ["Using Basic Query API"](#) on page 98-1
- ["Using Advanced Query API"](#) on page 99-1

Executing Queries

In TopLink, you execute most queries using the `Session` API summarized in [Table 96-2](#).

Table 96-2 Session Methods for Executing a Query

Query Type	Session Method	Advantages and Disadvantages
Session Queries	<code>readObject</code>	Advantages: the most convenient way to perform common data source operations on objects.
	<code>readAllObjects</code>	
	<code>writeObject</code>	Disadvantages: less control over query execution and results; less efficient for frequently executed queries.
	<code>writeAllObjects</code>	
	<code>deleteObject</code>	
	<code>deleteAllObjects</code>	
	<code>insertObject</code>	
	<code>updateObject</code>	
Database Queries Named Queries Redirect Queries	<code>executeQuery</code>	Advantages: greatest configuration and execution flexibility; can take advantage of named queries for efficiency. Disadvantages: you must explicitly create and configure <code>DatabaseQuery</code> and possibly <code>Call</code> objects.
Call Queries	<code>executeSelectingCall</code> <code>executeNonSelectingCall</code>	Advantages: convenient way to directly apply an action to unmapped data. Disadvantages: least control over query execution and results; your application must do more work to handle raw data results.

Note: Oracle recommends that you perform all data source operations using a unit of work: doing so is the most efficient way to manage transactions, concurrency, and referential constraints. For more information, see ["Understanding TopLink Transactions"](#) on page 100-1.

Alternatively, you can execute queries outside of a unit of work using a session API directly, but doing so places greater responsibility on your application to manage transactions, concurrency, and referential constraints.

TopLink executes `DescriptorQueryManager` queries when you execute a session query. For more information, see ["Descriptor Query Manager Queries"](#) on page 96-23.

You execute EJB finders when you call the appropriate finder method on an EJB. For more information, see ["EJB Finders"](#) on page 96-24.

WARNING: Allowing an unverified SQL string to be passed into methods (for example: `setSQLString(String sql)`, `readAllObjects(Class class, String sql)` methods) makes your application vulnerable to SQL injection attacks.

For more information, see the following:

- ["Using Basic Query API"](#) on page 98-1
- ["Using Advanced Query API"](#) on page 99-1

Handling Query Results

TopLink queries generally return Java objects as their result set. TopLink queries can return any of the following:

- Entire objects, with all attributes populated and the object reflected in the cache.
- Collections of objects (see ["Collection Query Results"](#) on page 96-8).
- Partial objects, with only the attributes you specify populated and without the object reflected in the cache (see ["Report Query Results"](#)).
- Streams of objects (see ["Stream and Cursor Query Results"](#) on page 96-8).
- EJB (see ["EJB Finders"](#) on page 96-24).

Collection Query Results

A collection is a group of Java objects contained by an instance of `Collection` or `Map`.

By default, queries that return more than one object return their results in a `Vector`.

You can configure TopLink to return query results in any concrete instance of `Collection` or `Map`.

Collection results are supported by all TopLink query types.

For information and examples on how to configure and handle collection query results, see ["Handling Collection Query Results"](#) on page 98-25.

Report Query Results

A `ReportQuery` (a type of partial object query) returns summary data for selected objects using the database reporting functions and features supported by your platform. Although the report query returns data (not objects), it does enable you to query the returned data and specify it at the object level.

By default, a `ReportQuery` returns a collection (see ["Collection Query Results"](#) on page 96-8) of `ReportQueryResult` objects, one collection per database row returned. You can use the `ReportQuery` API to configure how a `ReportQuery` returns its results. For more information see ["Handling Report Query Results"](#) on page 98-26.

For more information, see the following:

- ["Report Query"](#) on page 96-15
- ["Handling Report Query Results"](#) on page 98-26
- ["Partial Object Queries"](#) on page 96-11.

Stream and Cursor Query Results

A stream is a view of a collection, which can be a file, a device, or a `Vector`. A stream provides access to the collection, one element at a time in sequence. This makes it possible to implement stream classes in which the stream does not contain all the objects of a collection at the same time.

Large result sets can be resource-intensive to collect and process. To improve performance and give the client more control over the returned results, configure TopLink queries to use a cursor or stream.

Cursors & streams are supported by all subclasses of `DataReadQuery` and `ReadAllQuery`.

For more information, see ["Handling Cursor and Stream Query Results"](#) on page 99-17.

Session Queries

Sessions provide query methods that lets you perform the object operations listed in [Table 96–3](#).

Table 96–3 Session Object Query Summary

Session Type	Create	Read	Update	Delete
UnitOfWork	NA	readObject readAllObjects	NA	deleteObject deleteAllObjects
Server	NA	NA	NA	NA
ClientSession	NA	readObject readAllObjects	NA	NA
DatabaseSession	insertObject	readObject readAllObjects	updateObject writeObject writeAllObjects	deleteObject deleteAllObjects

Note: Oracle recommends that you perform all data source operations using a unit of work: doing so is the most efficient way to manage transactions, concurrency, and referential constraints. For more information, see ["Understanding TopLink Transactions"](#) on page 100-1.

These methods implicitly construct and execute a `DatabaseQuery` based on any of the following input parameters and return `Object` or `Object` collection:

- Reference Class (the Class of objects that the query accesses)
- Reference Class and Call
- Reference Class and Expression
- Example object with primary key set

These methods are a convenient way to perform the most common data source operations on objects.

WARNING: Allowing an unverified SQL string to be passed into these methods makes your application vulnerable to SQL injection attacks.

To access all configurable options to further refine and optimize a query, consider using a corresponding `DatabaseQuery` directly. For more information, see ["Database Queries"](#) on page 96-10.

For more information, see ["Using Session Queries"](#) on page 98-1.

Read-Object Session Queries

Read-object queries return the first instance of an `Object` that matches the specified selection criteria, and read-all object queries return all such instances.

You can also pass in a domain `Object` with its primary key set and `TopLink` will construct and execute a read-object query to select that object. This is one form of query by example. For more information on query by example, see ["Query by Example"](#) on page 96-6.

For more information, see ["Reading Objects With a Session Query"](#) on page 98-1.

Create, Update, and Delete Object Session Queries

Oracle recommends that you create and update objects using a unit of work: doing so is the most efficient way to manage transactions, concurrency, and referential constraints. For more information, see ["Understanding TopLink Transactions"](#) on page 100-1.

However, you can also create and update objects using a session query. These session queries are a convenient way to modify objects directly on the database when you manage simple, nonbusiness object data that has no relationships (for example, user preferences).

If you know an object is new, you can use an `insertObject` method to avoid having `TopLink` perform an existence check. If you do not know if an object is new, use the `updateObject`, `writeObject`, or `writeAllObject` methods: `TopLink` performs an existence check if necessary.

When you execute a write session query, it writes both the object and its privately owned parts to the database. To manage this behavior, use a corresponding `DatabaseQuery` (see ["Object-Level Modify Queries and Privately Owned Parts"](#) on page 96-15).

Using the `Session` method `deleteObject`, you can delete a specific object. Using the `Session` method `deleteAllObjects`, you can delete a collection of objects. Each specified object and all its privately owned parts are deleted. In the case of `deleteAllObjects`, all deletions are performed within a single transaction.

For more information, see ["Creating, Updating, and Deleting Objects With a Session Query"](#) on page 98-3.

Database Queries

All session types provide an `executeQuery` method that takes any of the following types of `DatabaseQuery`:

- [Object-Level Read Query](#)
- [Data-Level Read Query](#)
- [Object-Level Modify Query](#)
- [Data-Level Modify Query](#)
- [Report Query](#)

Using `DatabaseQuery` method `setCall`, you can define your own `Call` to accommodate a variety of data source options such as SQL (including stored procedures and stored functions), EJB QL queries, and EIS interactions. For more information, see ["Call Queries"](#) on page 96-17.

Using `DatabaseQuery` method `setSelectionCriteria`, you can specify your selection criteria using a `TopLink Expression`. For more information, see ["TopLink Expressions"](#) on page 96-3.

For more information, see ["Using DatabaseQuery Queries"](#) on page 98-4.

Object-Level Read Query

Using an `ObjectLevelReadQuery`, you can query your data source and return `Object` instances that match the specified selection criteria. This section describes the following:

- [ReadObjectQuery](#)
- [ReadAllQuery](#)
- [Partial Object Queries](#)
- [Join Reading and Object-Level Read Queries](#)
- [Fetch Groups and Object-Level Read Queries](#)

For more information, see "[Reading Objects Using a DatabaseQuery](#)" on page 98-4.

ReadObjectQuery

Using a `ReadObjectQuery`, you can query your data source and return the first object that matches the specified selection criteria.

ReadAllQuery

Using a `ReadAllQuery`, you can query your data source and return a `Collection` of all the objects that match the specified selection criteria.

Partial Object Queries

By default, an `ObjectLevelReadQuery` returns all attributes of the objects read.

If you require only certain attributes from selected objects, you can create a partial object query by using `ObjectLevelReadQuery` method `addPartialAttributes`. Using this method, you can improve query performance by making `TopLink` return objects with only specified attributes populated.

Applications frequently use partial object queries to compile a list for further selection. For example, a query to find the names and addresses of all employees over the age of 40 returns a list of data (the names and addresses) that partially represents objects (the employees). A common next step is to present this list so the user can select the required object or objects from the list. Later retrieval of a complete object is simplified because `TopLink` always includes the primary key attribute (even if you do not add it as a partial attribute).

Consider the following when you use partial object queries:

- You cannot edit or cache partial objects.
- Unspecified attributes will be left `null`.
- You cannot have two partial attributes of the same type.
- You cannot add a partial attribute which is of the same type as the class being queried.

If you require only summary information for certain attributes from selected objects, it is more efficient to use a `ReportQuery` (see "[Report Query](#)" on page 96-15).

For more information, see "[Reading Objects Using Partial Object Queries](#)" on page 98-5.

Join Reading and Object-Level Read Queries

Join reading is a query optimization feature that allows a single query for a class to return the data to build the instances of that class and its related objects. Use this feature to improve query performance by reducing database access. By default, relationships are not join-read: each relationship is fetched separately when accessed if you are using indirection (see ["Indirection"](#) on page 33-5), or as a separate database query if you are not using indirection.

You can use join reading with `ReadObjectQuery` and `ReadAllObjectQuery` to join only one-to-one or one-to-many mapped relationships. Join reading is not currently supported for any other relationship mappings.

Join reading can specify multiple and nested relationships to be joined. Nested joins are expressed through using expressions (see ["Expressions for Joining and Complex Relationships"](#) on page 97-5).

Outer joins can also be used with join reading through using the expression outer join API. If an outer join is not used, objects with missing one-to-one relationships or empty one-to-many relationships will be filtered from the result set.

You can use join reading between relationships to the same class, or to concrete or leaf inherited classes, but not to root or branch inherited classes that have multiple tables. For more information about inheritance, see ["Understanding Descriptors and Inheritance"](#) on page 26-12.

Note: TopLink does not support joining to a class that takes multiple SQL calls to read. For example, joining to a superclass (potentially).

You can use join reading with custom SQL or stored procedures, but the query must ensure that all of the required data to build all of the join-read objects is returned. If the result set includes the same tables or fields, they must be returned in the same table order as TopLink would have generated.

For more information, see ["Using Join Reading"](#) on page 98-11.

Avoiding Join-Reading Duplicate Data Join reading can result in returning duplicate data if a one-to-many or a shared one-to-one relationship is joined. Although TopLink correctly filters the duplicate results from the object result, the duplicate data still must be fetched from the database and can degrade performance, especially if multiple one-to-many relationships are joined. In general, batch reading can be used as a better alternative to join reading, as it does not require fetching duplicate data.

Oracle recommends that you use one-to-many joining with caution, because it does not scale well in many situations.

Because the main cost of a `ReadObjectQuery` is SQL execution, the performance of a one-to-many join in this case is usually better than a query without joining.

However, because the main cost of a `ReadAllObjectQuery` is row-fetching, which the duplicate data of a join increases, the performance of a one-to-many join in this case is less efficient than batch reading in many scenarios (even though one-to-many joining is more efficient than reading the objects one-by-one).

This is mainly due to the fact that a one-to-many join reads in duplicate data: the data for each source object will be duplicated for each target object. Depending on the size of the one-to-many relationship and the size of the source object's row, this can become very inefficient, especially if the source object has a Large Object (LOB).

If you use multiple or nested one-to-many joins in the same query, the problem is compounded: the source object's row is duplicated $n*m$ times, and each target object n and m times respectively. This can become a major performance issue.

To handle empty collections, you must use outer joins, so the queries can easily become very database intensive. Batch reading has the advantage of only returning the required data, and does not require outer joins.

Oracle recommends that you use batch reading to optimize querying relationships in read-all applications.

For more information, see the following:

- ["Batch and Join Reading"](#) on page 11-19
- ["Reading Case 2: Batch Reading Objects"](#) on page 11-24

Fetch Groups and Object-Level Read Queries

You can use a fetch group with a `ReadObjectQuery` or `ReadAllQuery`. When you execute the query, `TopLink` retrieves only the attributes in the fetch group. `TopLink` automatically executes a query to fetch all the attributes excluded from this subset when and if you call a getter method on any one of the excluded attributes.

Note: Currently, fetch groups only provide support for EJB.

For more information, see the following:

- ["Fetch Groups"](#) on page 26-4
- ["Using Queries With Fetch Groups"](#) on page 99-2

Data-Level Read Query

Using a `DataLevelReadQuery`, you can query your data source and return `Object` instances that match the specified selection criteria. This section describes the following:

- [DataReadQuery](#)
- [DirectReadQuery](#)
- [ValueReadQuery](#)

For more information, see ["Reading Data With a DatabaseQuery"](#) on page 98-14.

WARNING: Allowing an unverified SQL string to be passed into constructors of such objects as `DataReadQuery`, `DirectReadQuery` and `ValueReadQuery` makes your application vulnerable to SQL injection attacks.

DataReadQuery

Use a `DataReadQuery` to execute a selecting SQL string that returns a `Collection` of the `DatabaseRows` representing the result set.

DirectReadQuery

Use a `DirectReadQuery` to read a single column of data (that is, one field) that returns a `Collection` of values representing the result set.

ValueReadQuery

Use a `ValueReadQuery` to read a single data value (that is, one field). A single data value is returned, or null if no rows are returned.

Object-Level Modify Query

Using an `ObjectLevelModifyQuery`, you can query your data source to create, update, and delete objects. This section describes the following:

- [WriteObjectQuery](#)
- [UpdateObjectQuery](#)
- [InsertObjectQuery](#)
- [DeleteObjectQuery](#)
- [UpdateAllQuery](#)
- [Object-Level Modify Queries and Privately Owned Parts](#)

For more information, see ["Creating, Updating, and Deleting Objects With a DatabaseQuery"](#) on page 98-12.

Note: Oracle recommends that you create and update objects using a `TopLink UnitOfWork`: doing so is the most efficient way to manage transactions, concurrency, and referential constraints. For more information, see ["Understanding TopLink Transactions"](#) on page 100-1.

WriteObjectQuery

If you do not know whether or not an object is new, use a `WriteObjectQuery`: `TopLink` performs an existence check if necessary to determine whether to perform an insert or an update.

If you do know whether or not an object exists, you can avoid the existence check by using an `UpdateObjectQuery` (see ["UpdateObjectQuery"](#) on page 96-14) or `InsertObjectQuery` (see ["InsertObjectQuery"](#) on page 96-14).

UpdateObjectQuery

If you know that the object you want to modify exists, use an `UpdateObjectQuery` to avoid having `TopLink` perform an existence check.

InsertObjectQuery

If you know an object is new, you can use an `InsertObjectQuery` to avoid having `TopLink` perform an existence check.

DeleteObjectQuery

To delete a specific object, construct a `DeleteObjectQuery` with a single specific object as an argument.

UpdateAllQuery

The `UpdateAllQuery` allows you to take an expression and update a set of objects (at the object level) without loading the objects into memory. You can updated to either a

specific or relative value. For example, you can set the value to 5 or to increase by 5 percent.

Object-Level Modify Queries and Privately Owned Parts

When you execute a create or update object `DatabaseQuery`, it writes both the object and its privately owned parts to the database by default. To create a query that does not update privately owned parts, use the `DatabaseQuery` method `dontCascadeParts`. Use this method to do the following:

- Increase performance when you know that only the object's direct attributes have changed.
- Manually resolve referential integrity dependencies when you write large groups of new, independent objects.

Note: Because the unit of work resolves referential integrity internally, this method is not required if you use the unit of work to write to the data source. For more information, see "[Understanding TopLink Transactions](#)" on page 100-1.

Data-Level Modify Query

Using a `DataModifyQuery`, you can query your data source to execute a nonselecting SQL statement. It is equivalent to `Session` method `executeNonSelectingCall`.

For more information, see "[Updating Data With a DatabaseQuery](#)" on page 98-15.

Report Query

If you want to summarize (or roll up) certain attributes of a set of objects, you can use a `ReportQuery`.

A `ReportQuery` returns summary data from a set of objects and their related objects. That is, it returns data about objects, rather than the objects themselves. However, it still lets you query and specify the data at the object level. To build a report query, you specify the search criteria, the data you require about the objects, and how that data should be summarized.

For example, you can create a report query to compute the average age of all employees in your company. The report query is not interested in the specific objects (the employees), but rather, summary information about them (their average age).

A `ReportQuery` lets you do the following:

- Specify a subset of the object's attributes and its related object's attributes, which allows you to query for lightweight information.
- Build complex object-level expressions for the selection criteria and ordering criteria.
- Use data source aggregation functions (supported by your platform), such as SUM, MIN, MAX, AVG, and COUNT.
- Use expressions to group data.
- Request primary key attributes with each `ReportQueryResult`. This makes it easy to request the real object from a lightweight result.

A `ReportQuery` is the most efficient form of partial object query (see "[Partial Object Queries](#)" on page 96-11) because it takes advantage of the reporting capabilities of your

data source (if available). Oracle recommends that you use `ReportQuery` to do partial object queries.

The `ReportQuery` API returns a collection of `ReportQueryResult` objects, similar in structure and behavior to a `DatabaseRow` or a `Map`. For more information, see ["Report Query Results"](#) on page 96-8.

For more information, see the following:

- ["Reading Case 1: Displaying Names in a List"](#) on page 11-22
- ["Reading Objects Using Report Queries"](#) on page 98-6
- ["Configuring Named Queries at the Descriptor Level"](#) on page 28-11

Named Queries

When you use a session query method like `readAllObjects` (see ["Session Queries"](#) on page 96-9), TopLink creates a corresponding `ReadAllQuery`, which builds other objects it needs to perform its task. When TopLink finishes execution of the `readAllObjects` method, these objects are discarded. Each time you call this session method, TopLink creates these related objects again, uses them once, and then discards them.

Alternatively, you can create a `DatabaseQuery` (see ["Database Queries"](#) on page 96-10) and store it by name at the descriptor- (see ["Configuring Named Queries at the Descriptor Level"](#) on page 28-11) or session-level (see ["Configuring Named Queries at the Session Level"](#) on page 77-24).

TopLink prepares a named query once, and it (and all its associated supporting objects) can be efficiently reused thereafter making a named query well suited for frequently executed operations.

Using the `Session` API (see ["Using Named Queries"](#) on page 98-17), you can execute these queries by name, passing in any required arguments.

WARNING: Allowing an unverified SQL string to be passed into methods (for example: `readAllObjects(Class class, String sql)` method) makes your application vulnerable to SQL injection attacks.

When to Use Named Queries

For a reasonably complex query that you execute frequently, you should consider making the query a named query.

If a query is global to a project, configure the named query at the session level (["Configuring Named Queries at the Session Level"](#) on page 77-24).

If a query is global to a `Class` or you want to configure CMP finders, configure the named query at the descriptor level (see ["Configuring Named Queries at the Descriptor Level"](#) on page 28-11). For more information about descriptor level query configuration, see ["Descriptor Query Manager Queries"](#) on page 96-23.

For a very complex query, you can delegate query execution to your own static method using a special form of a named query called a redirect query. For more information about redirect queries, see ["Redirect Queries"](#) on page 96-20).

When Not to Use Named Queries

Rarely used queries may be more efficient when built on an as-needed basis. If you seldom use a given query, it may not be worthwhile to build and store that query when you invoke a session.

Call Queries

All session types provide `executeSelectingCall` and `executeNonSelectingCall` methods that take any of the following `Call` types:

- [SQL Calls](#)
- [EJB QL Calls](#)
- [Enterprise Information System \(EIS\) Interactions](#)

You can also execute a `Call` in the context of a `DatabaseQuery`. For more information on `DatabaseQuery`, see ["Database Queries"](#) on page 96-10.

WARNING: Allowing an unverified SQL string to be passed into methods (for example: `executeSelectingCall(String sql)` method) makes your application vulnerable to SQL injection attacks.

SQL Calls

SQL calls access fields in a relational database. TopLink supports the following SQL calls:

- [SQLCall](#)
- [StoredProcedureCall](#)
- [StoredFunctionCall](#)

Using the `Call` API (or SQL string conventions), you can specify input, output, and input-output parameters and assign values for input and input/output parameters.

Using a descriptor `ReturningPolicy`, you can control whether or not TopLink writes a parameter out, retrieves a value generated by the database, or both. For more information, see ["Configuring Returning Policy"](#) on page 28-76.

SQLCall

Using a `SQLCall`, you can specify any arbitrary SQL statement and execute it on a data source.

WARNING: Allowing an unverified SQL string to be passed into methods makes your application vulnerable to SQL injection attacks.

For more information, see ["Using an SQLCall"](#) on page 98-18.

StoredProcedureCall

A stored procedure is composed of one or more procedural language statements, such as Procedural Language/Structured Query Language (PLSQL), stored by name in the database. Most relational databases support stored procedures.

You invoke a stored procedure to execute logic and access data from the data source.

Using a `StoredProcedureCall`, you can detect execution errors, specify input parameters, output parameters, and input/output parameters. However, stored procedures do not provide a return value.

For more information, see ["Using a StoredProcedureCall"](#) on page 98-21.

StoredFunctionCall

A stored function is an Oracle database feature that provides all the functionality of a stored procedure as well as the ability to return a value.

Using a `StoredFunctionCall`, you can specify all the features of a `StoredProcedureCall` as well as the field name of the return value.

For more information, see ["Using a StoredFunctionCall"](#) on page 98-23.

Oracle Extensions

When you use TopLink with an Oracle database, you can make use of the following Oracle specific query features from within your TopLink applications:

- [Hints](#)
- [Hierarchical Queries](#)
- [Flashback Queries](#)
- [Stored Functions](#)

Hints Oracle lets you specify SQL query additions called hints that can influence how the database server SQL optimizer works. This lets you influence decisions usually reserved for the optimizer. You use hints to specify things such as join order for a join statement, or the optimization approach for a SQL call.

You specify hints using the `DatabaseQuery` method `setHintString`.

For more information, see the following:

- ["Database Queries"](#) on page 96-10
- ["Oracle Hints"](#) on page 99-6
- Your database *Performance Tuning Guide and Reference*.

Hierarchical Queries Oracle database Hierarchical Queries mechanism lets you select database rows based on hierarchical order. For example, you can design a query that reads the row of a given employee, followed by the rows of people the employee manages, followed by their managed employees, and so on.

You specify a hierarchical query clause using `DatabaseQuery` subclass `ReadAllQuery` method `setHierarchicalQueryClause`. For more information on `DatabaseQuery` queries, see ["Database Queries"](#) on page 96-10.

For more information on configuring a `ReadAllQuery` with an Oracle hierarchical query clause, see ["Hierarchical Queries"](#) on page 99-6.

Flashback Queries When using TopLink with Oracle9i (or later), you can acquire a special historical session where all objects are read as of a past time, and then you can express read queries depending on how your objects are changing over time.

For more information, see ["Historical Queries"](#) on page 96-21.

Stored Functions A stored function is an Oracle database mechanism that provides all the capabilities of a stored procedure in addition to returning a value.

For more information, see ["StoredFunctionCall"](#) on page 96-18.

EJB QL Calls

In TopLink, EJB QL calls represent EJB QL strings. An `EJBQLCall` object is an abstraction of a database invocation. You can execute an EJB QL call directly from a session or in the context of a `DatabaseQuery`.

For more information, see the following:

- ["Using EJB QL Calls"](#) on page 98-24
- ["Specifying a Custom EJB QL String in a DatabaseQuery"](#) on page 98-16
- ["EJB QL Queries"](#) on page 96-5

Enterprise Information System (EIS) Interactions

To invoke a query through a J2EE Connector Architecture (J2C) adapter to a remote EIS, you use an `EISInteraction`, an instance of `Call`. TopLink supports the following `EISInteraction` types:

- [IndexedInteraction](#)
- [MappedInteraction](#)
- [XMLInteraction](#)
- [XQueryInteraction](#)
- [QueryStringInteraction](#)

In each of these interactions, you specify a functional interface (similar to a stored procedure) that identifies the function to invoke on the EIS. This functional interface contains the following:

- The function name
- The record name (if different than the function name)
- A list of input arguments
- A list of output arguments

For more information, see

- ["EIS Projects"](#) on page 20-7
- ["Using EIS Interactions"](#) on page 98-24

IndexedInteraction

In an `IndexedInteraction`, you exchange data with the EIS using indexed records. The order of the specification of the arguments must match the order of the values defined in the indexed record.

MappedInteraction

In a `MappedInteraction`, you exchange data with the EIS using mapped records. The arguments you specify map by name to fields in the mapped record.

XMLInteraction

An `XMLInteraction` is a `MappedInteraction` that maps data to an XML record. For an `XMLInteraction`, you may also provide an optional root element name.

XQueryInteraction

If your J2C adapter supports the XQuery dynamic query language, you can use an `XQueryInteraction`, which is an `XMLInteraction` that lets you specify your XQuery string.

QueryStringInteraction

If your J2C adapter supports a query string based dynamic query language, you can use a `QueryStringInteraction`, which is a `MappedInteraction` that lets you specify the dynamic query string.

Redirect Queries

To accommodate complex query logic, you can implement a **redirect query**: a named query that delegates query execution control to your application. For more information, see "[Named Queries](#)" on page 96-16.

Redirect queries lets you define the query implementation in code as a static method. When you invoke the query, the call redirects to the specified static method. Redirect queries accept any arbitrary parameters passed into them packaged in a `Vector`.

Although most `TopLink` queries search for objects directly, a redirect query generally invokes a method that exists on another class and waits for the results. Redirect queries let you build and use complex operations, including operations that might not otherwise be possible within the query framework.

By delegating query invocation to a method you provide, redirect queries let you dynamically make decisions about how a query should be executed based on argument values.

Using a redirect query, you can do the following:

- Dynamically configure the query options based on the arguments (for example, ordering and query optimization)
- Dynamically define the selection criteria based on the arguments
- Pass query-by-example objects or expressions as the arguments
- Post-process the query results
- Perform multiple queries or special operations

If you execute the query on a `UnitOfWork`, the results register with that instance of `UnitOfWork`, so any objects you attempt to retrieve with the `invoke` method must come from the `Session` cache.

To create a redirect query, you implement the `QueryRedirector` interface and set your implementation on a named query.

Oracle recommends that you take advantage of the `MethodBasedQueryRedirector`, an instance of `QueryRedirector` that `TopLink` provides. It takes the name of a static method and the `Class` in which it is defined as parameters. When you set a `MethodBasedQueryRedirector` on a named query, whenever `invokeQuery` method is called on this instance, `TopLink` uses reflection to invoke your static method instead.

The advantages of using a `MethodBasedQueryRedirector` are as follows:

- You can specify the static method and its `Class` dynamically.
- The class that provides the static method does not need to implement `QueryRedirector`.
- Your static method can have any name.
- You can restrict the parameters to your static method to only a `Session` and a `Vector` of arguments.

For more information, see ["Using Redirect Queries"](#) on page 99-1.

Historical Queries

By default, a session represents a view of the most current version of objects and when you execute a query in that session, it returns the most current version of selected objects.

If your data source maintains past or historical versions of objects, you can configure `TopLink` to access this historical data (see ["Historical Client Sessions"](#) on page 75-25).

Once you configure `TopLink` to take advantage of this historical data, you can access historical versions using the historical queries that [Table 96-4](#) summarizes.

Note: Flashback queries do not support view selects. This means you cannot use a flashback query on objects with an inheritance policy for read-all-subclasses views. For more information, see ["Understanding Descriptors and Inheritance"](#) on page 26-12.

Table 96-4 Historical Queries

Historical Query Type	Session	Cache	Must set <code>maintainCache</code> to false?	Query both current and historical versions?
Using an <code>ObjectLevelReadQuery</code> With an <code>AsOfClause</code>	Regular ¹	■ Global ■ Read and write ■ Contains current versions	Yes	No
Using an <code>ObjectLevelReadQuery</code> With Expression Operator <code>asOf</code>	Regular ¹	■ Global ■ Read and write ■ Contains current versions	Yes	Yes
Using an <code>ObjectLevelReadQuery</code> in a Historical Session	Historical ²	■ Isolated ■ Read-only ■ Contains static snapshot as of specified time	No	No

¹ A server or database session based on an `OraclePlatform` for an `Oracle9i` (or later) or based on a `TopLinkHistoryPolicy`.

² A session returned by a server or database session based on an `OraclePlatform` or `TopLinkHistoryPolicy` using the `acquireHistoricalSession` method passing in an `AsOfClause`.

Using an `ObjectLevelReadQuery` With an `AsOfClause`

You can query historical versions of objects using an `ObjectLevelReadQuery` configured with an `AsOfClause` (set by `ObjectLevelReadQuery` method `setAsOfClause`) that specifies a point in time that applies to every `Expression` used in the query.

This type of historical query lets you query a static snapshot of object versions as of the specified time.

Note: To prevent corrupting the global shared cache with old versions of objects, you must set `ObjectLevelReadQuery` method `maintainCache` to `false` in this historical query. If you do not, `TopLink` will throw an exception when you execute the query.

For more information and examples of using an `ObjectLevelReadQuery` with an `AsOfClause`, see ["Using Historical Queries"](#) on page 99-2.

Using an `ObjectLevelReadQuery` With Expression Operator `asOf`

You can query historical versions of objects using an `ObjectLevelReadQuery` (such as `ReadObjectQuery` or `ReadAllQuery`) containing one or more expressions that use Expression operator `asOf` to specify a point in time on an Expression-by-Expression basis.

This type of historical query lets you combine both current and historical versions of objects in the same query.

If you configure the `ObjectLevelReadQuery` with an `AsOfClause`, that point in time overrides the point in time specified in any Expression in the query (see ["Using an ObjectLevelReadQuery With an AsOfClause"](#) on page 96-22).

Note: To prevent corrupting the global shared cache with old versions of objects, you must set `ObjectLevelReadQuery` method `maintainCache` to `false` in this historical query. If you do not, `TopLink` will throw an exception when you execute the query.

For more information and examples of using an `ObjectLevelReadQuery` with Expression operator `asOf`, see ["Using Historical Queries"](#) on page 99-2.

Using an `ObjectLevelReadQuery` in a Historical Session

Given a session that maintains historical versions of objects (based on an appropriate `OraclePlatform` or `TopLinkHistoryPolicy`), you can use `Session` method `acquireHistoricalSession` passing in an `AsOfClause` that specifies a point in time that applies to all queries and expressions.

This method returns a lightweight, read-only snapshot of object versions as of the specified time. The cache used in this type of session is isolated from the global shared cache. You do not need to set `ObjectLevelReadQuery` method `maintainCache` to `false` in this case.

For more information and examples of using an `ObjectLevelReadQuery` with a historical session, see ["Using Historical Queries"](#) on page 99-2.

Interface and Inheritance Queries

When you define an interface descriptor (see ["Relational Interface Descriptors"](#) on page 27-2), you can perform queries on interfaces and inheritance hierarchies.

For more information, see the following:

- ["Querying on Interfaces"](#) on page 99-4
- ["Querying on an Inheritance Hierarchy"](#) on page 99-4

Descriptor Query Manager Queries

Each `Descriptor` owns an instance of `DescriptorQueryManager` that you can use for the following:

- [Configuring Named Queries](#)
- [Configuring Default Query Implementations](#)
- [Configuring Additional Join Expressions](#)

Configuring Named Queries

The `DescriptorQueryManager` provides API for storing and retrieving frequently used queries by name.

For more information, see ["Named Queries"](#) on page 96-16.

Configuring Default Query Implementations

The `DescriptorQueryManager` of each `Descriptor` lets you customize the query implementation that TopLink uses for the following data source operations:

- insert object
- update object
- read object
- read all objects
- delete object

For example, if you need to insert an object using a stored procedure, you can override the default `SQLCall` used by the `DescriptorQueryManager` insert object query.

Whenever you execute a query on a given `Class`, TopLink consults the `DescriptorQueryManager` to determine how to perform the given data source operation.

You can use this capability for a variety of purposes such as to extend TopLink behavior, access nonrelational data, or use stored procedures or customized SQL calls.

WARNING: Allowing an unverified SQL string to be passed into methods makes your application vulnerable to SQL injection attacks.

For information and examples on customizing these default query implementations, see:

- ["Configuring Custom SQL Queries for Basic Persistence Operations"](#) on page 29-7
- ["Configuring Custom EIS Interactions for Basic Persistence Operations"](#) on page 31-6

Configuring Additional Join Expressions

You can configure the `DescriptorQueryManager` to automatically append an expression to every query it performs on a class. For example, you can add an expression that filters the data source for the valid instances of a given class.

For more information, see "[Appending Additional Join Expressions](#)" on page 99-4.

EJB Finders

An EJB finder is a query as defined by the EJB specification. It returns EJB, collections, and enumerations. The difference between a finder and a query is that queries return Java objects, but finders return EJB. The TopLink query framework lets you create and execute complex finders that retrieve entity beans.

Finders contain finder methods that define search criteria. The work involved in creating these methods depends on whether you are building container-managed persistence (CMP) bean finders or bean-managed persistence (BMP) bean finders:

- CMP finders require the developer to define the finder API method signature on the bean `Home` interface. The CMP provider generates the actual code mechanisms for the finder from the API definition.
- BMP finders require the developer to provide the code required to execute the finder methods.

In either case, you define finders in the `Home` interface of the bean.

You can implement finders using any TopLink query feature and you can take advantage of predefined finder implementations that TopLink provides for both CMP and BMP EJB.

This section describes the following:

- [Predefined Finders](#)
- [Default Finders](#)
- [Call Finders](#)
- [DatabaseQuery Finders](#)
- [Named Query Finders](#)
- [Primary Key Finders](#)
- [Expression Finders](#)
- [EJB QL Finders](#)
- [SQL Finders](#)
- [Redirect Finders](#)
- [The `ejbSelect` Method](#)

For more information, see "[Using EJB Finders](#)" on page 99-8.

Predefined Finders

TopLink provides predefined finder implementations that provide a rich API that lets you dynamically specify query properties at run time and take full advantage of TopLink query features.

TopLink provides the following predefined finders:

- [Predefined CMP Finders for EJB 2.0](#)
- [Predefined CMP Finders for EJB 1.1](#)
- [Predefined BMP Finders](#)

For more information, see the following:

- ["Understanding the EJB Entity Beans with CMP Architecture"](#) on page 2-27
- ["Understanding the EJB Entity Beans With BMP Architecture"](#) on page 2-34

Predefined CMP Finders for EJB 2.0

[Table 96–5](#) lists the predefined finders you can use with TopLink CMP for EJB 2.0 (using OC4J or BEA WebLogic Server).

The TopLink runtime reserves the method names listed in [Table 96–5](#).

Table 96–5 *Predefined CMP Finders for EJB2.0*

Method	Arguments	Return
findAll	()	Collection
findManyByEJBQL	(String ejbql) (String ejbql, Vector arguments)	Collection
findManyByQuery	(DatabaseQuery query) (DatabaseQuery query, Vector arguments)	Collection
findManyBySQL	(String sql) (String sql, Vector arguments)	Collection
findByPrimaryKey	(Object primaryKeyObject)	EJBObject
findOneByEJBQL	(String ejbql) (String ejbql, Vector arguments)	EJBObject
findOneByQuery	(DatabaseQuery query) (DatabaseQuery query, Vector arguments)	EJBObject
findOneBySQL	(String sql) (String sql, Vector arguments)	EJBObject

Note: With EJB 2.0, if the finder is located on a local home, replace EJBObject with EJBLocalObject in finders that contain findOneBy.

You can also use each of these finders without a vector of arguments. For example, EJBObject findOneByEJBQL(String ejbql) is a valid dynamic finder, but you must replace the return type of EJBObject with your bean's component interface.

For more information, see ["Using EJB Finders"](#) on page 99-8.

Predefined CMP Finders for EJB 1.1

[Table 96–6](#) lists the predefined finders you can use with TopLink CMP for EJB1.1 (using IBM WebSphere Application server). In [Table 96–6](#), finder methods with the suffix <name> can be declared with any name you choose provided the name has the given prefix and signature.

The TopLink runtime reserves the method names listed in [Table 96–6](#).

Table 96–6 *Predefined CMP Finders for EJB 1.1*

Method	Arguments	Return
findAll	()	Enumeration
findAll<name>	(Call) (Expression) (ReadAllQuery)	Enumeration
findAllByNamedQuery	(String queryName, Vector arguments)	Enumeration
findByPrimaryKey	(Object primaryKeyObject)	Object
findOne<name>	(Call) (Expression) (ReadObjectQuery)	Object
findOneByNamedQuery	(String queryName, Vector arguments)	Object

For more information and examples on using predefined CMP finders for EJB 1.1, see ["Using EJB Finders"](#) on page 99-8.

Predefined BMP Finders

[Table 96–7](#) lists the predefined finders you can use if you extend your BMP EJB from `oracle.toplink.ejb.bmp.BMPEntityBase` (see ["Understanding the EJB Entity Beans With BMP Architecture"](#) on page 2-34).

The TopLink runtime reserves the method names listed in [Table 96–7](#).

Table 96–7 *Predefined BMP Finders*

Method	Arguments	Return
findAll	() (Call) (Expression) (ReadAllQuery)	Enumeration
findAllByNamedQuery	(String queryName, Vector arguments)	Enumeration
findByPrimaryKey	(Object primaryKeyObject)	Object
findOne	(Call) (Expression) (ReadObjectQuery)	Object
findOneByNamedQuery	(String queryName, Vector arguments)	Object

For more information about using EJB finders, see ["Using EJB Finders"](#) on page 99-8.

Default Finders

For each finder method defined on the home interface of an entity bean, whose name matches `findBy<CMP-FIELD-NAME>` where `<CMP-FIELD-NAME>` is the name of a persistent field on the bean, TopLink generates a finder implementation including a TopLink query that uses the TopLink expressions framework. If the return type is a single bean type, TopLink creates a `ReadObjectQuery`; if the return type is an `Enumeration` (EJB 1.x) or `Collection` (EJB 2.x), TopLink creates a `ReadAllQuery`.

Although you must still define the finder in the entity home, you do not need to declare the finder in the `ejb-jar.xml` file.

For more information, see ["Creating a Finder"](#) on page 99-8.

Call Finders

Finders that use a `Call` lets you create dynamic queries that you generate at run time rather than at deployment time.

For more information, see the following:

- ["Call Queries"](#) on page 96-17.
- ["Predefined Finders"](#) on page 96-25

DatabaseQuery Finders

Finders that use a `DatabaseQuery` lets you create dynamic queries that you generate at run time rather than at deployment time.

In addition to finders that take a `DatabaseQuery`, TopLink also provides a default `findAll` finder that returns all the EJB of a given type. As with other dynamic finders, the TopLink runtime reserves the name `findAll`.

For more information, see ["Database Queries"](#) on page 96-10.

For more information on TopLink predefined finders that take a `DatabaseQuery`, see ["Predefined Finders"](#) on page 96-25.

Named Query Finders

Finders that use a named `DatabaseQuery` stored in a `DescriptorQueryManager` or `Session` lets you efficiently reuse frequently executed queries.

For more information, see

- ["Named Queries"](#) on page 96-16
- ["Predefined Finders"](#) on page 96-25

Primary Key Finders

TopLink provides predefined finder implementations that take a primary key class as a Java Object.

Because the EJB 2.0 specification requires the container to implement the `findByPrimaryKey` call on each bean `Home` interface, do not delete this finder from a bean.

For more information, see ["Predefined Finders"](#) on page 96-25.

Expression Finders

Using a finder based on a TopLink `Expression` offers the following advantages:

- Version-controlled standardized queries in Java code
- Ability to simplify most complex operations
- A more complete set of querying features than is available through EJB QL

Because expressions lets you specify finder search criteria based on the object model, they are frequently the best choice for constructing your finders.

For more information, see ["TopLink Expressions"](#) on page 96-3.

For more information on TopLink predefined finders that take an `Expression`, see ["Predefined Finders"](#) on page 96-25.

You can also use an `Expression` in a finder that takes a `DatabaseQuery` by using `DatabaseQuery` method `setSelectionCriteria`. For more information on TopLink predefined finders that take a `DatabaseQuery`, see ["DatabaseQuery Finders"](#) on page 96-27.

EJB QL Finders

TopLink supports EJB QL for both EJB 1.1 and EJB 2.0 beans. EJB QL finders lets you specify an EJB QL string as the implementation of the query.

EJB QL offers several advantages:

- It is the EJB 2.0 standard for queries.
- You can use it to construct most queries.
- You can implement dependent-object queries with EJB QL.

The disadvantage of EJB QL is that it is difficult to use when you construct complex queries.

For more information about EJB QL support in TopLink, see ["Query Languages"](#) on page 96-4.

For more information on TopLink predefined finders that take EJB QL, see ["Predefined Finders"](#) on page 96-25.

SQL Finders

Using SQL to define a finder offers the following advantages:

- You can implement logic that cannot be expressed when you use EJB QL or a TopLink `Expression`.
- It allows for the use of a stored procedure instead of TopLink generated SQL.
- There may be cases in which custom SQL will improve performance.

SQL finders also have the following disadvantages:

- Writing complex custom SQL statements requires a significant maintenance effort if the database tables change.
- Hard-coded SQL limits portability to other databases.
- No validation is performed on the SQL string. Errors in SQL statements will not be detected until run time.
- The use of SQL for a function other than `SELECT` may result in unpredictable errors.

For more information on TopLink predefined finders that take SQL, see ["Predefined Finders"](#) on page 96-25.

Redirect Finders

Redirect finders enable you to implement a finder that is defined on an arbitrary helper class as a static method. When you invoke the finder, TopLink redirects the call to the specified static method.

Redirect queries are complex and require an extra helper method to define the query. However, because they support complex logic, they are often the best choice when you need to implement logic unrelated to the bean on which the redirect method is called.

For more information, see the following:

- ["Redirect Queries"](#) on page 96-20
- ["Using EJB Finders"](#) on page 99-8

The `ejbSelect` Method

The `ejbSelect` method is a query method intended for internal use within an entity bean instance. Specified on the abstract bean itself, the `ejbSelect` method is not directly exposed to the client in the home or component interface. Defined as abstract, each bean can include zero or more such methods.

`ejbSelect` methods have the following characteristics:

- The method name must have `ejbSelect` as its prefix.
- It must be declared as `public`.
- It must be declared as `abstract`.
- The `throws` clause must specify the `javax.ejb.FinderException`, although it may also specify application-specific exceptions as well.
- Under EJB 2.0, the `result-type-mapping` tag in the `ejb-jar.xml` file determines the return type for `ejbSelect` methods. Set the flag to `Remote` to return `EJBObjects`; set it to `Local` to return `EJBLocalObjects`.

The format for an `ejbSelect` method definition looks like this:

```
public abstract type ejbSelect<METHOD>(...);
```

The `ejbSelect` query return type is not restricted to the entity bean type on which the `ejbSelect` is invoked. Instead, it can return any type corresponding to a container-managed relationship or container-managed field.

Although the `select` method is not based on the identity of the entity bean instance on which it is invoked, it can use the primary key of an entity bean as an argument. This creates a query that is logically scoped to a particular entity bean instance.

For more information and examples on using TopLink queries in the `ejbSelect` method, see ["Using EJB Finders"](#) on page 99-8.

Queries and the Cache

When you execute a query, TopLink retrieves the information from either the database or the TopLink session cache. You can configure the way queries use the TopLink cache to optimize performance.

TopLink maintains a client-side cache to reduce the number of read operations required from the database. TopLink caches objects written to and read from the database to maintain object identity. The sequence in which a query checks the cache and database affects query performance. By default, primary key queries check the cache before accessing the database, and all queries check the cache before rebuilding an object from its row.

Note: You can override the default behavior in the caching policy configuration information in the TopLink descriptor. For more information, see ["Explicit Query Refreshes"](#) on page 90-7.

This section illustrates ways to manipulate the relationship between query and cache, including the following:

- [Configuring the Cache](#)
- [Using In-Memory Queries](#)
- [Primary Key Queries and the Cache](#)
- [Disabling the Identity Map Cache Update During a Read Query](#)
- [Refreshing the Cache](#)
- [Caching Query Results in the Session Cache](#)
- [Caching Query Results](#)

Configuring the Cache

The cache in a TopLink application holds objects that have already been read from or written to the database. Use of the cache in a TopLink application reduces the number of accesses to the database. Because accessing the database consumes time and resources, an effective caching strategy is important to the efficiency of your application.

For more information about configuring and using the cache, see [Chapter 90, "Understanding the Cache"](#).

Using In-Memory Queries

An in-memory query is a query that is run against the shared session cache. Careful configuration of in-memory querying improves performance, but not all queries benefit from in-memory querying. For example, queries for individual objects based on primary keys generally see performance gains from in-memory querying; queries not based on primary keys are less likely to benefit.

By default, queries that look for a single object based on primary keys attempt to retrieve the required object from the cache first, and then to search the database if the object is not in the cache. All other query types search the database first, by default. You can specify whether a given query runs against the in-memory cache, the database, or both.

In-memory querying lets you perform queries on the cache rather than the database. In-memory querying supports the following relationships:

- One-to-one
- One-to-many
- Many-to-many
- Aggregate collection
- Direct collection

Note: By default, the relationships themselves must be in memory for in-memory traversal to work. Ensure that you trigger all value holders to enable in-memory querying to work across relationships.

This section describes the following:

- [Configuring Cache Usage for In-Memory Queries](#)

- [Expression Options for In-Memory Queries](#)
- [Handling Exceptions Resulting From In-Memory Queries](#)

Configuring Cache Usage for In-Memory Queries

You can configure in-memory query cache usage at the query level using `ReadObjectQuery` and `ReadAllQuery` methods:

- `checkCacheByPrimarykey`: The default setting; if a read-object query contains an expression that compares at least the primary key, you can obtain a cache hit if you process the expression against the objects in memory.
- `checkCacheByExactPrimarykey`: If a read-object query contains an expression where the primary key is the only comparison, you can obtain a cache hit if you process the expression against the object in memory.
- `checkCacheThenDatabase`: You can configure any read-object query to check the cache completely before you resort to accessing the database.
- `checkCacheOnly`: You can configure any read-all query to check only the parent session cache (not the unit of work cache) and return the result from the parent session cache without accessing the database.
- `conformResultsInUnitOfWork`: You can configure any read-object or read-all query within the context of a unit of work to conform the results with the changes to the object made within that unit of work. This includes new objects, deleted objects and changed objects. For more information and limitations on conforming, see ["Using Conforming Queries and Descriptors"](#) on page 102-8.

Alternatively, you can configure cache usage using the `ObjectLevelReadQuery` method `setCacheUsage`, passing in the appropriate `ObjectLevelReadQuery` field: `CheckCacheByPrimarykey`, `CheckCacheByExactPrimarykey`, `CheckCacheThenDatabase`, `CheckCacheOnly`, `ConformResultsInUnitOfWork`, or `DoNotCheckCache`.

Expression Options for In-Memory Queries

You can use a subset of `Expression` (see [Table 96–8](#)) and `ExpressionMath` (see [Table 96–9](#)) methods with in-memory queries. For more information about these options, see ["Understanding TopLink Expressions"](#) on page 97-1.

Table 96–8 *Expressions Operator Support for In-Memory Queries*

Expressions Operator	In-Memory Query Support
<code>addMonths</code>	
<code>and</code>	✓
<code>anyof¹</code>	✓
<code>anyofAllowingNone¹</code>	✓
<code>asciiValue</code>	
<code>between</code>	✓
<code>concat</code>	✓
<code>currentDate</code>	
<code>dateToString</code>	
<code>decode</code>	

Table 96–8 (Cont.) Expressions Operator Support for In-Memory Queries

Expressions Operator	In-Memory Query Support
equal	✓
get ¹	✓
getAllowingNull ¹	✓
getFunction	
greaterThan	✓
greaterThanEqual	✓
hexToRaw	
ifNull	
in	✓
isNull	✓
lastDay	
leftPad	
leftTrim	
length	✓
lessThan	✓
lessThanEqual	✓
like	✓
monthsBetween	
newTime	
nextDay	
notBetween	✓
notIn	
notIn	✓
notNull	✓
or	✓
ref	
replace	
rightPad	
rightTrim	
subQuery	
substring	✓
toCharacter	
toDate	
toLowerCase	✓
toNumber	✓
toUpperCase	✓

Table 96–8 (Cont.) Expressions Operator Support for In-Memory Queries

Expressions Operator	In-Memory Query Support
toUpperCasedWords	
translate	
trim	✓
truncateDate	

¹ For more information, see "[Join Reading and Object-Level Read Queries](#)" on page 96-12.

Table 96–9 ExpressionMath Operator Support for In-Memory Queries

ExpressionMath Operator	In-Memory Query Support
abs	✓
acos	✓
add	✓
asin	✓
atan	✓
atan2	
ceil	✓
chr	
cos	✓
cosh	
exp	✓
floor	✓
ln	
log	✓
max	✓
min	✓
mod	
none	
power	✓
round	✓
sign	
sin	✓
sinh	
sqrt	✓
subtract	✓
tan	✓
tanh	

Table 96–9 (Cont.) ExpressionMath Operator Support for In-Memory Queries

ExpressionMath Operator	In-Memory Query Support
trunc	

Handling Exceptions Resulting From In-Memory Queries

In-memory queries may fail for several reasons, the most common of which are the following:

- The query expression is too complex to execute in memory.
- There are untriggered value holders in which indirection is used. All object models that use indirection must first trigger value holders before they conform on the relevant objects.

TopLink provides a mechanism to handle indirection exceptions. To specify how the application must handle these exceptions, use the following `InMemoryQueryIndirectionPolicy` methods:

- `throwIndirectionException`: The default setting; it is the only setting that throws indirection exceptions.
- `triggerIndirection`: Triggers all valueholders to eliminate the problem.
- `ignoreIndirectionExceptionReturnConformed`: Returns conforming if an untriggered value holder is encountered. That is, results from the database are expected to conform, and an untriggered value holder is taken to mean that the underlying attribute has not changed.
- `ignoreIndirectionExceptionReturnNotConformed`: Returns not conforming if an untriggered value holder is encountered.

Note: When you build new applications, consider throwing all conform exceptions. This provides more detailed feedback for unsuccessful in-memory queries. For more information, see ["Exceptions During Conforming"](#) on page 102-36.

Primary Key Queries and the Cache

When a query searches for a single object by a primary key, TopLink extracts the primary key from the query and attempts to return the object from the cache without accessing the database. If the object is not in the cache, the query executes against the database, builds the resulting object(s), and places it in the identity map.

If the query is based on a nonprimary key selection criteria or is a read-all query, the query executes against the database (unless you are using `ReadObjectQuery` or `ReadAllQuery` method `checkCacheOnly`). The query matches primary keys from the result set to objects in the cache, and returns the cached objects, if any, in the result set.

If an object is not in the cache, TopLink builds the object. If the query is a refreshing query, TopLink updates the contents of any objects with the results from the query. Use "equals" on the object identity to properly configure and use an identity map.

Clients can refresh objects when they want to ensure that they have the latest data at a particular time.

Disabling the Identity Map Cache Update During a Read Query

To disable the identity map cache update, which is normally performed by a read query, call the `dontMaintainCache` method. This improves the query performance when you read objects that are not needed later by the application and can avoid exceptions during partial object queries (see ["Reading Objects Using Partial Object Queries"](#) on page 98-5).

Example 96–1 Disabling the Identity Map Cache Update

[Example 96–1](#) demonstrates how code reads `Employee` objects from the database and writes the information to a file.

```
// Reads objects from the employee table and writes them to an employee file
void writeEmployeeTableToFile(String filename, Session session)
{
    Vector employeeObjects;
    ReadAllQuery query = new ReadAllQuery();
    query.setReferenceClass(Employee.class);
    query.setSelectionCriteria(new
        ExpressionBuilder.get("id").greaterThan(100));
    query.dontMaintainCache();
    Vector employees = (Vector) session.executeQuery(query);
    // Write all the employee data to a file
    Employee.writeToFile(filename, employees);
}
```

Refreshing the Cache

You can refresh objects in the cache to ensure that they are current with the database, while preserving object identity. This section describes how to use query API to perform the following:

- Configure query refreshing at the descriptor level (see ["Configuring Cache Refreshing"](#) on page 28-32) to apply cache refreshing to all queries of a particular object type. Before configuring cache refresh options, consider their effect on performance (see ["Cache Optimization"](#) on page 11-15).

Object Refresh

To refresh objects in the cache with the data in the database, call the `Session` method `refreshObject` or the `ReadObjectQuery` method `setShouldRefreshIdentityMapResult(true)`.

Cascading Object Refresh

You can control the depth at which a refreshing updates objects and their related objects. There are three options:

1. *CascadePrivateParts*: Default refresh behavior. Refreshes the local level object and objects that are referenced in privately owned relationships.
2. *CascadeNone*: Refreshes only the first level of the object, but does not refresh related objects.
3. *CascadeAll*: Refreshes the entire object tree, stopping when it reaches leaf objects.

Refreshing the Identity Map Cache During a Read Query

Include the `refreshIdentityMapResult` method in a query to force refreshing of an identity map with the results of the query.

Example 96–2 Refreshing the Result of a Query in the Identity Map Cache During a Read Query

```
ReadObjectQuery query = new ReadObjectQuery();
query.setReferenceClass(Employee.class);
query.setSelectionCriteria(new
ExpressionBuilder().get("lastName").equal("Smith"));
query.refreshIdentityMapResult();
Employee employee = (Employee) session.executeQuery(query);
```

The `refreshIdentityMapResult` method refreshes the object's attributes, but not the attributes of its privately owned parts. However, under most circumstances, you should refresh an object's privately owned parts and other related objects to ensure consistency with the database.

To refresh privately owned or related parts, use the following methods:

- `cascadePrivateParts`: Refreshes all privately owned objects
- `cascadeAllParts`: Refreshes all related objects

Example 96–3 Using the `cascadePrivateParts` Method

```
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
query.refreshIdentityMapResult();
query.cascadePrivateParts();
Vector employees = (Vector) session.executeQuery(query);
```

Note: If the object is in the session cache, you can also use the `refreshObject` method to refresh an object and its privately owned parts.

Caching Query Results in the Session Cache

By default, TopLink stores query results in the session cache enabling TopLink to execute the query repeatedly, without accessing the database. This is useful when you execute queries that run against static data.

By default, a read-all query always goes to the database, as it does not know how many objects it is seeking. However if the object already exists in the cache, time can be saved by not having to build a new object from the row.

For more information, see "[Understanding the Cache](#)" on page 90-1.

Caching Query Results

In addition to TopLink's object cache, TopLink also supports a query cache.

- The **object cache** indexes objects by their primary key, allowing primary key queries to obtain cache hits. By using the object cache, queries that access the data source can avoid the cost of building the objects and their relationships if the object is already present.

- The **query cache** is distinct from the object cache. The query cache is indexed by the query and the query parameters – not the object's primary key. This allows for any query executed with the same parameters to obtain a query cache hit and return the same result set.

By default, a `ReadQuery` it does not cache its query result set. You can, however, configure the query to cache its result set. This is useful for frequently executed queries whose result set infrequently changes. The query cache always maintains hard references to the result set; the number of results sets for distinct parameters stored in the query cache is configurable. The query cache maintains its size number of the last executed queries with distinct parameters.

You can apply a cache invalidation policy to the query's internal cache (see ["Configuring Cache Expiration at the Query Level"](#) on page 99-24). For more information, see ["Cache Invalidation"](#) on page 90-8.

Internal Query Cache Restrictions

TopLink does not support the use of the query cache with cursors: if you use query caching with cursors, TopLink will throw an exception. For information on cursor query results, see ["Stream and Cursor Query Results"](#) on page 96-8 and ["Handling Cursor and Stream Query Results"](#) on page 99-17.

Caching and EJB Finders

TopLink caches EJB beans that EJB finders retrieve. For your application, you can configure the caching of the EJB finders' results in a variety of ways, force the cache to be refreshed, or disable the caching.

This section describes the following:

- [Caching Options](#)
- [Disabling Cache for Returned Finder Results](#)
- [Refreshing Finder Results](#)

Caching Options

You can apply various configurations to the underlying query to achieve the correct caching behavior for the application. There are several ways to control the caching options for queries. For most queries, you can set caching options using TopLink Workbench.

You can set the caching options on a per-finder basis. [Table 96–10](#) lists the valid values.

Table 96–10 *Finder Caching Options*

This Setting . . .	Causes Finders to . . .	When the Search Involves a Finder That . . .
<code>ConformResultsInUnitOfWork¹</code>	Check the unit of work cache before querying the session cache or the database. The finder's results always conform to uncommitted new, deleted, and changed objects.	Returns either a single bean or a collection.
<code>DoNotCheckCache</code>	Query the database, bypassing the TopLink internal caches.	Returns either a single bean or a collection.
<code>CheckCacheByExactPrimaryKey</code>	Check the session cache for the object.	Contains only a primary key, and returns a single bean.

Table 96–10 (Cont.) Finder Caching Options

This Setting . . .	Causes Finders to . . .	When the Search Involves a Finder That . . .
CheckCacheByPrimaryKey	Check the session cache for the object.	Contains a primary key (and may contain other search parameters), and returns a single bean.
CheckCacheThenDatabase	Search the session cache before accessing the database.	Returns a single bean.
CheckCacheOnly	Search the parent session cache only (not the unit of work cache), but not the database.	Returns either a single bean or a collection.

¹ Default.

For more information about the TopLink queries, as well as the TopLink unit of work and how it integrates with JTS, see [Chapter 100, "Understanding TopLink Transactions"](#).

Note: To apply caching options to finders with manually created queries (`findOneByQuery`, `findManyByQuery`), use the TopLink API.

Disabling Cache for Returned Finder Results

By default, TopLink adds all returned objects to the session cache. However, if you know the set of returned objects is very large, and you want to avoid the expense of storing these objects, you can disable this behavior. To override the default configuration, implement the `dontMaintainCache` method on the query, or disable returned object caching for the query in TopLink Workbench.

Refreshing Finder Results

A finder may return information from the database for an object whose primary key is already in the cache. When set to `true`, the Refresh Cache option (in TopLink Workbench) causes the query to refresh the object's nonprimary key attributes with the returned information. This occurs on `findByPrimaryKey` finders as well as all expression and SQL finders for the bean.

If you build a query in Java code, you can set this option by including the `refreshIdentityMapResult` method. This method automatically cascades changes to privately owned parts of the beans. If you require different behavior, configure the query using a dynamic finder instead.

Note: When you invoke this option from within a transaction, the refresh action overwrites object attributes, including any that have not yet been written to the database.

If your application includes an `OptimisticLock` field, use the refresh cache option in conjunction with the `onlyRefreshCacheIfNewerVersion` option. This ensures that the application refreshes objects in the cache only if the version of the object in the database is newer than the version in the cache.

For finders that have no refresh cache setting, the `onlyRefreshCacheIfNewerVersion` method has no effect.

Understanding the Query API

[Table 96–11](#) summarizes the query support provided by each type of session. For each session type, it shows the type of query operation (create, read, update, delete) that you can perform and whether or not you can execute a `DatabaseQuery` or `Call`. For example, using a unit of work, you can use session queries to read and delete; using a server session, you can use session queries to create, read, update, and delete.

Table 96–11 Session Query API Summary

Session	Create	Read	Update	Delete	Execute Database Query	Execute Call
Unit of work		✓		✓	✓	✓
Database	✓	✓	✓	✓	✓	✓
Server						
Client		✓			✓	✓

[Example 96–4](#) summarizes the important TopLink packages that provide query and expression support:

Example 96–4 Query and Expression Packages

```
oracle.toplink.queryframework
oracle.toplink.expressions
oracle.toplink.querykeys
oracle.toplink.descriptors.DescriptorQueryManager
```

Using Basic Query API

This chapter explains the essential TopLink query API calls most commonly used throughout the development cycle:

- [Using Session Queries](#)
- [Using DatabaseQuery Queries](#)
- [Using Named Queries](#)
- [Using SQL Calls](#)
- [Using EJB QL Calls](#)
- [Using EIS Interactions](#)
- [Handling Exceptions](#)
- [Handling Collection Query Results](#)
- [Handling Report Query Results](#)

For more information, see ["Using Advanced Query API"](#) on page 99-1.

Using Session Queries

This section provides examples of using the session query methods for the following:

- [Reading Objects With a Session Query](#)
- [Creating, Updating, and Deleting Objects With a Session Query](#)

Note: Oracle recommends that you perform all data source operations using a unit of work: doing so is the most efficient way to manage transactions, concurrency, and referential constraints. For more information, see ["Understanding TopLink Transactions"](#) on page 100-1.

For more information, see ["Session Queries"](#) on page 96-9.

Reading Objects With a Session Query

Using the session query API, you can perform the following read operations:

- [Reading an Object With a Session Query](#)
- [Reading All Objects With a Session Query](#)
- [Refreshing an Object with a Session Query](#)

Reading an Object With a Session Query

The `readObject` method retrieves a single object from the database. The application must specify the class of object to read. If no object matches the criteria, a null value is returned.

For example, the basic read operation is:

```
session.readObject(MyDomainObject.class);
```

This example returns the first instance of `MyDomainObject` found in the table used for `MyDomainObject`. `TopLink` provides the `Expression` class to specify querying parameters for a specific object.

When you search for a single, specific object using a primary key, the `readObject` method is more efficient than the `readAllObjects` method, because `readObject` can find an instance in the cache without accessing database. Because a `readAllObjects` method does not know how many objects match the criteria, it always searches the database to find matching objects, even if it finds matching objects in the cache.

Example 98–1 *readObject Using an Expression*

```
import oracle.toplink.sessions.*;
import oracle.toplink.expressions.*;

/* Use an expression to read in the employee whose last name is Smith. Create an
expression using the Expression Builder and use it as the selection criterion of
the search */
Employee employee = (Employee) session.readObject(Employee.class, new
ExpressionBuilder().get("lastName").equal("Smith"));
```

Reading All Objects With a Session Query

The `readAllObjects` method retrieves a `Vector` of objects from the database and does not put the returned objects in order. If the query does not find any matching objects, it returns an empty `Vector`.

Specify the class for the query. You can also include an expression to define more complex search criteria, as illustrated in [Example 98–2](#).

Example 98–2 *readAllObjects Using an Expression*

```
// Returns a Vector of employees whose employee salary is greater than 10000
Vector employees = session.readAllObjects(Employee.class, new
ExpressionBuilder().get("salary").greaterThan(10000));
```

Refreshing an Object with a Session Query

The `refreshObject` method causes `TopLink` to update the object in memory using data from the database. This operation refreshes any privately owned objects as well.

Note: A privately owned object is one that cannot exist without its parent, or source object.

Creating, Updating, and Deleting Objects With a Session Query

Using the session query API, you can perform the following create, update, and delete operations:

- [Writing a Single Object to the Database with a Session Query](#)
- [Writing All Objects to the Database With a Session Query](#)
- [Adding New Objects to the Database with a Session Query](#)
- [Modifying Existing Objects in the Database with a Session Query](#)
- [Deleting Objects in the Database with a Session Query](#)

Writing a Single Object to the Database with a Session Query

When you invoke the `writeObject` method, the method performs a *does-exist* check to determine whether or not an object exists. If the object exists, `writeObject` updates the object; if it does not exist, `writeObject` inserts a new object.

The `writeObject` method writes privately owned objects in the correct order to maintain referential integrity.

Call the `writeObject` method when you cannot verify that an object exists in the database.

Example 98–3 Writing a Single Object Using `writeObject`

```
// Create an instance of the employee and write it to the database
Employee susan = new Employee();
susan.setName("Susan");
...
// Initialize the susan object with all other instance variables
session.writeObject(susan);
```

Writing All Objects to the Database With a Session Query

You can call the `writeAllObjects()` method to write multiple objects to the database. The `writeAllObjects()` method performs the same *does-exist* check as the `writeObject()` method and then performs the appropriate insert or update operations.

Example 98–4 Writing Several Objects Using `writeAllObjects`

```
// Read a Vector of all the current employees in the database.
Vector employees = (Vector) session.readAllObjects(Employee.class);
...// Modify any employee data as necessary
// Create a new employee and add it to the list of employees
Employee susan = new Employee();
...
// Initialize the new instance of employee
employees.add(susan);
/* Write all employees to the database. The new instance of susan not currently in
the database will be inserted. All the other employees currently stored in the
database will be updated */
session.writeAllObjects(employees);
```

Adding New Objects to the Database with a Session Query

The `insertObject` method creates a new object in the database, but does not perform the *does-exist* check before it attempts the insert operation. The `insertObject` method is more efficient than the `writeObject` method if you are certain that the object does not yet exist in the database. If the object does exist, the database throws an exception when you execute the `insertObject` method.

Modifying Existing Objects in the Database with a Session Query

The `updateObject` method updates existing objects in the database, but does not perform the *does-exist* check before it attempts the update operation. The `updateObject` is more efficient than the `writeObject` method if you are certain that the object does exist in the database. If the object does not exist, the database throws an exception when you execute the `updateObject` method.

Deleting Objects in the Database with a Session Query

To delete a TopLink object from the database, read the object from the database and then call the `deleteObject` method. This method deletes both the specified object and any privately owned data.

Using DatabaseQuery Queries

This section describes creating and executing `DatabaseQuery` queries to perform a variety of basic persistence operations, including the following:

- [Reading Objects Using a DatabaseQuery](#)
- [Creating, Updating, and Deleting Objects with a DatabaseQuery](#)
- [Reading Data With a DatabaseQuery](#)
- [Updating Data With a DatabaseQuery](#)
- [Specifying a Custom SQL String in a DatabaseQuery](#)
- [Specifying a Custom EJB QL String in a DatabaseQuery](#)
- [Using Parameterized SQL and Statement Caching in a DatabaseQuery](#)

Reading Objects Using a DatabaseQuery

This section provides examples that illustrate how to read objects using a `DatabaseQuery`, including the following:

- [Basic DatabaseQuery Read Operations](#)
- [Reading Objects Using Partial Object Queries](#)
- [Reading Objects Using Report Queries](#)
- [Reading Objects Using Query By Example](#)
- [Specifying Read Ordering](#)
- [Specifying a Collection Class](#)
- [Specifying the Maximum Rows Returned](#)
- [Configuring Query Timeout at the Query Level](#)
- [Using Batch Reading](#)
- [Using Join Reading](#)

Basic DatabaseQuery Read Operations

[Example 98-5](#) illustrates a simple read query. It uses a TopLink expression, but does not use its own arguments for the query. Instead, it relies on the search parameters the expression provides. This example builds the expression within its code, but does not register the query with the session.

Example 98-5 A Simple ReadAllQuery

```
// This example returns a Vector of employees whose employee ID is > 100

// Initialize the DatabaseQuery by specifying the query type
ReadAllQuery query = new ReadAllQuery();

// Set the reference class for the query
query.setReferenceClass(Employee.class);

/* Configure the query execution. Because this example uses an expression, it uses
the setSelectionCriteria method */
query.setSelectionCriteria(new ExpressionBuilder.get("id").greaterThan(100));

// Execute the query
Vector employees = (Vector) session.executeQuery(query);
```

[Example 98-6](#) illustrates a complex readObject query that uses all available configuration options.

Example 98-6 A Named Read Query with Two Arguments

```
// Define two expressions that map to the first and last names of the employee
ExpressionBuilder emp = new ExpressionBuilder();
Expression firstNameExpression =
emp.get("firstName").equal(emp.getParameter("firstName"));
Expression lastNameExpression =
emp.get("lastName").equal(emp.getParameter("lastName"));

// Initialize the DatabaseQuery by specifying the query type
ReadObjectQuery query = new ReadObjectQuery();
// Set the reference class for the query
query.setReferenceClass(Employee.class);
/* Configure the query execution. Because this example uses an expression, it uses
the setSelectionCriteria method */
query.setSelectionCriteria(firstNameExpression.and(lastNameExpression));
// Specify the required arguments for the query
query.addArgument("firstName");
query.addArgument("lastName");

// Add the query to the session
session.addQuery("getEmployeeWithName", query);

/* Execute the query by referencing its name and providing values for the
specified arguments */
Employee employee = (Employee)
session.executeQuery("getEmployeeWithName", "Bob", "Smith");
```

Reading Objects Using Partial Object Queries

[Example 98-7](#) demonstrates the use of partial object reading. It reads only the last name and primary key for the employees. This reduces the amount of data read from the database.

Example 98-7 Optimization Through Partial Object Reading

```
/* Read all the employees from the database, ask the user to choose one and return
it. This uses partial object reading to read just the last name of the employees.
Since TopLink automatically includes the primary key of the object, the full
object can easily be read for editing */
List list;
// Fetch data from database and add to list box
ReadAllQuery query = new ReadAllQuery(Employee.class);
query.addPartialAttribute("lastName");
// The next line avoids a query exception
query.dontMaintainCache();
Vector employees = (Vector) session.executeQuery(query);
list.addAll(employees);

// Display list box
....
// Get selected employee from list
Employee selectedEmployee = (Employee)session.readObject(list.getSelectedItem());
return selectedEmployee;
```

Reading Objects Using Report Queries

[Example 98-8](#) reports the total and average salaries for Canadian employees grouped by their city.

Example 98-8 Querying Reporting Information on Employees

```
ExpressionBuilder emp = new ExpressionBuilder();
ReportQuery query = new ReportQuery(emp);
query.setReferenceClass(Employee.class);
query.addMaximum("max-salary", emp.get("salary"));
query.addAverage("average-salary", emp.get("salary"));
query.addAttribute("city", emp.get("address").get("city"));

query.setSelectionCriteria(emp.get("address").get("country").equal("Canada"));
query.addOrdering(emp.get("address").get("city"));
query.addGrouping(emp.get("address").get("city"));
Vector reports = (Vector) session.executeQuery(query);
```

The `ReportQuery` class provides an extensive reporting API, including methods for computing average, maximum, minimum, sum, standard deviation, variance, and count of attributes. For more information about the available methods for the `ReportQuery`, see the *Oracle TopLink API Reference*.

Note: Because `ReportQuery` inherits from `ReadAllQuery`, it also supports most `ReadAllQuery` properties.

Reading Objects Using Query By Example

Query by example enables you to specify query selection criteria in the form of a sample object instance that you populate with only the attributes you want to use for the query.

To define a query by example, provide a `ReadObjectQuery` or a `ReadAllQuery` with a sample persistent object instance and an optional query by example policy. The sample instance contains the data to query, and, optionally, a `QueryByExamplePolicy` (see ["Defining a QueryByExamplePolicy"](#) on page 98-7)

that specifies configuration settings, such as the operators to use and the attribute values to ignore. You can also combine a query by example with an expression (see ["Combining Query by Example and Expressions"](#) on page 98-8).

For more information, see ["Query by Example"](#) on page 96-6.

Example 98–9 Using Query by Example

[Example 98–9](#) queries the employee Bob Smith.

```
ReadObjectQuery query = new ReadObjectQuery();
Employee employee = new Employee();
employee.setFirstName("Bob");
employee.setLastName("Smith");
query.setExampleObject(employee);

Employee result = (Employee) session.executeQuery(query);
```

Example 98–10 Using Query by Example

[Example 98–10](#) queries across the employee's address.

```
ReadAllQuery query = new ReadAllQuery();
Employee employee = new Employee();
Address address = new Address();
address.setCity("Ottawa");
employee.setAddress(address);
query.setExampleObject(employee);

Vector results = (Vector) session.executeQuery(query);
```

Defining a QueryByExamplePolicy

TopLink support for query by example includes a query by example policy. You can edit the policy to modify query by example default behavior. You can modify the policy to do the following:

- Use LIKE or other operations to compare attributes. By default, query by example allows only EQUALS.
- Modify the set of values query by example ignores (the IGNORE set). The default ignored values are zero (0), empty strings, and FALSE.
- Force query by example to consider attribute values, even if the value is in the IGNORE set.
- Use isNull or notNull for attribute values.

To specify a query by example policy, include an instance of `QueryByExamplePolicy` with the query.

Example 98–11 Query by Example Policy Using like Operator

[Example 98–11](#) uses like operator for strings and includes only objects whose salary is greater than zero.

```
ReadAllQuery query = new ReadAllQuery();
Employee employee = new Employee();
employee.setFirstName("B%");
employee.setLastName("S%");
employee.setSalary(0);
query.setExampleObject(employee);
```

```
/* Query by example policy section adds like and greaterThan */
QueryByExamplePolicy policy = new QueryByExamplePolicy();
policy.addSpecialOperation(String.class, "like");
policy.addSpecialOperation(Integer.class, "greaterThan");
policy.alwaysIncludeAttribute(Employee.class, "salary");
query.setQueryByExamplePolicy(policy);
Vector results = (Vector) session.executeQuery(query);
```

Example 98–12 Query by Example Policy Using Keywords

[Example 98–12](#) uses keywords for strings and ignores the value -1.

```
ReadAllQuery query = new ReadAllQuery();
Employee employee = new Employee();
employee.setFirstName("bob joe fred");
employee.setLastName("smith mc mac");
employee.setSalary(-1);
query.setExampleObject(employee);
/* Query by example policy section */
QueryByExamplePolicy policy = new QueryByExamplePolicy();
policy.addSpecialOperation(String.class, "containsAnyKeyWords");
policy.excludeValue(-1);
query.setQueryByExamplePolicy(policy);
Vector results = (Vector) session.executeQuery(query);
```

Combining Query by Example and Expressions

To create more complex query by example queries, combine query by example with TopLink expressions, as shown in [Example 98–13](#).

Example 98–13 Combining Query by Example with Expressions

```
ReadAllQuery query = new ReadAllQuery();
Employee employee = new Employee();
employee.setFirstName("Bob");
employee.setLastName("Smith");
query.setExampleObject(employee);
/* This section specifies the expression */
ExpressionBuilder builder = new ExpressionBuilder();
query.setSelectionCriteria(builder.get("salary").between(100000, 200000));
Vector results = (Vector) session.executeQuery(query);
```

Specifying Read Ordering

Ordering is a common DatabaseQuery option. Use the Order tab in TopLink Workbench to order the collection of objects returned from a ReadAllQuery, or the addOrdering, addAscendingOrdering, or addDescendingOrdering methods in Java code. You can apply order based on attribute names or query keys and expressions.

Example 98–14 A Query with Simple Ordering

```
// Retrieves objects ordered by last name then first name in ascending order
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
query.addAscendingOrdering("lastName");
query.addAscendingOrdering("firstName");
Vector employees = (Vector) session.executeQuery(query);
```

Example 98–15 A Query with Complex Ordering

```

/* Retrieves objects ordered by street address, descending case-insensitive order
of cities, and manager's last name */
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
ExpressionBuilder emp = new ExpressionBuilder();
query.addOrdering (emp.getAllowingNull("address").get("street"));
query.addOrdering
(emp.getAllowingNull("address").get("city").toUpperCase().descending());
query.addOrdering(emp.getAllowingNull("manager").get("lastName"));
Vector employees = (Vector) session.executeQuery(query);

```

Note the use of `getAllowingNull`, which creates an outer join for the address and manager relationships. This ensures that employees without an address or manager still appear in the list.

For more information about configuring read ordering, see ["Configuring Read All Query Order"](#) on page 28-18.

Specifying a Collection Class

By default, a `ReadAllQuery` returns its result objects in a vector. You can configure the query to return the results in any collection class that implements the `Collection` or `Map` interface, as shown in [Example 98–16](#).

Example 98–16 Specifying the Collection Class for a Collection

```

ReadAllQuery query = new ReadAllQuery(Employee.class);
query.useCollectionClass(LinkedList.class);
LinkedList employees = (LinkedList) getSession().executeQuery(query);

```

Example 98–17 Specifying the Collection Class for a Map

```

ReadAllQuery query = new ReadAllQuery(Employee.class);
query.useMapClass(HashMap.class, "getFirstName");
HashMap employees = (HashMap) getSession().executeQuery(query);

```

Specifying the Maximum Rows Returned

You can limit a query to a specified maximum number of rows. Use this feature to avoid queries that can return an excessive number of objects.

To specify a maximum number of rows, use the `setMaxRows` method, and pass an integer that represents the maximum number of rows for the query, as shown in [Example 98–18](#).

Example 98–18 Setting the Maximum Returned Object Size

```

ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
query.setMaxRows(5);
Vector employees = (Vector) session.executeQuery(query);

```

The `setMaxRows` method limits the number of rows the query returns, but does not let you acquire more records after the initial result set.

If you want to browse the result set in fixed increments, use either cursors or cursored streams. For more information, see ["Handling Cursor and Stream Query Results"](#) on page 99-17.

Configuring Query Timeout at the Query Level

You can set the maximum amount of time that TopLink waits for results from a query. This forces a hung or lengthy query to abort after the specified time has elapsed. TopLink throws a `DatabaseException` after the timeout interval.

To specify a timeout interval on a per-query basis, use `DatabaseQuery` method `setQueryTimeout` and pass the timeout interval as an integer representing the number of seconds before the timeout interval should occur, as [Example 98–19](#) shows.

Example 98–19 DatabaseQuery Timeout

```
// Create the appropriate query and set timeout limits
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
query.setQueryTimeout(2);
try{
    Vector employees = (Vector)session.executeQuery(query);
} catch (DatabaseException ex) {
    // timeout occurs
}
```

To specify a timeout interval for all queries on a particular object type, configure a query timeout interval at the descriptor level (see ["Configuring Query Timeout at the Descriptor Level"](#) on page 28-31).

Using Batch Reading

Batch reading propagates query selection criteria through an object's relationship attribute mappings. You can also nest batch read operations down through complex object graphs. This significantly reduces the number of required SQL select statements and improves database access efficiency.

Consider the following guidelines when you implement batch reading:

- Use batch reading for processes that read in objects and all their related objects.
- Do not enable batch reading for both sides of a bidirectional relationship.
- Avoid nested batch read operations, because they result in multiple joins on the database, slowing query execution.

For more information, see ["Reading Case 2: Batch Reading Objects"](#) on page 11-22.

For example, in reading n employees and their related projects, TopLink may require $n + 1$ select operations. All employees are read at once, but the projects of each are read individually. With batch reading, all related projects can also be read with one select operation by using the original selection criteria, for a total of only two select operations.

To implement batch reading, use one of the following methods:

- To add the batch read attribute to a query, use the `query.addBatchReadAttribute(Expression anExpression)` API.

For example:

```
...
ReadAllQuery raq = new ReadAllQuery(Trade.class);
```

```

ExpressionBuilder tradeBuilder = raq.getBuilder();
...
Expression batchReadProduct = tradeBuilder.get("product");
readAllQuery.addBatchReadAttribute(batchReadProduct);
Expression batchReadPricingDetails = batchReadProduct.get("pricingDetails");
readAllQuery.addBatchReadAttribute(batchReadPricingDetails);
...

```

- Add batch reading at the mapping level for a descriptor. Use either TopLink Workbench or a descriptor amendment method to add the `setUsesBatchReading` API on the descriptor's relationship mappings.

For example:

```

public static void amendTradeDescriptor(Descriptor theDescriptor) {
    OneToOneMapping productOneToOneMapping =
        theDescriptor.getMappingForAttributeName("product");
    productOneToOneMapping.setUsesBatchReading(true);
}

```

You can combine batch reading and indirection to provide controlled reading of object attributes. For example, if you have one-to-one back pointer relationship attributes, you can defer back pointer instantiation until the end of the query, when all parent and owning objects are instantiated. This prevents unnecessary database access and optimizes TopLink cache use.

Using Join Reading

Use join reading to configure a query for a class to return the data to build the instances of that class and its related objects. For more information, see ["Join Reading and Object-Level Read Queries"](#) on page 96-12.

To add one or more joined attributes to a query, you can use either TopLink Workbench or Java.

Using TopLink Workbench To add one or more joined attributes to a query using TopLink Workbench, configure joined attributes when you define named queries (see ["Configuring Named Query Optimization"](#) on page 28-19) or Java. You cannot use TopLink Workbench to create an `ObjectLevelReadQuery` with a join expression on a one-to-many mapped attribute: you must use Java.

Using Java You can use `ObjectLevelReadQuery` API to add joined attributes for one-to-one and one-to-many relationships.

Use the `ObjectLevelReadQuery` method `addJoinedAttribute(Expression attributeExpression)` to add join expressions to the query. Using this method, you can add multiple joined attributes for one-to-one and one-to-many relationships, including nested joins. The source and target can be the same class type. You cannot use the `ObjectLevelReadQuery` method `addJoinedAttribute` with a join expression on a many-to-many mapped attribute.

- Use the `ObjectLevelReadQuery` method `addJoinedAttribute` with a join expression on a one-to-one mapped attribute to get the class of the `ObjectLevelReadQuery` and the target of the one-to-one mapped attribute of that class with a single database hit.
- Use the `ObjectLevelReadQuery` method `addJoinedAttribute` with a join expression on a one-to-many mapped attribute to get the class of the `ObjectLevelReadQuery` and the target collection of the one-to-many mapped attribute of that class with a single database hit.

[Example 98–20](#) is based on the TopLink ThreeTierEmployee example project. It shows a ReadAllQuery configured to join-read multiple attributes. This query produces the SQL that [Example 98–21](#) shows.

Example 98–20 Join Reading Multiple Attributes

```
ReadAllQuery query = new ReadAllQuery(Employee.class);

Expression managedEmployees = query.getExpressionBuilder().anyOfAllowingNone(
    "managedEmployees"
);
query.addJoinedAttribute(managedEmployees);
query.addJoinedAttribute(managedEmployees.get("address"));
query.addJoinedAttribute(managedEmployees.anyOf("phoneNumbers"));

Vector employees = (Vector)getSession().executeQuery(query);
```

Example 98–21 SQL for Multiple Attribute Join Reading

```
SELECT DISTINCT
    t2.VERSION, t3.EMP_ID, t2.GENDER, t3.SALARY, t2.EMP_ID, t2.F_NAME, t2.L_NAME,
    t2.MANAGER_ID, t2.ADDR_ID, t2.END_DATE, t2.START_DATE, t2.END_TIME,
    t2.START_TIME, t0.VERSION, t1.EMP_ID, t0.GENDER, t1.SALARY, t0.EMP_ID,
    t0.F_NAME, t0.L_NAME, t0.MANAGER_ID, t0.ADDR_ID, t0.END_DATE, t0.START_DATE,
    t0.END_TIME, t0.START_TIME
FROM
    SALARY t3, EMPLOYEE t2, SALARY t1, EMPLOYEE t0
WHERE
    ((t3.EMP_ID = t2.EMP_ID) AND
    ((t0.MANAGER_ID (+) = t2.EMP_ID) AND
    (t1.EMP_ID (+) = t0.EMP_ID)))
```

Use the ObjectLevelReadQuery method addJoinedAttribute(java.lang.String attributeName) to configure the query to join-read a single attribute, as [Example 98–22](#) shows.

Example 98–22 Join Reading a Single Attribute

```
ReadAllQuery query = new ReadAllQuery(Employee.class);
query.addJoinedAttribute("address");
Vector employees = (Vector)getSession().executeQuery(query);
```

Creating, Updating, and Deleting Objects With a DatabaseQuery

This section describes the following:

- [Write Query Overview](#)
- [Noncascading Write Queries](#)
- [Disabling the Identity Map Cache During a Write Query](#)

Write Query Overview

To execute a write query, use a WriteObjectQuery instance instead of using the writeObject method of the session. Likewise, substitute DeleteObjectQuery, UpdateObjectQuery, and InsertObjectQuery objects for their respective Session methods.

Example 98–23 Using a WriteObjectQuery

```
WriteObjectQuery writeQuery = new WriteObjectQuery();
writeQuery.setObject(domainObject);
session.executeQuery(writeQuery);
```

Example 98–24 Using InsertObjectQuery, UpdateObjectQuery, and DeleteObjectQuery

```
InsertObjectQuery insertQuery= new InsertObjectQuery();
insertQuery.setObject(domainObject);
session.executeQuery(insertQuery);
```

/* When you use UpdateObjectQuery without a unit of work, UpdateObjectQuery writes all direct attributes to the database */

```
UpdateObjectQuery updateQuery= new UpdateObjectQuery();
updateQuery.setObject(domainObject2);
session.executeQuery(updateQuery);
```

```
DeleteObjectQuery deleteQuery = new DeleteObjectQuery();
deleteQuery.setObject(domainObject2);
session.executeQuery(deleteQuery);
```

UpdateAll Queries

Use an UpdateAllQuery to update a large number of objects at once. With this query, you can update a large number of objects with a single SQL statement, instead of reading the objects into memory and updating them individually. [Example 98–25](#) shows an UpdateAllQuery to give all full-time employees a raise.

Since multiple tables cannot be updated from the same SQL statement, the UpdateAllQuery does not support objects that span multiple tables or inheritance. Additionally, the UpdateAllQuery must be executed from its own transaction—the unit of work must contain only the query.

Example 98–25 Using UpdateAllQuery

```
/* Give all full time employees a 10% raise
UpdateAllQuery updateQuery = new UpdateAllQuery();
updateQuery.setReferenceClass(Employee.class);
ExpressionBuilder employee = new ExpressionBuilder();
updateQuery.setSelectionCriteria(eb.get("status").equal("FULL_TIME"));
updateQuery.addUpdateExpression(employee.get("salary"),
    ExpressionMath.multiply(employee.get("salary"), new Float(1.10)));
```

Noncascading Write Queries

When you execute a write query, it writes both the object and its privately owned parts to the database by default. To build write queries that do not update privately owned parts, include the dontCascadeParts method in your query definition.

Use this method to do the following:

- Increase performance when you know that only the object's direct attributes have changed.
- Resolve referential integrity dependencies when you write large groups of new, independent objects.

Note: Because the unit of work resolves referential integrity internally, this method is not required if you use the unit of work to write to the database.

Example 98–26 Performing a Noncascading Write Query

```
// the Employee is an existing employee read from the database
Employee.setFirstName("Bob");
UpdateObjectQuery query = new UpdateObjectQuery();
query.setObject(Employee);
query.dontCascadeParts();
session.executeQuery(query);
```

Disabling the Identity Map Cache During a Write Query

When you write objects to the database, TopLink copies them to the session cache by default. To disable this within a query, call the `dontMaintainCache` method within the query. This improves query performance when you insert objects into the database, but must be used only on objects that will not be required later by the application.

Example 98–27 Disabling the Identity Map Cache During a Write Query

[Example 98–27](#) reads all the objects from a flat file and writes new copies of the objects into a table.

```
// Reads objects from an employee file and writes them to the employee table
void createEmployeeTable(String filename, Session session) {
    Iterator iterator;
    Employee employee;
    // Read the employee data file
    List employees = Employee.parseFromFile(filename);
    Iterator iterator = employees.iterator();
    while (iterator.hasNext()) {
        Employee employee = (Employee) iterator.next();
        InsertObjectQuery query = new InsertObjectQuery();
        query.setObject(employee);
        query.dontMaintainCache();
        session.executeQuery(query);
    }
}
```

Note: Disable the identity map only when object identity is unimportant in subsequent operations.

Reading Data With a DatabaseQuery

This section describes the following:

- [Using a DataReadQuery](#)
- [Using a DirectReadQuery](#)
- [Using a ValueReadQuery](#)

Using a DataReadQuery

You can use a `DataReadQuery` to execute a selecting SQL string that returns a `Collection` of the `DatabaseRows` representing the result set, as [Example 98–28](#) shows.

Example 98–28 Using a DataReadQuery

```
DataReadQuery dataReadQuery = new DataReadQuery();
dataReadQuery.setSQLStatement(sqlStatement);

// queryResults is a Vector of DatabaseRow objects
Vector queryResults = (Vector)session.executeQuery(dataReadQuery);
```

Using a DirectReadQuery

You can use a `DirectReadQuery` to read a single column of data (that is, one field) that returns a `Collection` of the `DatabaseRows` representing the result set, as [Example 98–29](#) shows.

Example 98–29 Using a DirectReadQuery

```
DirectReadQuery directReadQuery = new DirectReadQuery();
directReadQuery.setSQLStatement(sqlStatement);

// queryResults is a Vector of DatabaseRow objects
Vector queryResults = (Vector)session.executeQuery(directReadQuery);
```

Using a ValueReadQuery

You can use a `ValueReadQuery` to read a single data value (that is, one field). A single data value is returned, or null if no rows are returned, as [Example 98–30](#) shows.

Example 98–30 Using a ValueReadQuery

```
ValueReadQuery valueReadQuery = new ValueReadQuery();
valueReadQuery.setSQLString("SELECT DISTINCT CURRENT_TIMESTAMP FROM SYSTABLES");

// result is a single Object value
Object result = session.executeQuery(valueReadQuery);
```

WARNING: Allowing an unverified SQL string to be passed into methods (for example: `setSQLString` method) makes your application vulnerable to SQL injection attacks.

Updating Data With a DatabaseQuery

You can use a `DataModifyQuery` to execute a nonselecting SQL statement (directly or as an `SQLCall`), as [Example 98–31](#) shows. This is equivalent to `Session` method `executeNonSelectingCall` (see "[Using an SQLCall](#)" on page 98-18).

Example 98–31 Using a DataModifyQuery

```
DataModifyQuery query = new DataModifyQuery(new SQLCall("Delete from Employee"));
session.executeQuery(query);
```

Specifying a Custom SQL String in a DatabaseQuery

All DatabaseQuery objects provide a `setSQLString` method that you can use to define a custom SQL string.

For more information about using custom SQL in queries, see ["Using SQL Calls"](#) on page 98-18.

[Example 98-32](#) uses SQL to read all employee IDs.

Example 98-32 A Direct Read Query with SQL

```
DirectReadQuery query = new DirectReadQuery();
query.setSQLString("SELECT EMP_ID FROM EMPLOYEE");
Vector ids = (Vector) session.executeQuery(query);
```

[Example 98-33](#) uses SQL to switch to a different database.

Example 98-33 A Data Modify Query with SQL

```
DataModifyQuery query = new DataModifyQuery();
query.setSQLString("USE SALESDATABASE");
session.executeQuery(query);
```

WARNING: Allowing an unverified SQL string to be passed into methods (for example: `setSQLString` method) makes your application vulnerable to SQL injection attacks.

Specifying a Custom EJB QL String in a DatabaseQuery

All DatabaseQuery objects provide a `setEJBQLString` method that you can use to specify a custom EJB QL string.

For more information about using custom EJB QL in queries, see ["Using EJB QL Calls"](#) on page 98-24.

Provide both a reference class and a SELECT clause, and execute the query in the usual manner.

Example 98-34 EJB QL

```
ReadAllQuery theQuery = new ReadAllQuery();
theQuery.setReferenceClass(EmployeeBean.class);
theQuery.setEJBQLString("SELECT OBJECT(emp) FROM EmployeeBean emp");
...
Vector returnedObjects = (Vector)aSession.executeQuery(theQuery);
```

[Example 98-35](#) defines the query similarly to [Example 98-34](#), but creates, fills, and passes a vector of arguments to the `executeQuery` method.

Example 98-35 A Simple ReadAllQuery Using EJB QL and Passing Arguments

```
// First define the query
ReadAllQuery theQuery = new ReadAllQuery();
theQuery.setReferenceClass(EmployeeBean.class);
theQuery.setEJBQLString("SELECT OBJECT(emp) FROM EmployeeBean emp WHERE emp.firstName = ?1");
theQuery.addArgument("1", String.class);
...
// Next define the arguments
Vector theArguments = new Vector();
theArguments.add("Bob");
...
```

```
// Finally, execute the query passing in the arguments
Vector returnedObjects = (Vector)aSession.executeQuery(theQuery, theArguments);
```

Using Parameterized SQL and Statement Caching in a DatabaseQuery

To enable the parameterized SQL on individual queries, use `DatabaseQuery` methods `bindAllParameters` and `cacheStatement`. This causes TopLink to use a prepared statement, binding all SQL parameters and caching the prepared statement. When you reexecute this query, you avoid the SQL preparation, which improves performance.

Example 98–36 A Simple ReadObjectQuery with Parameterized SQL

```
ReadObjectQuery query = new ReadObjectQuery(Employee.class);
query.setShouldBindAllParameters(true);
query.setShouldCacheStatement(true);
```

Alternatively, you can configure parameterized SQL and binding at the `Login` level for all queries (see ["Configuring JDBC Options"](#) on page 86-9).

For more information about using parameterized SQL and binding for data access optimization, see ["Parameterized SQL \(Binding\) and Prepared Statement Caching"](#) on page 11-15.

Note: For applications using a J2EE data source or external connection pool, you must configure statement caching in the J2EE server's data source—not in TopLink.

Using Named Queries

Named queries improve application performance because they are prepared once and they (and all their associated supporting objects) can be efficiently reused thereafter making them well suited for frequently executed operations.

You can configure named queries at the session (see ["Configuring Named Queries at the Session Level"](#) on page 77-21) or descriptor (see ["Configuring Named Queries at the Descriptor Level"](#) on page 28-11) level.

For a session-level named query, you can execute the query using any of the following `Session` API calls:

- `executeQuery(String queryName)`
- `executeQuery(String queryName, arg1)`
- `executeQuery(String queryName, arg1, arg2)`
- `executeQuery(String queryName, arg1, arg2, arg3)`
- `executeQuery(String queryName, Vector args)`

Example 98–37 Executing a Session-Level Named Query

```
Vector args = new Vector();
args.add("Sarah");
Employee sarah = (Employee)session.executeQuery(
    "employeeReadByFirstName",
    args
);
```

For a descriptor-level named query, you can execute the query using any of the following `Session` API calls, as [Example 98-38](#) shows:

- `executeQuery(String queryName, Class domainClass)`
- `executeQuery(String queryName, Class domainClass, arg1)`
- `executeQuery(String queryName, Class domainClass, arg1, arg2)`
- `executeQuery(String queryName, Class domainClass, arg1, arg2, arg3)`
- `executeQuery(String queryName, Class domainClass, Vector args)`

Example 98-38 Executing a Descriptor Level Named Query

```
Vector args = new Vector();
args.add("Sarah");
Employee sarah = (Employee)session.executeQuery(
    "ReadByFirstName",
    Employee.class,
    args
);
```

For more information, see ["Named Queries"](#) on page 96-15

Using SQL Calls

The TopLink expression framework enables you to define complex queries at the object level. If your application requires a more complex query or one that accesses data or stored procedures directly, you can specify a custom SQL string in an `SQLCall` object and provide that `Call` object to any query.

You can also specify an SQL string directly on `DatabaseQuery`. For more information, see ["Specifying a Custom SQL String in a DatabaseQuery"](#) on page 98-16.

When using SQL calls, you can use a `ReturningPolicy` to control whether or not TopLink writes a parameter out or retrieves a value generated by the database. For more information, see ["Configuring Returning Policy"](#) on page 28-76.

This section describes the following:

- [Using an SQLCall](#)
- [Using a StoredProcedureCall](#)
- [Using a StoredFunctionCall](#)

Using an SQLCall

You can provide an `SQLCall` object to any query instead of an expression, but the SQL string contained in the `SQLCall` must return all data required to build an instance of the queried class.

The SQL string can be a complex SQL query, a stored procedure call, or a stored function call. You can specify input, output, and input/output parameters.

You can invoke an `SQLCall` through a session query method (as [Example 98-39](#) illustrates) or through a `DatabaseQuery`.

Example 98–39 Session Read Query With Custom SQL

```
List result = session.executeSelectingCall(
    new SQLCall("SELECT * FROM EMPLOYEE WHERE EMP_ID = 44"));
```

WARNING: Allowing an unverified SQL string to be passed into methods makes your application vulnerable to SQL injection attacks.

TopLink assumes that a token in the custom SQL string of an `SQLCall` is a parameter if it is prefixed with one or more number signs (`#`). You can bind values to these parameters using query API, as the following sections describe:

- [Specifying a SQLCall Input Parameter](#)
- [Specifying a SQLCall Output Parameter](#)
- [Specifying a SQLCall Input / Output Parameter](#)

Specifying a SQLCall Input Parameter

In [Example 98–40](#), you specify `last_name` as an input parameter by prefixing its name with one number sign (`#`). [Example 98–41](#) shows how to bind a value to this input parameter when you execute the query.

Example 98–40 Specifying an SQLCall with an Input Parameter Using the # Prefix

```
SQLCall sqlCall = new SQLCall(
    "INSERT INTO EMPLOYEE (L_NAME) VALUES (#last_name)");
```

Example 98–41 Executing an SQLCall with an Input Parameter

```
DataModifyQuery query = new DataModifyQuery();
query.setCall(sqlCall);
query.addArgument("last_name");    // input

Vector arguments = new Vector();
arguments.add("MacDonald");
session.executeQuery(query, arguments);
```

Specifying a SQLCall Output Parameter

In [Example 98–42](#), you specify `employee_id` as an output parameter by prefixing its name with three number signs (`###`). You specify the type of the output parameter with `SQLCall` method `setCustomSQLArgumentType`. You continue to specify `last_name` as an input parameter by prefixing its name with a number sign (`#`).

Example 98–42 Specifying a SQLCall with an Output Parameter Using the ### Prefix

```
SQLCall sqlCall = new SQLCall("begin;
    INSERT INTO EMPLOYEE (L_NAME) VALUES (#L_NAME) RETURNING EMP_ID INTO ###employee_id;
end");
sqlCall.setCustomSQLArgumentType("employee_id", Integer.class);
```

Example 98–43 Executing a SQLCall with an Output Parameter

```
ValueReadQuery query = new ValueReadQuery();
query.setCall(sqlCall);
```

```
query.addArgument("last_name"); // input

Vector args = new Vector();
args.add("MacDonald");

Number employeeID = getSession().executeQuery(query, args);
```

You can also obtain results for an output parameter declared to be of type `CURSOR`.

Specifying a SQLCall Input / Output Parameter

In [Example 98–44](#), you specify `in_out` as an input and output parameter by prefixing its name with four number signs (####). The type of the input value determines the type of the output value. In this example, a `String` ("MacDonald") is passed in and the output value (for `EMP_ID`) is returned as a `String`.

Example 98–44 Specifying an Input and Output Parameter Using the #### Prefix

```
SQLCall sqlCall = new SQLCall(
    "INSERT INTO EMPLOYEE (L_NAME) VALUES (####in_out) RETURNING EMP_ID INTO ####in_out");
```

Example 98–45 Executing a SQLCall with an Input and Output Parameter

```
ValueReadQuery query = new ValueReadQuery();
query.setCall(sqlCall);
query.addArgument("in_out"); // input and output

Vector args = new Vector();
args.add("MacDonald");

Number employeeID = getSession().executeQuery(query, args);
```

Using a StoredProcedureCall

You can provide a `StoredProcedureCall` object to any query instead of an expression or a SQL string, but the procedure must return all data required to build an instance of the class you query.

Example 98–46 A Read-All Query with a Stored Procedure

```
ReadAllQuery readAllQuery = new ReadAllQuery();
call = new StoredProcedureCall();
call.setProcedureName("Read_All_Employees");
call.useNamedCursorOutputAsResultSet("resultSet");
readAllQuery.setCall(call);
Vector employees = (Vector) session.executeQuery(readAllQuery);
```

Using a `StoredProcedureCall`, you can access the following:

- [Specifying an Input Parameter](#)
- [Specifying an Output Parameter](#)
- [Specifying an Input / Output Parameter](#)
- [Using an Output Parameter Event](#)

Note: You no longer need to use `DatabaseQuery` method `bindAllParameters` when using a `StoredProcedureCall` with `OUT` or `INOUT` parameters. However, you should always specify the Java type for all `OUT` and `INOUT` parameters. If you do not, be aware of the fact that they default to type `String`.

Specifying an Input Parameter

In [Example 98–48](#), you specify the parameter `POSTAL_CODE` as an input parameter using the `StoredProcedureCall` method `addNamedArgument`, and you can specify the value of the argument using method `addNamedArgumentValue`.

Example 98–47 *Stored Procedure Call with an Input Parameter*

```
StoredProcedureCall call = new StoredProcedureCall();
call.setProcedureName("CHECK_VALID_POSTAL_CODE");
call.addNamedArgument("POSTAL_CODE");
call.addNamedArgumentValue("POSTAL_CODE", "L5J1H5");
call.addNamedOutputArgument(
    "IS_VALID",    // procedure parameter name
    "IS_VALID",    // out argument field name
    Integer.class // Java type corresponding to type returned by procedure
);
ValueReadQuery query = new ValueReadQuery();
query.setCall(call);
Number isValid = (Number) session.executeQuery(query);
```

Specifying an Output Parameter

Output parameters enable the stored procedure to return additional information. You can use output parameters to define a `ReadObjectQuery` if they return all the fields required to build the object.

In [Example 98–49](#), you specify the parameter `IS_VALID` as an output parameter using the `StoredProcedureCall` method `addNamedOutputArgument`.

Example 98–48 *Stored Procedure Call with an Output Parameter*

```
StoredProcedureCall call = new StoredProcedureCall();
call.setProcedureName("CHECK_VALID_POSTAL_CODE");
call.addNamedArgument("POSTAL_CODE");
call.addNamedOutputArgument(
    "IS_VALID",    // procedure parameter name
    "IS_VALID",    // out argument field name
    Integer.class // Java type corresponding to type returned by procedure
);
ValueReadQuery query = new ValueReadQuery();
query.setCall(call);
query.addArgument("POSTAL_CODE");
Vector parameters = new Vector();
parameters.addElement("L5J1H5");
Number isValid = (Number) session.executeQuery(query, parameters);
```

Note: Not all databases support the use of output parameters to return data. However, because these databases generally support returning result sets from stored procedures, they do not require output parameters.

If you are using an Oracle database, you can make use of TopLink cursor and stream query results.

Specifying an Input / Output Parameter

In [Example 98–50](#), you specify the parameter `LENGTH` as an input/output parameter and specify the value of the argument when it is passed to the stored procedure using the `StoredProcedureCall` method `addNamedInOutArgumentValue`. If you do not want to specify a value for the argument, use method `addNamedInOutArgument`.

Example 98–49 *Stored Procedure Call with an Input/Output Parameter*

```
StoredProcedureCall call = new StoredProcedureCall();
call.setProcedureName("CONVERT_FEET_TO_METERS");
call.addNamedInOutArgumentValue(
    "LENGTH",          // procedure parameter name
    new Integer(100),  // in argument value
    "LENGTH",          // out argument field name
    Integer.class       // Java type corresponding to type returned by procedure
)
ValueReadQuery query = new ValueReadQuery();
query.setCall(call);
Integer metricLength = (Integer) session.executeQuery(query);
```

Using an Output Parameter Event

TopLink manages output parameter events for databases that support them. For example, if a stored procedure returns an error code that indicates that the application wants to check for an error condition, TopLink raises the session event `OutputParametersDetected` to allow the application to process the output parameters.

Example 98–50 *Stored Procedure with Reset Set and Output Parameter Error Code*

```
StoredProcedureCall call = new StoredProcedureCall();
call.setProcedureName("READ_EMPLOYEE");
call.addNamedArgument("EMP_ID");
call.addNamedOutputArgument(
    "ERROR_CODE", // procedure parameter name
    "ERROR_CODE", // out argument field name
    Integer.class // Java type corresponding to type returned by procedure
);
call.useNamedCursorOutputAsResultSet("resultSet");
ReadObjectQuery query = new ReadObjectQuery();
query.setCall(call);
query.addArgument("EMP_ID");
ErrorCodeListener listener = new ErrorCodeListener();
session.getEventManager().addListener(listener);
Vector args = new Vector();
args.addElement(new Integer(44));
Employee employee = (Employee) session.executeQuery(query, args);
```

Using a StoredFunctionCall

You use a `StoredProcedureCall` to invoke stored procedures defined on databases that support them. You can also use a `StoredFunctionCall` to invoke stored functions defined on databases that support them, that is, on databases for which the `DatabasePlatform` method `supportsStoredFunctions` returns `true`.

In general, both stored procedures and stored functions let you specify input parameters, output parameters, and input and output parameters. For more information, see ["Using a StoredProcedureCall"](#) on page 98-21. However, stored procedures need not return values, while stored functions always return a single value.

The `StoredFunctionCall` class extends `StoredProcedureCall` to add one new method: `setResult`. Use this method to specify the name (and alternatively both the name and type) under which `TopLink` stores the return value of the stored function.

When `TopLink` prepares a `StoredFunctionCall`, it validates its SQL and throws a `ValidationException` under the following circumstances:

- If your current platform does not support stored functions. Stored functions are supported only for Oracle.
- If you fail to specify the return type

In [Example 98-52](#), note that the name of the stored function is set using `StoredFunctionCall` method `setProcedureName`.

Example 98-51 Creating a StoredFunctionCall

```
StoredFunctionCall functionCall = new StoredFunctionCall();
functionCall.setProcedureName("CHECK_VALID_EMPLOYEE");
functionCall.addNamedArgument("EMP_ID");
functionCall.setResult("FUNCTION_RESULT", String.class);
ValueReadQuery query = new ValueReadQuery();
query.setCall(functionCall);
query.addArgument("EMP_ID");
Vector args = new Vector();
args.addElement(new Integer(44));
String valid = (String) session.executeQuery(query, args);
```

Using EJB QL Calls

The `TopLink` expression framework lets you define complex queries at the object level. Alternatively, you can specify a custom EJB QL string in an `EJB QL Call` object and provide that `Call` object to any query.

You can also specify an EJB QL string directly in a `DatabaseQuery`. For more information, see ["Specifying a Custom EJB QL String in a DatabaseQuery"](#) on page 98-16.

You can provide an `EJBQLCall` object to any query instead of an expression or EJB QL string, but the procedure must return all data required to build an instance of the class you query.

You can invoke EJB QL queries through the session query methods or through a `DatabaseQuery`.

Example 98-52 Session Read Query With Custom EJB QL

```
Vector theObjects = (Vector)aSession.readAllObjects(
    EmployeeBean.class,
    new EJBQLCall("SELECT OBJECT (emp) from EmployeeBean emp"));
```

Using EIS Interactions

For an EIS root descriptor, you can define EIS interactions to invoke methods on an EIS.

TopLink represents EIS interactions using instances of `oracle.toplink.eis.interactions.EISInteraction`. These classes implement the `Call` interface and can be used wherever a `Call` can be used.

[Table 98–1](#) lists the type of EIS interactions that TopLink supports.

Table 98–1 *EIS Interactions*

EIS Interaction Type	Description
<code>IndexedInteraction</code>	Defines the specification for a call to a J2C interaction that uses indexed records. Builds the input and output records from the arguments by position.
<code>MappedInteraction</code>	Defines the specification for a call to a J2C interaction that uses mapped records. Builds the input and output records from the arguments by name.
<code>XMLInteraction</code>	Specifies an instance of <code>MappedInteraction</code> that defines the specification for a call to a J2C interaction that uses XML records defined by the XML schema document (XSD) associated with the EIS project (for more information, see "Importing an XML Schema" on page 4-34).
<code>QueryStringInteraction</code>	Specifies an instance of <code>MappedInteraction</code> that defines the specification for a call to a J2C interaction that uses a query string. Prefix arguments in the query string with a number sign (#) character.
<code>XQueryInteraction</code>	Specifies an instance of <code>XMLInteraction</code> that defines the specification for a call to a J2C interaction that uses XQuery. Translates the XQuery from the query arguments.

You can use TopLink to define an interaction for each basic persistence operation (`insert`, `update`, `delete`, `read object`, `read all`, or `does exist`) so that when you query and modify your EIS-mapped objects, the TopLink runtime will use the appropriate EIS interaction. For more information, see ["Configuring Custom EIS Interactions for Basic Persistence Operations"](#) on page 31-6.

You can also use TopLink to define an interaction as a named query for `read object` and `read-all` object queries. These queries are not called for basic persistence operations; you can call these additional queries by name in your application for special purposes. For more information, see ["Creating an EIS Interaction for a Named Query"](#) on page 28-24.

Handling Exceptions

Most exceptions in queries are database exceptions, resulting from a failure in the database operation (see [Database Exceptions \(4002 – 4018\)](#) section on page 13-27 in the [TopLink Exception Reference](#) chapter for more information). Write operations can also throw an `OptimisticLockException` on a write, update, or delete operation in applications that use optimistic locking. To catch these exceptions, execute all database operations within a try-catch block:

```
try {
    Vector employees = session.readAllObjects(Employee.class);
}
catch (DatabaseException exception) {
```

```
// handle exception
}
```

See [Chapter 14, "TopLink Workbench Error Reference"](#) for more information about exceptions in a TopLink application.

Handling Collection Query Results

TopLink provides a `useCollectionClass` method to all subclasses of `DataReadQuery` and `ReadAllQuery`, that you can use to configure a query to return results as any concrete instance of `Collection` or `Map`.

Do not confuse collection query result configuration with a mapping container policy (see ["Configuring Container Policy"](#) on page 35-26): there is no relationship between the two. Collection query result configuration determines how TopLink returns multiobject results from a particular query. A mapping container policy tells TopLink how your domain object implements a data member that contains a collection.

For example, consider a class `Employee` with a data member `phoneNumbers`. In your implementation of `Employee`, the `getPhoneNumbers` method returns a `Vector`. Using TopLink Workbench, you map the `phoneNumbers` data member as a one-to-many mapping. You configure the mapping container policy so that the mapping contains its value (many `PhoneNumber` objects) in a `Vector`. This corresponds to your implementation of `Employee`.

You define a `ReadAllQuery` named `localPhoneNumbers` on the `DescriptorQueryManager` of the `PhoneNumber`. The `localPhoneNumbers` query takes one argument, the ID of an `Employee` object, and returns all the phone numbers from its `phoneNumbers` data member whose area code is 613.

You get this query by name from the `DescriptorQueryManager` for `PhoneNumber`. You call the `useCollectionClass` method on this `ReadAllQuery`, passing in the `ArrayList` class. You execute the query, passing in the ID of an `Employee`. The query returns all the `PhoneNumber` objects from the `Employee` object's `phoneNumbers` data member whose area code is 613. The query returns these results as an `ArrayList`.

Handling Report Query Results

[Table 98-2](#) lists the `ReportQuery` methods you can use to configure how a `ReportQuery` returns its results.

Table 98-2 *Report Query Result Options*

Method	Query Returns	Description
<code>setShouldReturnSingleAttribute</code>	<code>DatabaseRow</code>	Returns a single attribute (not wrapped in a <code>ReportQueryResult</code>). Use this option if you know that the <code>ReportQuery</code> returns only one attribute.
<code>setShouldReturnSingleResult</code>	<code>ReportQueryResult</code>	Returns only the first <code>ReportQueryResult</code> object (not wrapped in a <code>Collection</code> or <code>Map</code>). Use this option if you know that the <code>ReportQuery</code> returns only one row.
<code>setShouldReturnSingleValue</code>	<code>Object</code>	Returns only a single value. Use this option if you know that the <code>ReportQuery</code> returns only one row that contains only one attribute.

For more information, see the following:

- ["Report Query"](#) on page 96-15
- ["Report Query Results"](#) on page 96-8
- ["Reading Objects Using Report Queries"](#) on page 98-6
- ["Configuring Named Query Attributes"](#) on page 28-20

Understanding TopLink Expressions

Using the TopLink expressions framework, you can specify query search criteria based on your domain object model. This section describes the following:

- [Understanding the Expression Framework](#)
- [Expression Components](#)
- [Parameterized Expressions](#)
- [Query Keys and Expressions](#)
- [Using Multiple Expressions](#)
- [Data Queries and Expressions](#)
- [Creating an Expression](#)
- [Creating and Using a User-Defined Function](#)

Understanding the Expression Framework

The TopLink expression framework provides methods through the following classes:

- The `Expression` class provides most general functions, such as `toUpperCase`.
- The `ExpressionMath` class supplies mathematical methods.

[Example 97-1](#) illustrates how to use the `Expression` class.

Example 97-1 Using the Expression Class

```
expressionBuilder.get("lastName").equal("Smith");
```

[Example 97-2](#) illustrates how to use the `ExpressionMath` class.

Example 97-2 Using the ExpressionMath Class

```
ExpressionMath.abs(ExpressionMath.subtract(emp.get("salary"),  
emp.get("spouse").get("salary")).greaterThan(10000)
```

This division of functionality enables TopLink expressions to provide similar mathematical functionality to the `java.lang.Math` class, but keeps both the `Expression` and `ExpressionMath` classes from becoming unnecessarily complex.

Expressions Compared to SQL

Expressions offer the following advantages over SQL when you access a database:

- Expressions are easier to maintain because the database is abstracted.

- Changes to descriptors or database tables do not affect the querying structures in the application.
- Expressions enhance readability by standardizing the `Query` interface so that it looks similar to traditional Java calling conventions. For example, the Java code required to get the street name from the `Address` object of the `Employee` class looks like this:

```
emp.getAddress().getStreet().equals("Meadowlands");
```

The expression to get the same information is similar:

```
emp.get("address").get("street").equal("Meadowlands");
```

- Expressions allow read queries to transparently query between two classes that share a relationship. If these classes are stored in multiple tables in the database, `TopLink` automatically generates the appropriate join statements to return information from both tables.
- Expressions simplify complex operations. For example, the following Java code retrieves all employees that live on "Meadowlands" whose salary is greater than 10,000:

```
ExpressionBuilder emp = new ExpressionBuilder();
Expression exp = emp.get("address").get("street").equal("Meadowlands");
Vector employees = session.readAllObjects(Employee.class,
    exp.and(emp.get("salary").greaterThan(10000)));
```

`TopLink` automatically generates the appropriate SQL from that code:

```
SELECT t0.VERSION, t0.ADDR_ID, t0.F_NAME, t0.EMP_ID, t0.L_NAME, t0.MANAGER_ID,
t0.END_DATE, t0.START_DATE, t0.GENDER, t0.START_TIME, t0.END_TIME, t0.SALARY
FROM EMPLOYEE t0, ADDRESS t1 WHERE (((t1.STREET = 'Meadowlands') AND (t0.SALARY
> 10000)) AND (t1.ADDRESS_ID = t0.ADDR_ID))
```

Expression Components

A simple expression usually consists of three parts:

1. The *attribute*, which represents a mapped attribute or query key of the persistent class
2. The *operator*, which is an expression method that implements boolean logic, such as `GreaterThan`, `Equal`, or `Like`
3. The *constant* or *comparison*, which refers to the value used to select the object

In the following code fragment:

```
expressionBuilder.get("lastName").equal("Smith");
```

- The attribute is `lastName`.
- The operator is `equal`.
- The constant is the string `"Smith"`.

The `expressionBuilder` substitutes for the object or objects to be read from the database. In this example, `expressionBuilder` represents employees.

You can use the following components when constructing an `Expression`:

- [Boolean Logic](#)
- [Database Functions](#)

- [Mathematical Functions](#)
- [XMLType Functions](#)
- [Platform and User-Defined Functions](#)
- [Expressions for One-to-One and Aggregate Object Relationships](#)
- [Expressions for Joining and Complex Relationships](#)

Boolean Logic

Expressions use standard boolean operators, such as AND, OR, and NOT, and you can combine multiple expressions to form more complex expressions. [Example 97-3](#) illustrates a code fragment that queries for projects managed by a selected person, and that have a budget greater than or equal to 1,000,000.

Example 97-3 Using Boolean Logic in an Expression

```
ExpressionBuilder project = new ExpressionBuilder();
Expression hasRightLeader, bigBudget, complex;
Employee selectedEmp = someWindow.getSelectedEmployee();
hasRightLeader = project.get("teamLeader").equal(selectedEmp);
bigBudget = project.get("budget").greaterThanEqual(1000000);
complex = hasRightLeader.and(bigBudget);
Vector projects = session.readAllObjects(Project.class, complex);
```

Database Functions

TopLink expressions support a variety of database functions, including, but not limited to, the following:

- `like`
- `notLike`
- `equal`
- `notEqual`
- `toUpperCase`
- `toLowerCase`
- `toDate`
- `rightPad`
- `rightTrim`

Note: Some functions may be database platform specific.

Database functions let you define more flexible queries. [Example 97-4](#) illustrates a code fragment that matches several last names, including "SMART", "Smith", and "Smothers":

Example 97-4 Using a Database Function Supported by the Expression API

```
emp.get("lastName").toUpperCase().like("SM%")
```

You access most functions using Expression methods such as `toUpperCase`.

Some functions have very specific purpose: you can use `ascending` and `descending` functions only within an ordering expression to place the result in ascending or descending order, as [Example 97-5](#) shows:

Example 97-5 Using an Ordering Database Function

```
readAllQuery.addOrderBy(expBuilder.get("address").get("city").ascending())
```

You can use aggregate functions, such as `average`, `minimum`, `maximum`, `sum` and so forth, with `ReportQuery` (see ["Report Query"](#) on page 96-15).

Mathematical Functions

Mathematical functions are available through the `ExpressionMath` class. Mathematical function support in expressions is similar to the support provided by the Java class `java.lang.Math`. [Example 97-6](#) illustrates using the `abs` and `subtract` methods.

Example 97-6 Using Mathematical Functions in an Expression

```
ExpressionMath.abs(ExpressionMath.subtract(emp.get("salary"), emp.get("spouse")  
    .get("salary")).greaterThan(10000)
```

XMLType Functions

You can use the following operators when constructing queries against data mapped to an Oracle Database `XMLType` column:

- `extract`: Takes an XPath string and returns an `XMLType` which corresponds to the part of the original document that matches the XPath.
- `extractValue`: Takes an XPath string and returns either a numerical or string value based on the contents of the node pointed to by the XPath.
- `existsNode`: Takes an XPath expression and returns the number of nodes that match the XPath.
- `getStringVal`: Gets the string representation of an `XMLType` object.
- `getNumberVal`: Gets the numerical representation of an `XMLType` object.

[Example 97-7](#) illustrates how to use the `extract` operator in a query:

Example 97-7 Using the XMLType Extract Operator

```
Expression criteria =  
builder.get("resume").extract("//education/degree/text()").getStringVal().equal("BCS");  
Vector employees = session.readAllObject(Employee.class, criteria);
```

Platform and User-Defined Functions

You can use the `Expression` method `getFunction` to access database functions that `TopLink` does not support directly. [Example 97-8](#) illustrates how to access a database function named `VacationCredit` from within an expression, even though there is no support for such a function in the `Expression` API.

Example 97-8 Using a Database Function Not Supported by the Expression API

```
emp.get("lastName").getFunction("VacationCredit").greaterThan(42)
```

This expression produces the following SQL:

```
SELECT . . . WHERE VacationCredit(EMP.LASTNAME) > 42
```

The Expression API includes additional forms of the `getFunction` method that allow you to specify arguments. For more information, see *Oracle TopLink API Reference*.

You can also access a custom function that you create. For more information on creating a custom function in TopLink, see ["Creating and Using a User-Defined Function"](#) on page 97-17.

Expressions for One-to-One and Aggregate Object Relationships

Expressions can include an attribute that has a one-to-one relationship with another persistent class. A one-to-one relationship translates naturally into a SQL join that returns a single row.

[Example 97-9](#) illustrates a code fragment that accesses fields from an employee's address.

Example 97-9 Using an Expression with a One-to-One Relationship

```
emp.get("address").get("country").like("%S%")
```

[Example 97-9](#) corresponds to joining the EMPLOYEE table to the ADDRESS table, based on the address foreign key, and checking for the country name.

You can nest these relationships infinitely, so it is possible to ask for complex information as follows:

```
project.get("teamLeader").get("manager").get("manager").get("address").get("street")
```

Expressions for Joining and Complex Relationships

You can query against complex relationships, such as one-to-many, many-to-many, direct collection, and aggregate collection relationships. Expressions for these types of relationships are more complex to build, because the relationships do not map directly to joins that yield a single row per object.

This section describes the following:

- [Understanding Joins](#)
- [Using TopLink Expression API For Joins](#)

Understanding Joins

A **join** is a relational database query that combines rows from two or more tables. Relational databases perform a join whenever multiple tables appear in the query's FROM clause. The query's select list can select any columns from any of these tables.

An inner join (sometimes called a "simple join") is a join of two or more tables that returns only those rows that satisfy the join condition.

An outer join extends the result of an inner join. An outer join returns all rows that satisfy the join condition and also returns some or all of those rows from one table for which no rows from the other satisfy the join condition. Outer joins can be categorized as left or right:

- A query that performs a left outer join of tables A and B returns all rows from A. For all rows in A that have no matching rows in B, the database returns null for any select list expressions containing columns of B.
- A query that performs a right outer join of tables A and B returns all rows from B. For all rows in B that have no matching rows in A, the database returns null for any select list expressions containing columns of A.

When you query with a join expression, TopLink can use joins to check values from other objects or other tables that represent parts of the same object. Although this works well under most circumstances, it can cause problems when you query against a one-to-one relationship, in which one side of the relationship is not present.

For example, `Employee` objects may have an `Address` object, but if the `Address` is unknown, it is `null` at the object level and has a null foreign key at the database level. When you attempt a read that traverses the relationship, missing objects cause the query to return unexpected results. Consider the following expression:

```
(emp.get("firstName").equal("Steve")).or(emp.get("address").get("city").equal("Ottawa"))
```

In this case, employees with no address do not appear in the result set, regardless of their first name. Although not obvious at the object level, this behavior is fundamental to the nature of relational databases.

Outer joins rectify this problem in the databases that support them. In this example, the use of an outer join provides the expected result: all employees named Steve appear in the result set, even if their address is unknown.

To implement an outer join, use Expression method `getAllowingNull`, rather than `get`, and Expression method `anyOfAllowingNone`, rather than `anyOf`.

For example:

```
(emp.get("firstName").equal("Steve")).or(emp.getAllowingNull("address").get("city").equal("Ottawa"))
```

Support and syntax for outer joins vary widely between databases and database drivers. TopLink supports outer joins for Oracle databases, IBM DB2, SQL Anywhere, Microsoft Access, Microsoft SQL Server, Sybase SQL Server, and the JDBC outer join syntax.

Using TopLink Expression API For Joins

You can use joins anywhere expressions are used, including: selection-criteria, ordering (see ["Specifying Read Ordering"](#) on page 98-8), report queries (see ["Report Query"](#) on page 96-15), partial objects (see ["Partial Object Queries"](#) on page 96-11), one-to-one relational mappings (see ["Configuring Joining at the Mapping Level"](#) on page 40-1), and join reading (see ["Join Reading and Object-Level Read Queries"](#) on page 96-12).

Use the expression API shown in [Table 97-1](#) to configure inner and outer join expressions.

Table 97-1 Expression API for Joins

Expression API	Type of Join	Type of Mapping
<code>get</code>	inner	one-to-one
<code>getAllowingNull</code>	outer	one-to-one
<code>anyOf</code>	inner	one-to-many, many-to-many
<code>anyOfAllowingNone</code>	outer	one-to-many, many-to-many

To query across a one-to-many or many-to-many relationship, use the `anyOf` operation. As its name suggests, this operation supports queries that return all items on the "many" side of the relationship that satisfy the query criteria.

[Example 97-10](#) illustrates an expression that returns employees who manage at least one employee (through a one-to-many relationship) with a salary less than \$10,000.

Example 97-10 Using an Expression with a One-to-Many Relationship

```
emp.anyOf("managedEmployees").get("salary").lessThan(10000);
```

[Example 97-11](#) illustrates how to query across a many-to-many relationship using a similar strategy:

Example 97-11 Using an Expression with a Many-to-Many Relationship

```
emp.anyOf("projects").equal(someProject)
```

TopLink translates these queries into SQL that joins the relevant tables using a `DISTINCT` clause to remove duplicates. TopLink translates [Example 97-10](#) into the following SQL:

```
SELECT DISTINCT . . . FROM EMP t1, EMP t2 WHERE
t2.MANAGER_ID = t1.EMP_ID AND t2.SALARY < 10000
```

You can use one-to-one and one-to-many join expressions in an `ObjectLevelReadyQuery` to configure joins on a per-query basis (see ["Join Reading and Object-Level Read Queries"](#) on page 96-12).

You can also configure joins at the mapping level (see ["Configuring Joining at the Mapping Level"](#) on page 40-1).

Note: Calling `anyOf` once would result in a different outcome than if you call it twice. For example, if you query for an employee with a telephone area code of 613 and a number of 123-4599, you would use a single `anyOf` and a temporary variable. If you query for an employee, whose telephone has an area code of 613, and whose telephone has a number of 123-4599, you would call `anyOf` twice.

Parameterized Expressions

A relationship mapping differs from a regular query because it retrieves data for many different objects. To be able to specify these queries, supply arguments when you execute the query. Use the `getParameter` and `getField` methods to acquire values for the arguments.

A parameterized expression executes searches and comparisons based on variables instead of constants. This approach lets you build expressions that retrieve context-sensitive information. This technique is useful when you define EJB finders (see ["EJB Finders"](#) on page 96-24).

Parameterized expressions require that the relationship mapping know how to retrieve an object or collection of objects based on its current context. For example, a one-to-one mapping from `Employee` to `Address` must query the database for an address based on foreign key information from the `Employee` table. Each mapping contains a query that TopLink constructs automatically based on the information

provided in the mapping. To specify expressions yourself, use the mapping customization mechanisms.

You can use parameterized expressions to create reusable queries (see ["Named Queries"](#) on page 96-16).

Expression Method `getParameter`

The Expression method `getParameter` returns an expression that becomes a parameter in the query. This lets you create a query that includes user input in the search criteria. The parameter must be either the fully qualified name of the field from a descriptor's row, or a generic name for the argument.

Parameters you construct this way are global to the current query, so you can send this message to any expression object.

[Example 97-12](#) illustrates how to use a custom query to find an employee by first name.

Example 97-12 Using a Parameterized Expression in a Custom Query

```
ExpressionBuilder emp = new ExpressionBuilder();
Expression firstNameExpression;
firstNameExpression = emp.get("firstName").equal(emp.getParameter("firstName"));
ReadObjectQuery query = new ReadObjectQuery();
query.setReferenceClass(Employee.class);
query.setSelectionCriteria(firstNameExpression);
query.addArgument("firstName");
Vector v = new Vector();
v.addElement("Sarah");
Employee e = (Employee) session.executeQuery(query, v);
```

[Example 97-13](#) illustrates how to use a custom query to find all employees that live in the same city as a given employee.

Example 97-13 Using Nested Parameterized Expressions

```
ExpressionBuilder emp = new ExpressionBuilder();
Expression addressExpression;
addressExpression =
emp.get("address").get("city").equal(emp.getParameter("employee").get("address").get("city"));
ReadObjectQuery query = new ReadObjectQuery(Employee.class);
query.setName("findByCity");
query.setReferenceClass(Employee.class);
query.setSelectionCriteria(addressExpression);
query.addArgument("employee");
Vector v = new Vector();
v.addElement(employee);
Employee e = (Employee) session.executeQuery(query, v);
```

[Example 97-14](#) illustrates how to obtain a simple one-to-many mapping from class `PolicyHolder` to `Policy` using a nondefault selection criteria. The SSN field of the `POLICY` table is a foreign key to the SSN field of the `HOLDER` table.

Example 97-14 Using a Parameterized Expression in a Mapping

```
OneToManyMapping mapping = new OneToManyMapping();
mapping.setAttributeName("policies");
mapping.setGetMethodName("getPolicies");
```

```
mapping.setSetMethodName("setPolicies");
mapping.setReferenceClass(Policy.class);

// Build a custom expression here rather than using the defaults
ExpressionBuilder policy = new ExpressionBuilder();
mapping.setSelectionCriteria(policy.getField("POLICY.SSN").equal(policy.
    getParameter("HOLDER.SSN")));
```

Expression Method getField

The Expression method `getField` returns an expression that represents a database field with the given name. Use this method to construct the selection criteria for a mapping. The argument is the fully qualified name of the required field. Because fields are not global to the current query, you must send this method to an expression that represents the table from which this field is derived. See also ["Data Queries and Expressions"](#) on page 97-12.

[Example 97-15](#) illustrates how to use the Expression method `getField`.

Example 97-15 Using Expression Method getParameter

```
ExpressionBuilder address = new ExpressionBuilder();
Expression exp = address.getField("ADDRESS.EMP_ID").equal(
    address.getParameter("EMPLOYEE.EMP_ID")
);
exp = exp.and(address.getField("ADDRESS.TYPE").equal(null));
```

Query Keys and Expressions

A query key is a schema-independent alias for a database field name.

Query keys are supported in relational database projects only.

Query keys are generated automatically for all direct and relationship mappings. The name of the query key is the class attribute name.

For more information on how query keys are created and modified, see ["Configuring Query Keys"](#) on page 28-35.

[Example 97-16](#) illustrates how to use the query key `firstName` for the corresponding directly mapped Employee attribute.

Example 97-16 Using an Automatically Generated Query Key in an Expression

```
Vector employees = session.readAllObjects(Employee.class,
    new ExpressionBuilder().get("firstName").equal("Bob"));
```

[Example 97-17](#) illustrates how to use a one-to-one query key within the TopLink expression framework.

Example 97-17 Using a One-to-One Query Key in an Expression

```
ExpressionBuilder employee = new ExpressionBuilder();
Vector employees = session.readAllObjects(Employee.class,
    employee.get("address").get("city").equal("Ottawa"));
```

To access one-to-many and many-to-many query keys that define a distinct join across a collection relationship, use Expression method `anyOf`.

Using Multiple Expressions

Expressions support subqueries (SQL subselects) and parallel selects. To create a subquery, use a single expression builder. With parallel selects, use multiple expression builders when you define a single query. This lets you specify joins for unrelated objects at the object level.

Subselects and Subqueries

Some queries compare the results of other, contained queries (or subqueries). SQL supports this comparison through subselects. TopLink expressions provide subqueries to support subselects.

Subqueries lets you define complex expressions that query on aggregated values (counts, min, max) and unrelated objects (exists, in, comparisons). To obtain a subquery, pass an instance of a report query to any expression comparison operation, or use the `subQuery` operation on an expression builder. The subquery is not required to have the same reference class as the parent query, and it must use its own expression builder.

You can nest subqueries, or use them in parallel. Subqueries can also make use of custom SQL.

For expression comparison operations that accept a single value (`equal`, `greaterThan`, `lessThan`), the subquery result must return a single value. For expression comparison operations that accept a set of values (`in`, `exists`), the subquery result must return a set of values.

[Example 97-18](#) illustrates how to create an expression that matches all employees with more than five managed employees.

Example 97-18 A Subquery Expression Using a Comparison and Count Operation

```
ExpressionBuilder emp = new ExpressionBuilder();
ExpressionBuilder managedEmp = new ExpressionBuilder();
ReportQuery subQuery = new ReportQuery(Employee.class, managedEmp);
subQuery.addCount();
subQuery.setSelectionCriteria(managedEmp.get("manager").equal(emp));
Expression exp = emp.subQuery(subQuery).greaterThan(5);
```

[Example 97-19](#) illustrates how to create an expression that matches the employee with the highest salary in the city of Ottawa.

Example 97-19 A Subquery Expression Using a Comparison and Max Operation

```
ExpressionBuilder emp = new ExpressionBuilder();
ExpressionBuilder ottawaEmp = new ExpressionBuilder();
ReportQuery subQuery = new ReportQuery(Employee.class, ottawaEmp);
subQuery.addMax("salary");
subQuery.setSelectionCriteria(ottawaEmp.get("address").get("city").equal("Ottawa"));
Expression exp =
    emp.get("salary").equal(subQuery).and(emp.get("address").get("city").equal("Ottawa"));
```

[Example 97-20](#) illustrates how to create an expression that matches all employees that have no projects.

Example 97-20 A Subquery Expression Using a Not Exists Operation

```
ExpressionBuilder emp = new ExpressionBuilder();
ExpressionBuilder proj = new ExpressionBuilder();
ReportQuery subQuery = new ReportQuery(Project.class, proj);
```



```
subQuery.addAttribute("id");
subQuery.setSelectionCriteria(proj.equal(emp.anyOf("projects")));
Expression exp = emp.notExists(subQuery);
```

Parallel Expressions

Parallel expressions enable you to compare unrelated objects. Parallel expressions require multiple expression builders, but do not require the use of report queries. Each expression must have its own expression builder, and you must use the constructor for expression builder that takes a `class` as an argument. The class does not have to be the same for the parallel expressions, and you can create multiple parallel expressions in a single query.

Only one of the expression builders is considered the primary expression builder for the query. This primary builder makes use of the zero argument expression constructor, and `TopLink` obtains its class from the query.

[Example 97–21](#) illustrates how to create an expression that matches all employees with the same last name as another employee of different gender, and accounts for the possibility that returned results could be a spouse.

Example 97–21 A Parallel Expression on Two Independent Employees

```
ExpressionBuilder emp = new ExpressionBuilder();
ExpressionBuilder spouse = new ExpressionBuilder(Employee.class);
Expression exp = emp.get("lastName").equal(spouse.get("lastName"))
    .and(emp.get("gender").notEqual(spouse.get("gender")));
```

Data Queries and Expressions

You can use expressions to retrieve data rather than objects. This is a common approach when you work with unmapped information in the database, such as foreign keys and version fields.

Expressions that query for objects generally refer to object attributes, which may in turn refer to other objects. Data expressions refer to tables and their fields. You can combine data expressions and object expressions within a single query. `TopLink` provides two main methods for expressions that query for data: `getField` and `getTable`.

getField

The `getField` method lets you retrieve data from either an unmapped table or an unmapped field from an object. In either case, the field must be part of a table represented by that object's class; otherwise, `TopLink` raises an exception when you execute the query.

You can also use the `getField` method to retrieve the foreign key information for an object.

[Example 97–22](#) illustrates how to use the data expression method (operator) `getField` with an object.

Example 97–22 Using getField with an Object

```
bUILDER.getField("[FIELD_NAME]").greaterThan("[ARGUMENT]");
```

getTable

The `getTable` method returns an expression that represents an unmapped table in the database. This expression provides a context from which to retrieve an unmapped field when you use the `getField` method.

[Example 97-23](#) illustrates how to combine both `getField` and `getTable` in the same expression.

Example 97-23 Using getTable and getField Together

```
builder.getTable("[TABLE_NAME]").getField("[FIELD_NAME]").equal("[ARGUMENT]");
```

A common use for the `getTable` and `getField` methods is to retrieve information from a link table (or reference table) that supports a many-to-many relationship.

[Example 97-24](#) reads a many-to-many relationship that uses a link table and also checks an additional field in the link table. This code combines an object query with a data query, using the employee's manager as the basis for the data query. It also features parameterization for the project ID.

Example 97-24 Using a Data Query Against a Link Table

```
ExpressionBuilder emp = new ExpressionBuilder();
Expression manager = emp.get("manager");
Expression linkTable = manager.getTable("PROJ_EMP");
Expression empToLink = emp.getField("EMPLOYEE
    .EMP_ID").equal(linkTable.getField("PROJ_EMP.EMP_ID");
Expression projToLink = linkTable.getField("PROJ_EMP
    .PROJ_ID").equal(emp.getParameter("PROJECT.PROJ_ID"));
Expression extra = linkTable.getField("PROJ_EMP.TYPE").equal("W");
query.setSelectionCriteria((empToLink.and(projToLink)).and(extra));
```

Creating an Expression

You can create an expression using TopLink Workbench or Java code.

Use TopLink Workbench for creating basic expressions for use in named queries (see ["Using TopLink Workbench"](#) on page 97-13).

Use Java code to create more complex expressions and to take full advantage of the features in the expressions API (see ["Using Java"](#) on page 97-16).

Using TopLink Workbench

To create TopLink expressions for named queries, use this procedure:

1. From the **Named Queries Format** tab, click **Edit** (or double-click a query string). The Expression Builder dialog box appears.

See ["Named Queries"](#) on page 96-16 for more information.

Figure 97–1 Expression Builder Dialog Box

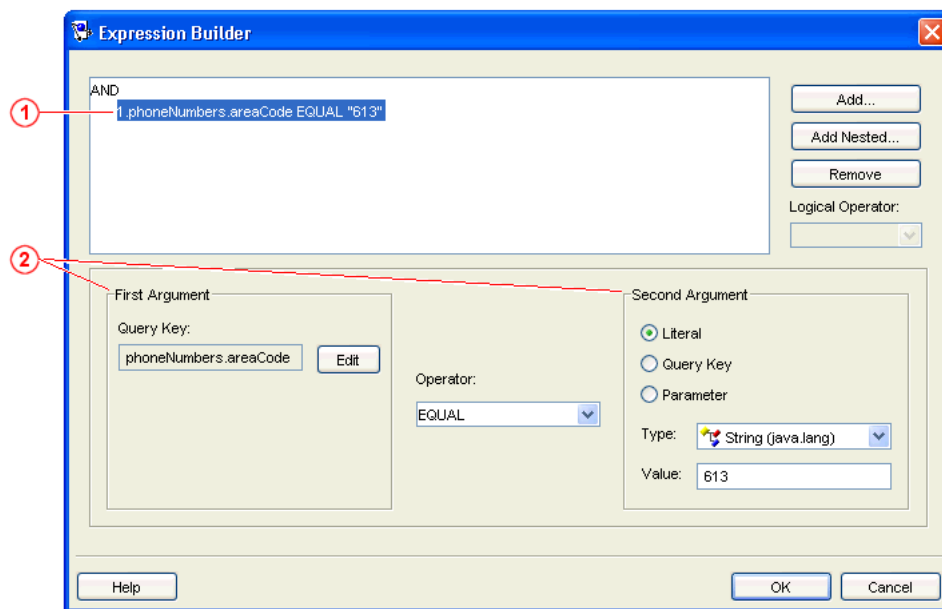


Figure 97–1 numbered callouts identify the following user-interface components:

1. Expression tree
2. Arguments
2. Click **Add** or **Add Nested** to create a new expression. TopLink assigns a sequence number to each node and nested node.
Click **Remove** to remove an existing expression.
3. Select the node and use the **Logical Operator** list to specify the operator for the node (**AND**, **OR**, **Not AND**, or **Not OR**).

Use this table to complete the argument fields for each expression:

Field	Description
First Argument	Click Edit and select the query key for the first argument. The Choose Query Key dialog box appears. Continue with "Adding Arguments" on page 97-15.
Operator	Specify how TopLink should evaluate the expression. Valid operators include: Equal, Not Equal, Equal Ignore Case, Greater Than, Greater Than Equal, Less Than, Less Than Equal, Like, Not Like, Like Ignore Case, Is Null, and Not Null.
Second Argument	Specify the second argument: ■ Literal—Select the Type and enter a literal value for Value. ■ Query Key—Click Edit and select the query key. ■ Parameter—Click Add to add a new parameter and then select from the list. Continue with "Adding Arguments" on page 97-15

Click **OK**. TopLink Workbench adds the expression to the **Named Queries** tab.

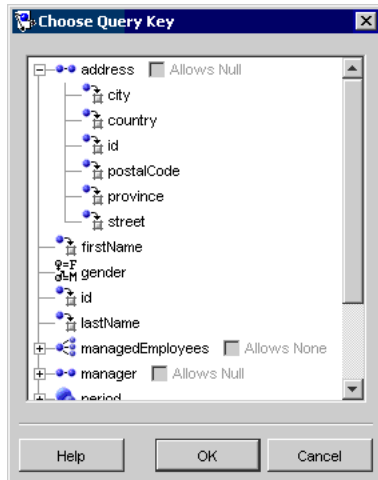
Adding Arguments

Each expression contains elements (*arguments*) to evaluate. Expressions using the Is Null or Not Null operators require only a single argument.

To add new arguments, use this procedure:

1. Select an existing expression or click **Add** (or **Add Nested**) to add a new expression to the named query.
2. For the **First Argument**, click **Edit**. The Choose Query Key dialog box appears.

Figure 97–2 Choose Query Key



3. Select the attribute, specify if the query allows a null value, and click **OK**.
Use the **Allows Null** and **Allows None** options to define an expression with an outer join.
Check the **Allows Null** option to use the ExpressionBuilder method `getAllowingNull`.
Check the **Allows None** option to use the ExpressionBuilder method `anyOfAllowingNone`.
For more information, see ["Using TopLink Expression API For Joins"](#) on page 97-6.
4. Use the **Operator** list to specify how TopLink should evaluate the expression.
5. For the **Second Argument**, select **Literal**, **Query Key**, or **Parameter**.
 - For **Literal** arguments, choose the literal type (such as **String** or **Integer**) and enter the literal value.
 - For **Query Key** arguments, click **Edit**. The Choose Query Key dialog box appears (see step 3 and [Figure 97–2](#) on page 97-15).
 - For **Parameter** arguments, click **Add** to add a parameter and then use the list to select it.

Repeat this procedure for each expression or subexpression.

Example 97–25 Sample Expression

The following expression will find employees who:

- Have a manager with the last name Jones or have no manager, and

- work on projects with the name Beta or project ID 4, and
- live in Canada and have a salary of more than 25,000, or
live in the United States and have a salary of more than 37,500

```

AND
1.manager(Allows Null).lastName EQUAL "Jones"
2.OR
  2.1.projects.name LIKE "BETA"
  2.2.projects.id EQUAL "4"
3.OR
  3.1.AND
    3.1.1.address.country EQUAL "Canada"
    3.1.2.salary GREATER THAN "25000"
  3.2.AND
    3.1.1.address.country EQUAL "United States"
    3.1.2.salary GREATER THAN "37500"

```

Using Java

To create an expression in Java code, use the `Expression` class or `ExpressionBuilder` method `get`.

The `ExpressionBuilder` acts as a substitute for the objects that you query. To construct a query, call methods on the `ExpressionBuilder` that correspond to the attributes of the objects. We recommend that you name `ExpressionBuilder` objects according to the type of objects against which you do a query.

Note: An instance of `ExpressionBuilder` is specific to a particular query. Do not attempt to build another query using an existing builder, because it still contains information related to the first query.

[Example 97–26](#) illustrates how to use the query key `lastName` to reference the field name `L_NAME`.

Example 97–26 Using ExpressionBuilder to Build a Simple Expression

```
Expression expression = new ExpressionBuilder().get("lastName").equal("Young");
```

[Example 97–27](#) illustrates how to create a complex expression by combining two smaller expressions with a logical and operator.

Example 97–27 Combining Two Expressions with a Logical AND Operator

```

ExpressionBuilder emp = new ExpressionBuilder();
Expression exp1, exp2;
exp1 = emp.get("firstName").equal("Ken");
exp2 = emp.get("lastName").equal("Young");
return exp1.and(exp2);

```

[Example 97–28](#) illustrates how to create an expression using the `notLike` operator.

Example 97–28 Using Database Function notLike in an Expression

```
Expression expression = new ExpressionBuilder().get("lastName").notLike("%ung");
```

Creating and Using a User-Defined Function

Different databases sometimes implement the same functions in different ways. For example, an argument that specifies that data returns in ascending order might be ASC or ASCENDING. To manage differences, TopLink recognizes functions and other operators that vary according to the relational database.

Although most platform-specific operators exist in TopLink, if necessary, you can create your own operators.

To create a user-defined function, use the `ExpressionOperator` class.

An `ExpressionOperator` has a selector and a `Vector` of strings:

- The selector is the identifier (*id*) by which users refer to the function.
- The strings are the constant strings used in printing the function. When printed, the strings alternate with the function arguments.

You can also specify whether the operator is prefix or postfix. In a prefix operator, the first constant string prints before the first argument; in a postfix, it prints afterwards.

Where you create a user-defined function and how you add it to the TopLink expression framework depends on whether you want the new function available to all database platforms or to only a specific database platform.

This section describes the following:

- [Making a User-Defined Function Available to a Specific Platform](#)
- [Making a User-Defined Function Available to All Platforms](#)

Making a User-Defined Function Available to a Specific Platform

To make the function that overrides a specific operation on your own platform, use the following procedure:

1. Create a subclass of the desired `DatabasePlatform` (from `oracle.toplink.platform.database` or `oracle.toplink.platform.database.oracle` package) that provides a public method that calls the protected superclass method `addOperator`:

```
...
public class MyOraclePlatform extends DatabasePlatform {
    protected void initializePlatformOperators() {
        super.initializePlatformOperators();
        // Create user-defined function
        ExpressionOperator toUpper = new ExpressionOperator();
        toUpper.setSelector(ExpressionOperator.ToUpperCase);
        Vector v = new Vector();
        v.addElement("UPPERCASE(");
        v.addElement(")");
        toUpper.printAs(v);
        toUpper.bePrefix();
        toUpper.setNodeClass(FunctionExpression.class);

        // Make it available to this platform only
        addOperator(toUpper);
    }
}
```

2. Configure your session to use your platform subclass (see ["Configuring Relational Database Platform at the Project Level"](#) on page 23-2 or ["Configuring a Relational Database Platform at the Session Level"](#) on page 86-1).

Making a User-Defined Function Available to All Platforms

To make the function available to all platforms, use `ExpressionOperator` method `addOperator` as [Example 97-29](#) shows.

Example 97-29 Adding a toUpper Function for All Platforms

```
ExpressionOperator toUpper = new ExpressionOperator();
toUpper.setSelector(600);
Vector v = new Vector();
v.addElement("NUPPER(");
v.addElement(")");
toUpper.printAs(v);
toUpper.bePrefix();
toUpper.setNodeClass(FunctionExpression.class);

ExpressionOperator.addOperator(toUpper);
```

Note: Represent the number in the `setSelector` method by a constant value. Ensure that this number is greater than 500 (numbers below 500 are reserved in `TopLink`).

Using a User-Defined Function

Regardless of whether you added the function for all platforms or for a specific platform, [Example 97-30](#) illustrates how to use the `Expression` method `getFunction` to access the user-defined expression operator represented by a constant with the value 600.

Example 97-30 Accessing a User-Defined Function

```
ReadObjectQuery query = new ReadObjectQuery(Employee.class);
Expression functionExpression = new ExpressionBuilder().get("firstName").
    getFunction(600).equal("BOB");
query.setSelectionCriteria(functionExpression);
session.executeQuery(query);
```

Using Advanced Query API

This section explains more advanced TopLink query API calls and techniques most commonly used later in the development cycle, including the following:

- [Using Redirect Queries](#)
- [Using Historical Queries](#)
- [Using Queries With Fetch Groups](#)
- [Querying on Interfaces](#)
- [Querying on an Inheritance Hierarchy](#)
- [Appending Additional Join Expressions](#)
- [Using Queries on Variable One-to-One Mappings](#)
- [Using Oracle Database Features](#)
- [Using EJB Finders](#)
- [Handling Cursor and Stream Query Results](#)
- [Using Queries and the Cache](#)

For more information about the available query API, see *Oracle TopLink API Reference*.

Using Redirect Queries

A redirect query is a named query that delegates query execution control to your application. redirect queried allow you to define the query implementation in code as a static method.

To perform complex operations, you can combine query redirectors with the TopLink query framework.

Creating a Redirect Query

To perform complex operations, you can combine query redirectors with the TopLink query framework. To create a redirector, implement the `oracle.toplink.queryframework.QueryRedirector` interface. The query mechanism executes the `Object invokeQuery(DatabaseQuery query, Record arguments, Session session)` method and waits for the results.

TopLink provides one preimplemented redirector, the `MethodBasedQueryRedirector` method. To use this redirector, create a static `invoke` method on a class, and use the `setMethodName(String)` call to specify the method to invoke.

Example 99–1 Redirect Query

```
ReadObjectQuery query = new ReadObjectQuery(Employee.class);
query.setName("findEmployeeByAnEmployee");
query.addArgument("employee");

MethodBaseQueryRedirector redirector = new
    MethodBaseQueryRedirector(QueryRedirectorTest.class, "findEmployeeByAnEmployee");
query.setRedirector(redirector);
Descriptor descriptor = getSession().getDescriptor(query.getReferenceClass());
descriptor.getQueryManager().addQuery(query.getName(), query);

Vector arguments = new Vector();
arguments.addElement(employee);
objectFromDatabase =
    getSession().executeQuery("findEmployeeByAnEmployee", Employee.class, arguments);

public class QueryRedirectorTest {
    public static Object findEmployeeByAnEmployee(
        DatabaseQuery query,
        oracle.toplink.publicinterface.DatabaseRow arguments,
        oracle.toplink.sessions.Session
        session) {
        ((ReadObjectQuery) query).setSelectionObject(arguments.get("employee"));
        return session.executeQuery(query);
    }
}
```

Using Historical Queries

To make a query time-aware, you specify an `ASOfClause` clause that TopLink appends to the query. Use the `ASOfClause` class if your historical schema is based on time stamps or the `ASOfSCNC` clause class if your historical schema is based on database system change numbers. You can specify an `ASOfClause` at the time you acquire a historical session so that TopLink appends the same clause to all queries, or you can specify an `ASOfClause` on a query-by-query basis.

[Example 99–2](#) shows how to create a query that uses a particular `ASOfClause`. This query will read all `Employee` objects as of the time specified by `timestamp` using the appropriate history tables described by the `HistoryPolicy` set on the `Employee` descriptor.

Example 99–2 Using a Historical Session

```
ReadAllQuery historicalQuery = new ReadAllQuery(Employee.class);
ASOfClause asOfClause = new ASOfClause(timestamp);
historicalQuery.setASOfClause(asOfClause);
historicalQuery.dontMaintainCache();
Vector pastEmployees = (Vector)historicalSession.executeQuery(historicalQuery);
```

Using Queries With Fetch Groups

You can use a fetch group with a `ReadObjectQuery` or `ReadAllQuery`. When you execute the query, TopLink retrieves only the attributes in the fetch group. TopLink automatically executes a query to fetch all the attributes excluded from this subset when and if you call a getter method on any one of the excluded attributes.

Note: Fetch groups are only supported with CMP.

This section describes the following:

- [Configuring Default Fetch Group Behavior](#)
- [Querying With a Static Fetch Group](#)
- [Querying With a Dynamic Fetch Group](#)

For more information about fetch groups, see "[Fetch Groups and Object Level Read Queries](#)" on page 96-12.

Configuring Default Fetch Group Behavior

You can optionally designate at most one fetch group as the default fetch group for a descriptor's reference class.

If you execute a `ReadObjectQuery` or `ReadAllQuery` without specifying a fetch group, TopLink will use the default fetch group unless you configure the query otherwise, as [Example 99-3](#) shows.

Example 99-3 Configuring Default Fetch Group Behavior

```
// at the descriptor level
FetchGroup group = new FetchGroup("nameOnly");
group.addAttribute("firstName");
group.addAttribute("lastName");
employeeDescriptor.getFetchGroupManager().addFetchGroup(group);
// set the default fetch group
employeeDescriptor.getFetchGroupManager().setDefaultFetchGroup(group);

// when query1 is executed, the default fetch group applies
ReadAllQuery query1 = new ReadAllQuery(Employee.class);

// when query2 is executed, the default fetch group does not apply
ReadAllQuery query2 = new ReadAllQuery(Employee.class);
query2.setShouldUsedefaultFetchGroup(false);
```

Querying With a Static Fetch Group

[Example 99-4](#) shows how to configure a `ReadObjectQuery` for the `Employee` class with a `FetchGroup` named `nameOnly` previously stored in the `FetchGroupManager` owned by the `Employee` class's descriptor.

Example 99-4 Configuring a Query with a FetchGroup Using the FetchGroupManager

In this example, only the `Employee` attributes `firstName` and `lastName` are fetched. If you call the `Employee` method `get` for any other attribute, TopLink executes another query to retrieve all unfetched attribute values. Thereafter, calling that `get` method will return the value directly from the object.

```
// create static fetch group at the descriptor level
FetchGroup group = new FetchGroup("nameOnly");
group.addAttribute("firstName");
group.addAttribute("lastName");
descriptor.getFetchGroupManager().addFetchGroup(group);

// use static fetch group at query level
ReadAllQuery query = new ReadAllQuery(Employee.class);
query.setFetchGroupName("nameOnly");
```

Querying With a Dynamic Fetch Group

Example 99–5 shows how to create a `FetchGroup` instance dynamically, at the time you create and execute a query, and configure the query with that `FetchGroup` directly.

Example 99–5 *Configuring a Query with a FetchGroup Dynamically*

In this example, only the `firstName`, `lastName`, and `salary` attributes are fetched. If you call the `Employee` method `get` for any other attribute, TopLink executes another query to retrieve all unfetched attribute values. Thereafter, calling that `get` method will return the value directly from the object.

```
// dynamic fetch group query
ReadAllQuery query = new ReadAllQuery(Employee.class);
FetchGroup group = new FetchGroup("nameAndSalary");
group.addAttribute("firstName");
group.addAttribute("lastName");
group.addAttribute("salary");
query.setFetchGroup(group);
```

Querying on Interfaces

When you define descriptors for an interface to enable querying, TopLink supports querying on an interface, as follows:

- If there is only a single implementor of the interface, the query returns an instance of the concrete class.
- If there are multiple implementors of the interfaces, the query returns instances of all implementing classes.

Querying on an Inheritance Hierarchy

When you query on a class that is part of an inheritance hierarchy, the session checks the descriptor to determine the type of the class, as follows:

- If you configure the descriptor to read subclasses (the default configuration), the query returns instances of the class and its subclasses.
- If you configure the descriptor not to read subclasses, the query returns only instances of the queried class, but no instances of the subclasses.
- If neither of these conditions applies, the class is a leaf class and does not have any subclasses. The query returns instances of the queried class.

Appending Additional Join Expressions

You can set the query manager to automatically append an expression to every query it performs on a class. For example, you can add an expression that filters the database for the valid instances of a given class.

Use this to do the following:

- Filter logically deleted objects
- Enable two independent classes to share a single table without inheritance
- Filter historical versions of objects

Using Java

Using Java, configure a descriptor with additional join expressions by creating an amendment method (see ["Configuring Amendment Methods"](#) on page 28-90), and then using the `DescriptorQueryManager` methods `setAdditionalJoinExpression` or `setMultipleTableJoinExpression`, as [Example 99-6](#) shows.

Example 99-6 *Registering a Query That Includes a Join Expression*

In [Example 99-6](#), the join expression filters invalid instances of employee from the query.

```
public static void addToDescriptor(Descriptor descriptor) {
    ExpressionBuilder builder = new ExpressionBuilder();
    descriptor.getQueryManager().setAdditionalJoinExpression(
        (builder.getField("EMP.STATUS").notEqual("DELETED")).and(
            builder.getField("EMP.STATUS").notEqual("HISTORICAL"))
    );
}
```

Using Queries on Variable One-to-One Mappings

`TopLink` does not provide a method to directly query against variable one-to-one mappings. To query against this type of mapping, combine `TopLink DirectQueryKeys` and `TopLink ReportQueries` to create query selection criteria for classes that implement the interface, as follows:

1. Create two `DirectQueryKeys` to query for the possible implementors of the interface:
 - The first `DirectQueryKey` is for the class indicator field for the variable one-to-one mapping.
 - The second `DirectQueryKey` is for the foreign key to the class or table that implements the interface.
2. Create a `subSelect` statement for each concrete class that implements the interface included in the query selection criteria.
3. Implement a `ReportQuery`.

Example 99-7 *Creating DirectQueryKeys*

```
/* The DirectQueryKeys as generated in the TopLink project Java source code from
TopLink Workbench */

...
descriptor.addDirectQueryKey("locationTypeCode", "DEALLOCATION.DEALLOCATIONOBJECTTYPE");
descriptor.addDirectQueryKey("locationTypeId", "DEALLOCATION.DEALLOCATIONOBJECTID");
;
```

Using Oracle Database Features

If you are using Oracle Database, you can take advantage of `TopLink` support for the following Oracle Database features:

- [Oracle Hints](#)

- [Hierarchical Queries](#)

Oracle Hints

Oracle Hints is an Oracle Database feature through which you can make decisions usually reserved for the optimizer. You use hints to specify things such as join order for a join statement, or the optimization approach of an SQL call.

The TopLink query framework supports Oracle Hints with the following API:

```
setHintString("/*[hints or comments]*/");
```

TopLink adds the hint to the SQL string as a comment immediately following a SELECT, UPDATE, INSERT, or DELETE statement.

Add hints to a read query as follows:

1. Create a `ReadObjectQuery` or a `ReadAllQuery`
2. Set the selection criteria.
3. Add hints as needed.

For example, the following code uses the FULL hint (which explicitly chooses a full table scan for the specified table):

```
// This line sets up the query
ReadObjectQuery query = new ReadObjectQuery(Employee.class);
query.setSelectionCriteria(new ExpressionBuilder().get("id").equal(new Integer(1)));
// This line adds the hint
query.setHintString("/*+ FULL */" );
```

This code generates the following SQL:

```
SELECT /*+ FULL */ FROM EMPLOYEE WHERE ID=1
```

To add hints to WRITE, INSERT, UPDATE, and DELETE, create custom queries for these operations in the TopLink query framework, then specify hints as required. For more information, see the following:

- ["Configuring Custom SQL Queries for Basic Persistence Operations"](#) on page 29-6
- ["Configuring Custom EIS Interactions for Basic Persistence Operations"](#) on page 31-6

For more information about the available hints, see the Oracle Database documentation.

Hierarchical Queries

Hierarchical Queries is an Oracle Database mechanism that lets you select database rows based on hierarchical order. For example, you can design a query that reads the row of a given employee, followed by the rows of people this employee manages, followed by their managed employees, and so on.

To create a hierarchical query, use the `setHierarchicalQueryClause` method. This method takes three parameters, as follows:

```
setHierarchicalQueryClause(startWith, connectBy, ordersSibling)
```

This expression requires all three parameters, as described in the subsequent text.

startWith Parameter

The `startWith` parameter in the expression specifies the first object in the hierarchy. This parameter mirrors the Oracle Database `START WITH` clause.

To include a `startWith` parameter, build an expression to specify the appropriate object, and pass it as a parameter in the `setHierarchicalQueryClause` method. If you do not specify the root object for the hierarchy, set this value to `null`.

connectBy Parameter

The `connectBy` parameter specifies the relationship that creates the hierarchy. This parameter mirrors the Oracle Database `CONNECT BY` clause.

Build an expression to specify the `connectBy` parameter, and pass it as a parameter in the `setHierarchicalQueryClause` method. Because this parameter defines the nature of the hierarchy, it is required for the `setHierarchicalQueryClause` implementation.

orderSibling Parameter

The `orderSibling` parameter in the expression specifies the order in which the query returns sibling objects in the hierarchy. This parameter mirrors the Oracle Database `ORDER SIBLINGS` clause.

To include an `orderSibling` parameter, define a vector, and to include the order criteria, use the `addElement` method. Pass the vector as the third parameter in the `setHierarchicalQueryClause` method. If you do not specify an order, set this value to `null`.

Example 99–8 Hierarchical Query

```
ReadAllQuery raq = new ReadAllQuery(Employee.class);
// Specifies a START WITH expression
Expression startExpr = expressionBuilder.get("id").equal(new Integer(1));
// Specifies a CONNECT BY expression
Expression connectBy = expressionBuilder.get("managedEmployees");
// Specifies an ORDER SIBLINGS BY vector
Vector order = new Vector();
order.addElement(expressionBuilder.get("lastName"));
order.addElement(expressionBuilder.get("firstName"));
raq.setHierarchicalQueryClause(startExpr, connectBy, order);
Vector employees = uow.executeQuery(raq);
```

This code generates the following SQL:

```
SELECT * FROM EMPLOYEE START WITH ID=1 CONNECT BY PRIOR ID=MANAGER_ID ORDER
SIBLINGS BY LAST_NAME, FIRST_NAME
```

Using EJB Finders

This section describes how to use EJB finders in TopLink, including the following:

- [Creating a Finder](#)
- [Using DatabaseQuery Finders](#)
- [Using Named Query Finders](#)
- [Using Primary Key Finders](#)
- [Using EJB QL Finders](#)

- [Using SQL Finders](#)
- [Using Redirect Finders](#)
- [Using the ejbSelect Method](#)

Creating a Finder

In general, to create a finder for an entity bean that uses the TopLink query framework, you must define, declare, and configure it.

For predefined finders (see ["Predefined Finders"](#) on page 96-24), you do not need to explicitly create a finder.

For default finders (see ["Default Finders"](#) on page 96-26), you only need to define the finder method.

To create a finder for an entity bean that uses the TopLink query framework, follow these steps:

1. Define the finder method.

For EJB 1.1 beans, define the method on the entity bean's `remote` interface.

For EJB 2.0 beans, define the method on the entity bean's `remoteHome` or `localHome` interface.

For CMP beans, define the method on the entity bean's `Home` interface.

For default finders (see ["Default Finders"](#) on page 96-26), you must define the finder as follows:

- `<RETURN-TYPE> findBy<CMP-FIELD-NAME>(<CMP-FIELD-TYPE>)`
- the first letter of `<CMP-FIELD-NAME>` must be capitalized
- `<RETURN-TYPE>` may be a single bean type, an `Enumeration` (EJB 1.1), or `Collection` (EJB 2.0).

For example:

```
EmployeeBean (Integer id, String name)
EmployeeHome ..{
    Employee findById(Integer id) throws...;
    Collection findByName(String name) throws...;
}
```

Note: If you are using default finders (see ["Default Finders"](#) on page 96-26), you are finished. TopLink will implement the finder for you at run time.

2. Declare the finder in the `ejb-jar.xml` file (see ["ejb-jar.xml Finder Options"](#) on page 99-9).
3. Start TopLink Workbench.
4. Click the project icon in the **Navigators** and select: **Selected > Update Project from ejb-jar.xml** to read in the finders.
5. Go to the **Queries > Named Queries** tab for the bean (see ["Using Named Queries"](#) on page 98-17).
6. Select and configure the finder.

Notes: For predefined finders `findOneByQuery` and `findManyByQuery`, the client creates a query at run time and passes it as a parameter to the finder. Because of this, do not configure query options on these finders. Instead, configure options on the query passed into the finder. For more information about predefined finders, see ["Predefined Finders"](#) on page 96-24.

7. If required, create an implementation for the query. Some query options require a query definition in code on a helper class, but most common queries do not.

When you use TopLink CMP, define finder methods on the bean's `Home` interface, not in the entity bean itself. TopLink CMP provides this functionality and offers several strategies to create and customize finders. The EJB container and TopLink automatically generate the implementation.

ejb-jar.xml Finder Options

The `ejb-jar.xml` file contains a project's EJB entity bean information, including definitions for any finders used for the beans. To create and maintain the `ejb-jar.xml` file, use either a text editor or TopLink Workbench.

The `entity` tag encapsulates a definition for an EJB entity bean. Each bean has its own `entity` tag that contains several other tags that define bean functionality, including bean finders.

[Example 99-9](#) illustrates the structure of a typical finder defined within the `ejb-jar.xml` file.

Note: Use a combination of an escape character and a double-quote (`\`) when defining your query using EJB QL. For more information on correct query syntax, see a note at the end of [Configuring Named Query Selection Criteria](#) on page 28-17.

Example 99-9 A Simple Finder Within the ejb-jar.xml File

```
<entity>...
  <query>
    <query-method>
      <method-name>findLargeAccounts</method-name>
      <method-params>
        <method-param>double</method-param>
      </method-params>
    </query-method>
    <ejb-ql><![CDATA[SELECT OBJECT(account) FROM AccountBean account WHERE
      account.balance > ?1]]></ejb-ql>
  </query>
  ...
</entity>
```

The `entity` tag contains zero or more `query` elements. Each `query` tag corresponds to a finder method defined on the bean's home or local `Home` interface.

Note: You can share a single query between both `Home` interfaces, as follows:

- Define the same finder (same name, return type, and parameters) on both `Home` interfaces.
 - Include a single `query` element in the `ejb-jar.xml` file.
-

Here are the elements defined in the `query` section of the `ejb-jar.xml` file:

- `description` (optional): Provides a description of the finder.
- `query-method`: Specifies the method for a finder or `ejbSelect` query.
- `method-name`: Specifies the name of a finder or select method in the entity bean implementation class.
- `method-params`: Contains a list of the fully qualified Java type names of the method parameters.
- `method-param`: Contains the fully qualified Java type name of a method parameter.
- `result-type-mapping` (optional): Specifies how to map an abstract schema type returned by a query for an `ejbSelect` method. You can map the type to an `EJBLocalObject` or `EJBObject` type. Valid values are `Local` or `Remote`.
- `ejb-ql`: Used for all EJB QL finders. It contains the EJB QL query string that defines the finder or `ejbSelect` query. Leave this element empty for non-EJB QL finders.

Using DatabaseQuery Finders

TopLink provides a predefined finder that takes a `DatabaseQuery` such as a `ReadAllQuery`. To use this feature in a bean, add the following finder definition to the `Home` interface of your bean:

```
public Collection findManyByQuery(ReadAllQuery query) throws RemoteException,
FinderException;
```

```
public <EJBLocal/Remote> findOneByQuery(ReadObjectQuery query) throws
RemoteException, FinderException;
```

To execute a `ReadAllQuery` finder, create the query on the client, as [Example 99–10](#) shows.

Example 99–10 A *ReadAllQuery* Finder

```
{
    ReadAllQuery query = new ReadAllQuery(Employee.class);
    query.addJoinedAttribute("address");
    Enumeration employees = getEmployeeHome().findManyByQuery(query);
}
```

Using Named Query Finders

You can implement an EJB finder method (including TopLink predefined finders) as a named query. For more information, see ["Using Named Queries"](#) on page 98-17. You execute such a finder as you would any other.

Using Primary Key Finders

TopLink provides a predefined finder (`findByPrimaryKey`) that takes a primary key as an `Object`.

Example 99–11 Executing a Primary Key Finder

```
{
    Employee employee = getEmployeeHome().findByPrimaryKey(primaryKey);
}
```

Using EJB QL Finders

EJB QL is the standard query language defined in the EJB 2.0 specification. TopLink supports EJB QL for both EJB 1.1 and EJB 2.0 beans. EJB QL finders let you specify an EJB QL string as the implementation of the query.

Note: Use a combination of an escape character and a double-quote (`\`) when defining your query using EJB QL. For more information on correct query syntax, see a note at the end of [Configuring Named Query Selection Criteria](#) on page 28-17.

EJB QL offers several advantages:

- It is the EJB 2.0 standard for queries.
- You can use it to construct most queries.
- You can implement dependent object queries with EJB QL.

The disadvantage of EJB QL is that it is difficult to use when you construct complex queries.

To create an EJB QL finder under EJB 1.1 specification, use this procedure:

1. Declare the finder on the `Remote` interface.
2. Start TopLink Workbench.
3. Go to the **Queries > Finders > Named Queries** tab for the bean.
4. Add a finder and give it a name that matches the method name you declared in Step 1.
5. Set the required parameters.
6. Set **Query Format** to EJB QL, and enter the EJB QL query in the **Query String** field.

To create an EJB QL finder under EJB 2.0 specification, use this procedure:

1. Declare the finder on either the `LocalHome` or the `RemoteHome` interface.
2. Start TopLink Workbench.
3. Reimport the `ejb-jar.xml` file to synchronize the project to the file.
TopLink Workbench synchronizes changes between the project and the `ejb-jar.xml` file.

The following is an example of a simple EJB QL query that requires one parameter. In this example, the question mark ("`?`") in `?1` specifies a parameter:

```
SELECT OBJECT(employee) FROM Employee employee WHERE (employee.name =?1)
```

To create an EJB QL finder for a CMP bean, use this procedure:

1. Declare the finder in the `ejb-jar.xml` file, and enter the EJB QL string in the `ejb-ql` tag.
2. Declare the finder on the `Home` interface, the `LocalHome` interface, or both, as required.
3. Start TopLink Workbench.
4. Specify the `ejb-jar.xml` file location and choose **File > Updated Project** from the `ejb-jar.xml` file to read in the finders.
5. Go to the **Queries > Finders > Named Queries** tab for the bean.
6. Add a finder, and give it the same name as the finder you declared on your bean's home. Then add any required parameters.
7. Select and configure the finder.

The following is an example of a simple EJB QL query that requires one parameter. In this example, the question mark ("?) in ?1 specifies a parameter.

```
SELECT OBJECT(employee) FROM Employee employee WHERE (employee.name =?1)
```

Using SQL Finders

You can use custom SQL code to specify finder logic. SQL lets you implement logic that might not be possible to express with TopLink expressions or EJB QL.

To create a SQL finder, use this procedure:

1. Declare the finder in the `ejb-jar.xml` file, and leave the `ejb-ql` tag empty.
2. Start TopLink Workbench.
3. Specify the `ejb-jar.xml` file location and choose **File > Updated Project** from the `ejb-jar.xml` file to read in the finders.
4. Go the **Queries > Named Queries** tab for the bean.
5. Select the finder, select the **SQL** radio button, and enter the SQL string.
6. Configure the finder.

The following is an example of a simple SQL finder that requires one parameter. In this example, the number sign character (#) is used to bind the argument `projectName` within the SQL string:

```
SELECT * FROM EJB_PROJECT WHERE (PROJ_NAME = #projectName)
```

Using Redirect Finders

Redirect finders let you specify a finder in which the implementation is defined as a static method on an arbitrary helper class. When you invoke the finder, it redirects the call to the specified static method.

For more information about redirect queries, see ["Redirect Queries"](#) on page 96-19.

The finder can have any arbitrary parameters. If the finder includes parameters, TopLink packages them into a `Vector` and passes them to the redirect method.

Redirect finders offer several advantages. Because you define the redirect finder implementation independently from the bean that invokes it, you can build the redirect finder to accept any type and number of parameters. This lets you create a generic redirect finder that accepts several different parameters and return types, depending on input parameters.

A common strategy for using redirect finders is to create a generic finder that does the following:

- Includes logic to perform several tasks
- Reads the first passed parameter to identify the type of finder requested and select the appropriate logic

The redirect method contains the logic required to extract the relevant data from the parameters and uses it to construct a TopLink query.

The main disadvantage of redirect finders is that they are complex and can be difficult to configure. They also require an extra helper method to define the query. However, because they support complex logic, they are often the best choice when you need to implement logic unrelated to the bean on which the redirect method is called.

To create a redirect finder, use the following procedure:

1. Declare the finder in the `ejb-jar.xml` file, and leave the `ejb-ql` tag empty.
2. Declare the finder on the `Home` interface, the `localHome` interface, or both, as required.
3. Create an amendment method.
For more information, see ["Configuring Amendment Methods"](#) on page 28-90.
4. Start TopLink Workbench.
5. Choose **Advanced Properties > After Load** from the menu for the bean.
6. Specify the class and name of the static method to enable the amendment method for the descriptor.

The amendment method then adds a query to the descriptor's query manager, as follows:

```
ReadAllQuery query = new ReadAllQuery();
query.setRedirector(new MethodBaseQueryRedirector (examples.ejb.cmp20.advanced.
    FinderDefinitionHelper.class,"findAllEmployeesByStreetName"));
descriptor.getQueryManager().addQuery ("findAllEmployeesByStreetName", query);
```

The redirect method must return either a single entity bean (object) or a `Vector`. Here are the possible method signatures:

```
public static Object redirectedQuery(oracle.toplink.sessions.Sessions, Vector args)
```

and

```
public static Vector redirectedQuery(oracle.toplink.sessions.Sessions, Vector args)
```

When you implement the query method, ensure that the method returns the correct type. For methods that return more than one bean, set the return type to `java.util.Vector`. TopLink converts this result to `java.util.Enumeration` (or `Collection`) if required.

Note: The redirect method also interprets a TopLink session as a parameter. For more information about a TopLink session, see [Part XVI, "TopLink Sessions"](#).

At run time, the client invokes the finder from the entity bean home and packages the arguments into the `args` vector in order of appearance from the finder method signature. The client passes the vector to the redirect finder, which uses them to execute a TopLink expression.

Example 99–12 A Simple Redirect Query Implementation

```
public class RedirectorTest {
    private Session session;
    private Project project;
    public static void main(String args[]) {

        RedirectorTest test = new RedirectorTest();

        test.login();

        try {
            // Create the arguments to be used in the query
            Vector arguments = new Vector(1);
            arguments.add("Smith");

            // Run the query
            Object o = test.getSession()
                .executeQuery(test.redirectorExample(), arguments);
            o.toString();
        }
        catch (Exception e) {
            System.out.println("Exception caught -> " + e);
            e.printStackTrace();
        }
    }

    public ReadAllQuery redirectorExample() {

        // Create a redirector
        MethodBasedQueryRedirector redirector = new MethodBasedQueryRedirector();

        // Set the class containing the public static method
        redirector.setMethodClass(RedirectorTest.class);

        // Set the name of the method to be run
        redirector.setMethodName("findEmployeeByLastName");

        // Create a query and add the redirector previously created
        ReadAllQuery readAllQuery = new ReadAllQuery(Employee.class);
        readAllQuery.setRedirector(redirector);
        readAllQuery.addArgument("lastName");

        return readAllQuery;
    }

    // Call the static method
    public static Object findEmployeeByLastName(oracle.toplink.sessions
        .Session
        session, Vector arguments) {
```

```

        // Create a query
        ReadAllQuery raq = new ReadAllQuery();
        raq.setReferenceClass(Employee.class);
        raq.addArgument("lastName");

        // Create the selection criteria
        ExpressionBuilder employee = new ExpressionBuilder();
        Expression whereClause =
            employee.get("lastName").equal(arguments.firstElement());

        // Set the selection criteria
        raq.setSelectionCriteria(whereClause);

        return (Vector)session.executeQuery(raq, arguments);
    }
    [...]
}

```

Using the ejbSelect Method

The `ejbSelect` method is a query method intended for internal use within an entity bean instance. Specified on the abstract bean itself, the `ejbSelect` method is not directly exposed to the client in the home or component interface. Defined as abstract, each bean can include zero or more such methods.

Select methods have the following characteristics:

- The method name must have `ejbSelect` as its prefix.
- It must be declared as public.
- It must be declared as abstract.
- The `throws` clause must specify the `javax.ejb.FinderException`, although it may also specify application-specific exceptions as well.
- Under EJB 2.0 specification, the `result-type-mapping` tag in the `ejb-jar.xml` file determines the return type for `ejbSelect` methods. Set the flag to `Remote` to return `EJBObjects`; set it to `Local` to return `EJBLocalObjects`.

The format for an `ejbSelect` method definition should be similar to the following:

```
public abstract type ejbSelect<METHOD>(...);
```

The `ejbSelect` query return type is not restricted to the entity bean type on which the `ejbSelect` is invoked. Instead, it can return any type corresponding to a container-managed relationship or container-managed field.

Although the `ejbSelect` method is not based on the identity of the entity bean instance on which it is invoked, it can use the primary key of an entity bean as an argument. This creates a query that is logically scoped to a particular entity bean instance.

To create an `ejbSelect` method, use this procedure:

1. Update the `ejb-jar.xml` file as follows:
 - Declare the `ejbSelect` method.
 - Enter the EJB QL string in the `ejb-ql` tag.
 - Specify the return type in the `result-type-mapping` tag (if required).
2. Declare the `ejbSelect` on the abstract bean class.

3. Start TopLink Workbench.
4. Click the project icon in the **Navigator**, and select: **Selected > Update Project from ejb-jar.xml** to read in the finders.
5. Go the **Queries > Named Queries** tab for the bean.
6. Select and configure the `ejbSelect` method.

Handling Cursor and Stream Query Results

Cursors and streams are related mechanisms that let you work with large result sets efficiently. See ["Stream and Cursor Query Results"](#) on page 96-8 for more information.

[Table 99–1](#) lists the methods that TopLink provides for all subclasses of `DataReadQuery` and `ReadAllQuery` that you can use to make your query return its results as a cursor or stream.

Table 99–1 *Stream and Cursor Query Result Options*

Method	Query Returns	Description
<code>useScrollableCursor</code>	<code>ScrollableCursor</code>	Allows you access a database result set cursor, allowing you to move forward and backward through the result set.
<code>useCursoredStream</code>	<code>CursoredStream</code>	Allows you to access results one at a time in sequence, as results become available to the underlying database result set cursor.

Using a `ScrollableCursor` or `CursoredStream` combines the features of a TopLink with the ability of the database to cursor data, and breaks up the result set into smaller, more manageable pieces.

The behavior of a query that uses a `ScrollableCursor` or `CursoredStream` differs from other queries in that the elements requested by the client are sent to the client.

This section describes the following:

- [Cursors and Java Iterators](#)
- [Java Streams](#)
- [Optimizing Streams](#)
- [Using Cursors and Streams With EJB Finders](#)

Cursors and Java Iterators

The TopLink scrollable cursor lets you scroll through a result set from the database without reading the whole result set in a single database read operation. The `ScrollableCursor` class implements the Java `ListIterator` interface to allow for direct and relative access within the stream. Scrollable cursors also let you scroll forward and backward through the stream.

Traversing Data With Scrollable Cursors

Several methods let you navigate data with a scrollable cursor:

- `relative(int i)`: advances the row number in relation to the current row by one row

- `absolute(int i)`: places the cursor at an absolute row position, 1 being the first row

Several strategies are available for traversing data with cursors. For example, to start at the end of the data set and work toward the first record:

1. Call the `afterLast` method to place the cursor after the last row in the result set.
2. Use the `hasPrevious` method to determine whether there is a record above the current record. This method returns `false` when you reach the final record in the data set.
3. If the `hasPrevious` method returns `true`, call the `previous` method to move the cursor to the row prior to the current row and read that object.

These are common methods for data traversal, but they are not the only available methods. For more information about the available methods, see *Oracle TopLink API Reference*.

To use the `ScrollableCursor` object, the JDBC driver must be compatible with the JDBC 2.0 specifications.

Example 99–13 Traversing with a Scrollable Cursor

```
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
query.useScrollableCursor();
ScrollableCursor cursor = (ScrollableCursor) session.executeQuery(query);

while (cursor.hasNext()) {
    System.out.println(cursor.next().toString());
}
cursor.close();
```

Java Streams

Java streams let you retrieve query results as individual records or groups of records, which can result in a performance increase. You can use streams to build efficient TopLink queries, especially when the queries are likely to generate large result sets.

Cursored Stream Support

Cursored streams combine the iterative ability of the `ScrollableCursor` interface with TopLink support for streams. The result is the ability to read back a query result set from the database in manageable subsets, and to scroll through the result set stream.

The `useCursoredStream` method of the `ReadAllQuery` class provides cursored stream support.

Example 99–14 Cursored Streams

```
CursoredStream stream;
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
query.useCursoredStream();
stream = (CursoredStream) session.executeQuery(query);
```

The query returns an instance of `CursoredStream` rather than a `Vector`, which can be a more efficient approach. For example, consider the following two code examples.

[Example 99–15](#) returns a `Vector` that contains all employee objects. If ACME has 10,000 employees, the `Vector` contains references to 10,000 `Employee` objects.

Example 99–15 Using a Vector

```
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
Enumeration employeeEnumeration;

Vector employees = (Vector) session.executeQuery(query);
employeeEnumeration = employee.elements();

while (employeeEnumeration.hasMoreElements())
{
    Employee employee = (Employee) employeeEnumeration.nextElement();
    employee.doSomeWork();
}
```

The following example returns a `CursoredStream` instance rather than a `Vector`. The `CursoredStream` collection appears to contain all 10,000 objects, but initially contains a reference to only the first 10 `Employee` objects. It retrieves the remaining objects in the collection as they are needed. In many cases, the application never needs to read all the objects:

```
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
query.useCursoredStream();

CursoredStream stream = (CursoredStream) session.executeQuery(query);
while (! stream.atEnd()) {
    Employee employee = (Employee) stream.read();
    employee.doSomeWork();
    stream.releasePrevious();
}
stream.close();
```

Note: The `releasePrevious` message is optional. This releases any previously read objects and frees system memory. Even though released objects are removed from the cursored stream storage, they remain in the identity map.

Optimizing Streams

To optimize `CursoredStream` performance, provide a *threshold* and *page size* to the `useCursoredStream(Threshold, PageSize)` method, as follows:

- The threshold specifies the number of objects to read into the stream initially. The default threshold is 10.
- The page size specifies the number of objects to read into the stream after the initial group of objects. This occurs after the threshold number of objects is read. Although larger page sizes result in faster overall performance, they introduce delays into the application when TopLink loads each page. The default page size is 5.

When you execute a batch-type operation, use the `dontMaintainCache` method with a cursored stream. A batch operation performs simple operations on large numbers of objects and then discards the objects. Cursored streams create the required

objects only as needed, and the `dontMaintainCache` ensures that these transient objects are not cached.

Using Cursors and Streams With EJB Finders

Large result sets can be resource-intensive to collect and process. To give the client more control over the returned results, configure TopLink finders to use cursors. This combines TopLink's `CursoredStream` with the ability of the database to cursor data, and breaks up the result set into smaller, more manageable pieces.

Note: If you use the transactional attribute `REQUIRED` for an entity bean, wrap all read operations in `UserTransaction` methods `begin` and `commit` to ensure that read operations beyond the first page of the cursor have a transaction in which to work.

Building the Query

You can configure any finder that returns a `java.util.Enumeration` (under EJB 1.1 specification) or a `java.util.Collection` (under EJB 2.0 specification) to use a cursor. When you create the query for the finder, add the `useCursoredStream` option to enable cursoring.

Example 99–16 Cursored Stream in a Finder

```
ReadAllQuery raq = new ReadAllQuery();
ExpressionBuilder bldr = new ExpressionBuilder();
raq.setReferenceClass(ProjectBean.class);
raq.useCursoredStream();
raq.addArgument("projectName");
raq.setSelectionCriteria(bldr.get("name").
    like(bldr.getParameter("projectName")));
descriptor.getQueryManager().addQuery ("findByNameCursored", query);
```

Executing the Finder From the Client in EJB 1.1

TopLink offers additional elements for traversing finder results. These elements include the following:

- `hasMoreElements` method: Returns a `boolean` value indicating whether or not there are any more elements in the result set.
- `nextElement` method: Returns the next available element.
- `nextElements(int count)` method: Retrieves a `Vector` of at most **count** elements from the available results, depending on how many elements remain in the result set.
- `close` method: Closes the cursor on the server. The client must send this message, or the database connection does not close.

[Example 99–17](#) illustrates client-code executing a cursored finder.

Example 99–17 Cursored Finder Under EJB 1.1 Specification

```
import oracle.toplink.ejb.cmpwls11. CursoredEnumerator;
//... other imports as necessary
getTransaction().begin();
CursoredEnumerator cursoredEnumerator = (CursoredEnumerator)getProjectHome()
    .findByNameCursored("proj%");
```

```
Vector projects = new Vector();
for (int index = 0; index < 50; i++) {
    Project project = (Project)cursoredEnumerator.nextElement();
    projects.addElement(project);
}
// Rest all at once ...
Vector projects2 = cursoredEnumerator.nextElements(50);
cursoredEnumerator.close();
getTransaction().commit();
```

Executing the Finder From the Client in EJB 2.0

As with EJB 1.1, TopLink offers additional elements for traversing finder results under EJB 2.0 specification. These elements include the following:

- **isEmpty** method: As with `java.util.Collection`, `isEmpty` method returns a boolean value indicating whether or not the `Collection` is empty.
- **size** method: As with `java.util.Collection`, `size` method returns an integer indicating the number of elements in the `Collection`.
- **iterator** method: As with `java.util.Collection`, `iterator` method returns a `java.util.Iterator` for enumerating the elements in the `Collection`.

TopLink also offers an extended protocol for `oracle.toplink.ejb.cmp.wls.CursoredIterator` (based on `java.util.Iterator`):

- **close** method: Closes the cursor on the server. The client must call this method to close the database connection.
- **hasNext** method: Returns a boolean value indicating whether or not any more elements are in the result set.
- **next** method: Returns the next available element.
- **next(int count)** method: Retrieves a `Vector` of at most `count` elements from the available results, depending on how many elements remain in the result set.

[Example 99-18](#) illustrates client code executing a cursored finder.

Example 99-18 *Cursored Finder Under EJB 2.0 Specification*

```
// import both CursoredCollection and CursoredIterator
import oracle.toplink.ejb.cmp.wls.*;
//... other imports as necessary
getTransaction().begin();
CursoredIterator cursoredIterator = (CursoredIterator)
getProjectHome().findByNameCursored("proj%").iterator();
Vector projects = new Vector();
for (int index = 0; index < 50; i++) {
    Project project = (Project)cursoredIterator.next();
    projects.addElement(project);
}
// Rest all at once ...
Vector projects2 = cursoredIterator.next(50);
cursoredIterator.close();
getTransaction().commit();
```

Using Queries and the Cache

This section describes how to use caching options in TopLink queries, including the following:

- [Caching Results in a ReadQuery](#)
- [Configuring Cache Expiration at the Query Level](#)

Caching Results in a ReadQuery

By default, each time you execute a `ReadQuery`, TopLink applies the current query configuration to the read operation. In doing so, TopLink will access the session cache, the data source, or both.

Some queries are known to return the same result set (for example, the number of units sold last year by the current sales person). After the first query execution, there is no need to actually execute the query if it is invoked again.

For these types of queries, you can use any TopLink `ReadQuery` and configure it to store its query results in an internal query cache.

After its first execution for a set of query parameters, the query will return its cached result set each time it is invoked with the same query parameters. This improves query performance for frequently executed queries. By default a query will cache the results sets for the last 100 queries of specific parameters. You can configure this query cache as part of the `QueryResultsCachePolicy`.

Enable this feature using `ReadQuery` method `cacheQueryResults` or by calling the `ReadQuery` method `setQueryResultsCachePolicy` with an instance of `QueryResultsCachePolicy`, and disable it using `ReadQuery` method `doNotCacheQueryResults`.

Before using this feature, consider the restrictions in ["Internal Query Cache Restrictions"](#) on page 96-36. For more information, see ["Caching Query Results"](#) on page 96-36.

You can apply a cache invalidation policy to the query's internal cache (see ["Configuring Cache Expiration at the Query Level"](#) on page 99-22). For more information, see ["Cache Invalidation"](#) on page 90-8.

[Example 99-19](#) shows how to configure a `ReadQuery` to cache its results.

Example 99-19 Configuring a ReadQuery to Cache Its Query Results

```
ReadObjectQuery query = new ReadObjectQuery();
query.setReferenceClass(Employee.class);

// Instruct the ReadQuery to cache its query results
query.cacheQueryResults();

// The first time you invoke it, the ReadQuery reads from the database, session
// cache, or both and stores the result set in its internal query cache
Employee employeeFirst = (Employee) session.executeQuery(query);
```

[Example 99-20](#) shows how to configure the `ReadQuery` to stop caching its results. The next time the query is executed, TopLink does not use the query cache. Instead, the query accesses the data source.

Example 99-20 Configuring a ReadQuery to Stop Caching Its Query Results

```
// Disable query caching
```

```
query.doNotCacheQueryResults();
```

```
// The ReadQuery does not use the query cahce and instead accesses the database  
Employee employee = (Employee) session.executeQuery(query);
```

Alternatively, you can clear the query's internal cache using `ReadQuery` method `clearQueryResults` passing in your session. This clears the currently cached results and ensures that the next query execution reads from the database.

Configuring Cache Expiration at the Query Level

You can configure a `ReadQuery` with a `CacheInvalidationPolicy`.

If you configure a query to cache results in its own internal cache (see ["Caching Results in a ReadQuery"](#) on page 99-21), the cache invalidation policy allows the cached query result set to expire, based on a time-to-live or daily-expiry. This invalidation time is calculated from the time of the query execution that cached the query result set for the specific set of query parameters.

[Example 99-21](#) shows how to configure a `ReadQuery` so that a `TimeToLiveCacheInvalidationPolicy` is applied to all the objects returned by the query and cached in the query's internal cache.

Example 99-21 Configuring a CacheInvalidationPolicy on a ReadQuery for the Query's Internal Cache

```
// The TimeToLiveCacheInvalidationPolicy applies to all objects returned by the query and  
// cached in the query's internal cache  
readQuery.setQueryResultsCachePolicy(  
    new QueryResultsCachePolicy(new TimeToLiveCacheInvalidationPolicy(1000))  
);
```

For more information, see ["Cache Invalidation"](#) on page 90-8.

Part XX

Transactions

This part describes how to use the TopLink unit of work to transactionally perform create, read, update, and delete operations with and without an external transaction processor. It contains the following chapters:

- [Chapter 100, "Understanding TopLink Transactions"](#)

This chapter describes how to use the unit of work, the TopLink wrapper for a transaction, and how TopLink integrates with transaction management and other important query concepts.

- [Chapter 101, "Using Basic Unit of Work API"](#)

This chapter explains how to use basic TopLink unit of work options.

- [Chapter 102, "Using Advanced Unit of Work API"](#)

This chapter explains how to use advanced TopLink unit of work options.

Understanding TopLink Transactions

This chapter explains how transactions are implemented in TopLink, including the following:

- [Unit of Work Architecture](#)
- [Unit of Work Concepts](#)
- [Understanding the Unit of Work API](#)
- [Example Model Object and Schema](#)

Unit of Work Architecture

A database **transaction** is a set of operations (create, read, update, or delete) that either succeed or fail as a single operation. The database discards, or *rolls back*, unsuccessful transactions, leaving the database in its original state. Transactions may be *internal* (that is, provided by TopLink) or *external* (that is, provided by a source external to the application, such as an application server).

In TopLink, transactions are contained in the unit of work object. You acquire a unit of work from a session (see "[Acquiring a Unit of Work](#)" on page 101-1) and using its API, you can control transactions directly or through a Java 2 Enterprise Edition (J2EE) application server transaction controller such as the Java Transaction API (JTA).

Transactions execute in their own context, or logical space, isolated from other transactions and database operations.

The transaction context is demarcated; that is, it has a defined structure that includes:

- A **begin** point, where the operations within the transaction begin. At this point, the transaction begins to execute its operations.
- A **commit** point, where the operations are complete and the transaction attempts to formalize changes on the database.

The degree to which concurrent (parallel) transactions on the same data are allowed to interact is determined by the level of *transaction isolation* configured. ANSI/SQL defines four levels of database transaction isolation as shown in [Table 100-1](#). Each offers a trade-off between performance and resistance from the following unwanted actions:

- Dirty read: a transaction reads uncommitted data written by a concurrent transaction.
- Nonrepeatable read: a transaction rereads data and finds it has been modified by some other transaction that was committed after the initial read operation.

- Phantom read: a transaction reexecutes a query and the returned data has changed due to some other transaction that was committed after the initial read operation.

Table 100–1 Transaction Isolation Levels

Transaction Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read Uncommitted	Yes	Yes	Yes
Read Committed	No	Yes	Yes
Repeatable Read	No	No	Yes
Serializable	No	No	No

As a transaction is committed, the database maintains a log of all changes to the data. If all operations in the transaction succeed, the database allows the changes; if any part of the transaction fails, the database uses the log to roll back the changes.

Like any transaction, a unit of work transaction provides the following:

- [Unit of Work Transaction Context](#)
- [Unit of Work Transaction Demarcation](#)
- [Unit of Work Transaction Isolation](#)

Unit of Work Transaction Context

Unit of work operations occur within a unit of work *context*, in which writes are isolated from the database until commit time. The unit of work executes changes on copies, or *clones*, of objects in its own internal cache, and if successful, applies changes to objects in the database and the session cache.

Unit of Work Transaction Demarcation

In a standalone TopLink application, your application demarcates transactions using the unit of work.

If your application includes a J2EE container that provides container-managed transactions, your application server demarcates transactions using its own transaction service. You configure TopLink to integrate with the container's transaction service by specifying a TopLink external transaction controller.

A TopLink external transaction controller class integrates the unit of work with an application server's transaction service. Using an external transaction controller, your application can participate in transactions that span multiple data sources and that are managed by the application server. The external transaction controller coordinates messages and callbacks between the application server's transaction service and the unit of work.

When you configure your application to use an external transaction controller (see ["Configuring the Server Platform"](#) on page 77-16), the unit of work executes as part of an external transaction. The unit of work still manages its own internal operations, but it waits for the external transaction to tell it to write changes back to the database and to the session cache.

Note that because the transaction happens outside of the unit of work context and is controlled by the application server's transaction service, errors can be more difficult to diagnose and fix because exceptions may occur outside of your application code, for example, during application server initiated call-backs.

You can integrate the unit of work with the following:

- [JTA Controlled Transactions](#)
- [OTS Controlled Transactions](#)
- [CMP Controlled Transactions](#)

JTA Controlled Transactions

The Java Transaction API (JTA) is the application programming interface you use to interact with a transaction manager.

Using JTA, your application can participate in a distributed transaction. A transaction manager that implements JTA provides transaction management and connection pooling and enables your application to interact with multiple data sources transparently by using JTA.

For more information, see ["Integrating the Unit of Work With an External Transaction Service"](#) on page 102-22.

OTS Controlled Transactions

The CORBA Object Transaction Service (OTS) specification is part of the Common Object Request Brokers Architecture (CORBA) Object Services model and is the standard for Object Request Broker (ORB) implementations. Some servers implement the Java APIs for the OTS rather than for JTA (see ["JTA Controlled Transactions"](#) on page 100-3).

Use TopLink OTS support with the unit of work to directly access the Java OTS interfaces of servers that do not support JTA.

To integrate your application with an OTS transaction service, you must configure your application to use a custom server platform (see ["Configuring the Server Platform"](#) on page 77-16).

For more information, see ["Integrating the Unit of Work With an External Transaction Service"](#) on page 102-22.

CMP Controlled Transactions

Entity beans that use container-managed persistence may participate in transactions that are either client demarcated or container demarcated.

A client demarcated transaction occurs when a client of an entity bean directly sets up transaction boundaries using the `javax.transaction.UserTransaction` interface.

A container demarcated transaction occurs when the container automatically wraps an invocation on an EJB in a transaction based upon the transaction attributes supplied in the EJB deployment descriptor.

In transactions involving EJB, TopLink waits until the transaction begins its two-phase commit process before updating the database. This allows for the following:

- SQL optimizations that ensure only changed data is written to the data source
- Proper ordering of updates to allow for database constraints

For more information, see ["Integrating the Unit of Work With CMP"](#) on page 102-26.

Unit of Work Transaction Isolation

The unit of work does not directly participate in database transaction isolation. Because the unit of work may execute queries outside the database transaction (and, by interacting with the cache, outside the database itself), the database does not have control over this data and its visibility.

However, by default, TopLink provides a degree of transaction isolation regardless of what database transaction isolation has been configured on the underlying database.

Each unit of work instance operates on its own copy (clone) of registered objects (see ["Clones and the Unit of Work"](#) on page 100-9). In this case, because the unit of work provides an API that allows querying to be done on object changes within a unit of work (see ["Using Conforming Queries and Descriptors"](#) on page 102-8), the unit of work provides read committed operations.

Optimistic locking, optimistic read locking, or pessimistic locking can be used to further manage concurrency (see ["Locking and the Unit of Work"](#) on page 100-12).

Changes are committed to the database only when the unit of work `commit` method is called (either directly or by way of an external transaction controller).

For detailed information on configuring and using TopLink to achieve a particular level of transaction isolation and transaction isolation level limitations, see ["Database Transaction Isolation Levels"](#) on page 102-28.

Unit of Work Concepts

This section introduces transaction concepts unique to TopLink, including the following:

- [Unit of Work Benefits](#)
- [Unit of Work Life Cycle](#)
- [Unit of Work and Change Policy](#)
- [Clones and the Unit of Work](#)
- [Nested and Parallel Units of Work](#)
- [Commit and Rollback Transactions](#)
- [Primary Keys](#)
- [Unit of Work Optimization](#)
- [Example Model Object and Schema](#)

Unit of Work Benefits

The TopLink unit of work simplifies transactions and improves transactional performance. It is the preferred method of writing to a database in TopLink because it performs the following:

- Sends a minimal amount of SQL to the database during the commit by updating only the exact changes down to the field level
- Reduces database traffic by isolating transaction operations in their own memory space
- Optimizes cache coordination, in applications that use multiple caches, by passing change sets (rather than objects) between caches

- Isolates object modifications in their own transaction space to allow parallel transactions on the same objects
- Ensures referential integrity and minimizes deadlocks by automatically maintaining SQL ordering
- Orders database insert, update, and delete operations to maintain referential integrity for mapped objects
- Resolves bidirectional references automatically
- Frees the application from tracking or recording its changes
- Simplifies persistence with *persistence by reachability* (see ["Associating a New Source to an Existing Target Object"](#) on page 101-6)

Unit of Work Life Cycle

TopLink uses the unit of work as follows:

1. The client application acquires a unit of work from a session object.
2. The client application queries TopLink to obtain a cache object it wants to modify, and then registers the cache object with the unit of work.
3. The unit of work registers the object according to the object's change policy. For more information about how change policy affects registration, see ["Unit of Work and Change Policy"](#) on page 100-7.

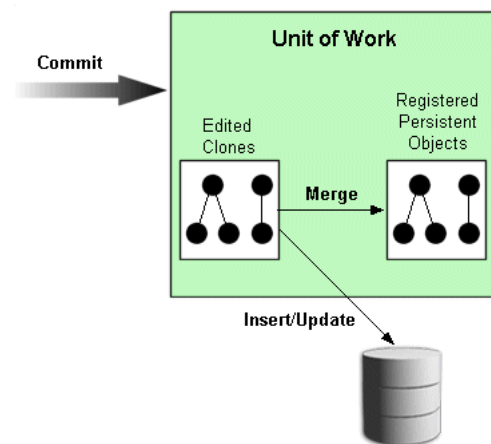
By default, as each object is registered, the unit of work accesses the object from the session cache or database and creates a backup clone and working clone (see ["Clones and the Unit of Work"](#) on page 100-9). The unit of work returns the working clone to the client application.

4. The client application modifies the working object returned by the unit of work.
5. The client application (or external transaction controller) commits the transaction.
6. The unit of work calculates the change set for each registered object according to the object's change policy. For more information about how change policy affects change set calculation, see ["Unit of Work and Change Policy"](#) on page 100-7.

By default, at commit time, the unit of work compares the working clones to the backup clones and calculates the change set (that is, determines the minimum changes required). The comparison is done with a backup clone so that concurrent changes to the same objects will not result in incorrect changes being identified. The unit of work then attempts to commit any new or changed objects to the database.

If the commit transaction succeeds, the unit of work merges changes into the shared session cache. Otherwise, no changes are made to the objects in the shared cache. For more details, see ["Commit and Rollback Transactions"](#) on page 100-10.

If there are no changes, the unit of work does not start a new transaction.

Figure 100–1 The Life Cycle of a Unit of Work

[Example 100–1](#) shows the default life cycle in code.

Example 100–1 Unit of Work Life Cycle

```
// The application reads a set of objects from the database
Vector employees = session.readAllObjects(Employee.class);

// The application specifies an employee to edit
...
Employee employee = (Employee) employees.elementAt(index);

try {
    // Acquire a unit of work from the session
    UnitOfWork uow = session.acquireUnitOfWork();
    // Register the object that is to be changed. Unit of work returns a clone
    // of the object and makes a backup copy of the original employee
    Employee employeeClone = (Employee)uow.registerObject(employee);
    // Make changes to the employee clone by adding a new phoneNumber.
    // If a new object is referred to by a clone, it does not have to be
    // registered. Unit of work determines it is a new object at commit time
    PhoneNumber newPhoneNumber = new PhoneNumber("cell", "212", "765-9002");
    employeeClone.addPhoneNumber(newPhoneNumber);
    // Commit the transaction: unit of work compares the employeeClone with
    // the backup copy of the employee, begins a transaction, and updates the
    // database with the changes. If successful, the transaction is committed
    // and the changes in employeeClone are merged into employee. If there is an
    // error updating the database, the transaction is rolled back and the
    // changes are not merged into the original employee object
    uow.commit();
} catch (DatabaseException ex) {
    // If the commit fails, the database is not changed. The unit of work should
    // be thrown away and application-specific action taken
}
// After the commit, the unit of work is no longer valid. Do not use further
```

Unit of Work and Change Policy

The unit of work tracks changes for a registered object based on the change policy you configure for the object's descriptor. If there are no changes, the unit of work will not start a new transaction.

[Table 100–2](#) lists the change policies that TopLink provides.

Table 100–2 TopLink Change Policies

Change Policy	Applicable to...
Deferred Change Detection Policy	Wide range of object change characteristics. The default change policy.
Object-Level Change Tracking Policy	Objects with few attributes or with many attributes and many changed attributes.
Attribute Change Tracking Policy	Objects with many attributes and few changed attributes. The most efficient change policy. The default change policy for EJB 3.0 or 2.x CMP on OC4J.

For more information, see ["Configuring Change Policy"](#) on page 28-82.

Deferred Change Detection Policy

The `DeferredChangeDetectionPolicy` is the change policy that all persistent objects use by default.

This option provides good unit of work commit performance for a wide range of object change characteristics.

When you register in a unit of work an object whose descriptor is configured with a `DeferredChangeDetectionPolicy` (see ["Configuring Deferred Change Detection Policy"](#) on page 28-83), a backup clone is made of the object (see ["Clones and the Unit of Work"](#) on page 100-9) and at commit time, the unit of work computes changes by making an attribute-by-attribute comparison between the backup clone and the original object.

This change policy is applicable to all mapping types.

Object-Level Change Tracking Policy

The `ObjectChangeTrackingPolicy` optimizes the unit of work commit transaction by including objects in the change set calculation only if at least one attribute has changed.

This option provides improved unit of work commit performance for objects with few attributes, or with many attributes and many changed attributes.

When you register in a unit of work an object whose descriptor is configured with `ObjectChangeTracking` change policy, a backup clone is made of the object and at commit time, the unit of work computes changes by comparing the backup to the current object if and only if at least one attribute is changed (if the object's `hasChanges` method returns `true`). If a registered object has no changes, the unit of work does not compare it to the backup clone.

This change policy is applicable to a subset of mapping types (see ["Change Policy Mapping Support"](#) on page 100-9).

TopLink provides different levels of support for this change policy depending on the EJB version and application server you are using:

EJB 2.n or 3.0 CMP For EJB 2.0 or 3.0 CMP applications deployed to an application server for which TopLink provides CMP integration (see ["Application Server Support"](#) on page 7-1), when you configure a CMP entity bean's descriptor with an `ObjectChangeTrackingPolicy`, TopLink code generates a concrete subclass to implement the `TopLink ChangeTracker` interface at deploy time (see ["Configuring Object Change Tracking Policy"](#) on page 28-83).

EJB 1.*n* CMP and Plain Java Objects For EJB 1.*n* CMP on any application server or plain Java objects, you must implement the `ChangeTracker` interface for each persistent class and configure the class's descriptor with an `ObjectChangeTrackingPolicy` (see "[Configuring Object Change Tracking Policy](#)" on page 28-83).

Attribute Change Tracking Policy

The `AttributeChangeTrackingPolicy` optimizes the unit of work commit transaction by tracking all object changes at the attribute level. This eliminates two unit of work operations: backup clone creation and change detection through comparison.

This option provides improved unit of work commit performance for objects with many attributes, and few changed attributes. Generally, this is the most efficient change policy.

This change policy is applicable to a subset of mapping types (see "[Change Policy Mapping Support](#)" on page 100-9).

Note: You cannot use the `AttributeChangeTrackingPolicy` if you are using any instance of `FieldsLockingPolicy` (see "[Optimistic Field Locking Policies](#)" on page 26-21).

TopLink provides different levels of support for this change policy depending on the EJB version and application server you are using:

EJB 3.0 or 2.*n* CMP on OC4J When using EJB 3.0 or 2.*n* CMP on OC4J, if you want to benefit from this performance enhancement, configure your descriptors with the default `DeferredChangeDetectionPolicy` and allow TopLink to automatically apply an `AttributeChangeTrackingPolicy`. If you configure your project's descriptors with any other change policy, TopLink will honor that configuration and not apply an `AttributeChangeTrackingPolicy`.

When you deploy a TopLink-enabled EJB 3.0 CMP application to OC4J, for each mapped class configured with the default `DeferredChangeDetectionPolicy`, TopLink uses bytecode weaving to automatically override this configuration with an `AttributeChangeTrackingPolicy` and to make the class implement the required interfaces.

When you deploy a TopLink-enabled EJB 2.x CMP application to OC4J, for each mapped class configured with the default `DeferredChangeDetectionPolicy`, TopLink uses code generation to automatically override this configuration with an `AttributeChangeTrackingPolicy` and to make the class implement the required interfaces.

EJB 1.*n* CMP, Plain Java Objects, or Other Application Servers For EJB 1.*n* CMP, plain Java objects, or application servers other than OC4J, to use the `AttributeChangeTrackingPolicy` with a class, you must configure the class's descriptor with an `AttributeChangeTrackingPolicy` and you must implement the `ChangeTracker` interface in that class (see "[Configuring Attribute Change Tracking Policy](#)" on page 28-84).

Change Policy Mapping Support

TopLink supports alternative change tracking policies (policies other than `DeferredChangeDetectionPolicy`) for attributes that use any of the following mapping types:

- [Direct-to-Field Mapping](#)
- [Transformation Mapping](#) (immutable mappings only)
- [One-to-One Mapping](#)
- [Variable One-to-One Mapping](#)
- [One-to-Many Mapping](#)
- [Many-to-Many Mapping](#)
- [Direct Collection Mapping](#)
- [Direct Map Mapping](#)
- [Aggregate Object Mapping](#)
- [EIS Transformation Mapping](#) (immutable mappings only)

TopLink uses the `DeferredChangeDetectionPolicy` (see "[Deferred Change Detection Policy](#)" on page 100-7) for attributes that use any other type of mapping.

Clones and the Unit of Work

When using the `DeferredChangeDetectionPolicy` or the `ObjectLevelChangeTrackingPolicy` (see "[Deferred Change Detection Policy](#)" on page 100-7), the unit of work maintains two copies of the original objects registered with it:

- Working clones
- Backup clones

After you change the working clones and the transaction is committed, the unit of work compares the working clones to the backup clones, and writes any changes to the database. The unit of work uses clones to allow parallel units of work (see "[Nested and Parallel Units of Work](#)" on page 100-10) to exist, a requirement in multiuser three-tier applications.

The TopLink cloning process is efficient in that it clones only the mapped attributes of registered objects, and stops at indirection objects unless you trigger the indirection. For more information, see "[Configuring Indirection](#)" on page 35-4.

You can customize the cloning process using the descriptor's copy policy. For more information, see "[Configuring Copy Policy](#)" on page 28-81.

Never use a clone after committing the unit of work that the clone is from (even if the transaction fails and rolls back). A clone is a working copy used during a transaction and as soon as the transaction is committed (successful or not), the clone must not be used. Accessing an uninstantiated clone value holder after a unit of work commit transaction will raise an exception. The only time you can use a clone after a successful commit transaction is when you use the advanced API described in "[Resuming a Unit of Work After Commit](#)" on page 102-15.

Nested and Parallel Units of Work

You can use TopLink to create the following:

- [Nested Unit of Work](#)
- [Parallel Unit of Work](#)

For additional information and examples on using nested and parallel units of work, see "[Using a Nested or Parallel Unit of Work](#)" on page 102-16.

Nested Unit of Work

You can nest a unit of work (the *child*) within another unit of work (the *parent*). A nested unit of work does not commit changes to the database. Instead, it passes its changes to the parent unit of work, and the parent attempts to commit the changes at commit time. Nesting units of work lets you break a large transaction into smaller isolated transactions, and ensures that:

- Changes from each nested unit of work commit or fail as a group.
- Failure of a nested unit of work does not affect the commit or rollback transaction of other operations in the parent unit of work.
- Changes are presented to the database as a single transaction.

Parallel Unit of Work

You can modify the same objects in multiple unit of work instances in parallel because the unit of work manipulates copies of objects. TopLink resolves any concurrency issues when the Units of Work commits the changes.

Commit and Rollback Transactions

When a unit of work transaction is committed, it either succeeds, or fails and rolls back. A commit transaction can be initiated by your application or by a J2EE container.

Commit Transactions

At commit time, the unit of work compares the working clones and backup clones to calculate the change set (that is, to determine the minimum changes required). Changes include updates to or deletion of existing objects, and the creation of new objects. The unit of work then begins a database transaction, and attempts to write the changes to the database. If all changes commit successfully on the database, the unit of work merges the changed objects into the session cache. If any one of the changes fail on the database, the unit of work rolls back any changes on the database, and does not merge changes into the session cache.

The unit of work calculates commit order using foreign key information from one-to-one and one-to-many mappings. If you encounter constraint problems during a commit transaction, verify your mapping definitions. The order in which you register objects with the `registerObject` method does not affect the commit order.

Commit and JTA When your application uses JTA, the unit of work commit transaction acts differently than in a non-JTA application. In most cases, the unit of work attaches itself to an external transaction. If no transaction exists, the unit of work creates a transaction. This distinction affects commit activity as follows:

- *If the unit of work attaches to an existing transaction*, the unit of work ignores the `commit` call. The transaction commits the unit of work when the entire external transaction is complete.
- *If the unit of work starts the external transaction*, the transaction treats the unit of work `commit` call as a request to commit the external transaction. The external transaction then calls its own commit code on the database.

In either case, only the external transaction can call `commit` on the database because it owns the database connection.

For more information, see ["Integrating the Unit of Work With an External Transaction Service"](#) on page 102-22.

Rollback Transactions

A unit of work commit transaction must succeed or fail as a unit. Failure in writing changes to the database causes the unit of work to roll back the database to its previous state. Nothing changes in the database, and the unit of work does not merge changes into the session cache.

Rollback and JTA In a JTA environment, the unit of work does not own the database connection. In this case, the unit of work sends the rollback call to the external transaction rather than the database, and the external transaction treats the rollback call as a request to roll the transaction back.

For more information, see ["Integrating the Unit of Work With an External Transaction Service"](#) on page 102-22.

Primary Keys

You cannot modify the primary key attribute of an object in a unit of work. This is an unsupported operation and doing so will result in unexpected behaviour (exceptions or database corruption).

To replace one instance of an object with unique constraints with another, see ["Using the Unit of Work `setShouldPerformDeletesFirst` Method"](#) on page 102-18.

Unit of Work Optimization

By default, the unit of work performs change set calculation efficiently for a wide range of object change characteristics. However, there are various ways you can use the unit of work to enhance application performance.

One way to improve performance is to consider using an alternative change policy (see ["Unit of Work and Change Policy"](#) on page 100-7).

For more performance options, see ["Unit of Work Optimization"](#) on page 11-32.

Understanding the Unit of Work API

You do not instantiate an instance of `oracle.toplink.sessions.UnitOfWork`. Rather, you acquire a unit of work from an instance of `oracle.toplink.sessions.Session` or from another unit of work.

For more information on creating sessions, see ["Creating Sessions"](#) on page 76-1.

For more information on acquiring a unit of work, see ["Acquiring a Unit of Work"](#) on page 101-1.

For more information on using the basic API of the unit of work, see ["Using Basic Unit of Work API"](#) on page 101-1.

For more information on using the advanced API of the unit of work, see ["Using Advanced Unit of Work API"](#) on page 102-1.

Unit of Work as Session

The unit of work extends the interface `oracle.toplink.sessions.Session`, and implements all the usual session API. When using session API from a unit of work, you should consider the following:

- [Reading and Querying Objects with the Unit of Work](#)
- [Locking and the Unit of Work](#)

Reading and Querying Objects with the Unit of Work

A unit of work offers the same set of database access methods as a regular session.

When called from a unit of work, these methods access the objects in the unit of work, register the selected objects automatically, and return clones.

Although this makes it unnecessary for you to call the `registerObject` and `registerAllObjects` methods, be aware of the restrictions on registering objects described in ["Creating an Object"](#) on page 101-2 and ["Associating a New Source to an Existing Target Object"](#) on page 101-6.

Reading Objects with the Unit of Work As with regular sessions, you use the `readObject` and `readAllObjects` methods to read objects from the database.

Querying Objects with the Unit of Work You can execute queries in a unit of work with the `executeQuery` method.

Note: Because a unit of work manages changes to existing objects and the creation of new objects, modifying queries such as `InsertObjectQuery` or `UpdateObjectQuery` are not necessary and therefore are not supported by the unit of work.

Locking and the Unit of Work

For information on locking API generic to all sessions, see:

- ["Locking"](#) on page 2-10
- ["Configuring Locking Policy"](#) on page 28-73

For information on locking API specific to a unit of work, see ["Using Optimistic Read Locking With forceUpdateToVersionField"](#) on page 102-19.

Example Model Object and Schema

Throughout the chapters in this part, the following object model and schema are used in the examples provided. The example object model appears in [Figure 100-2](#) and the example entity-relationship (data model) diagram appears in [Figure 100-3](#).

Figure 100-2 Example Object Model

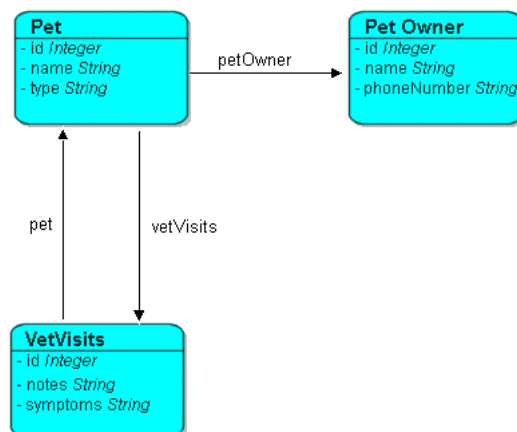
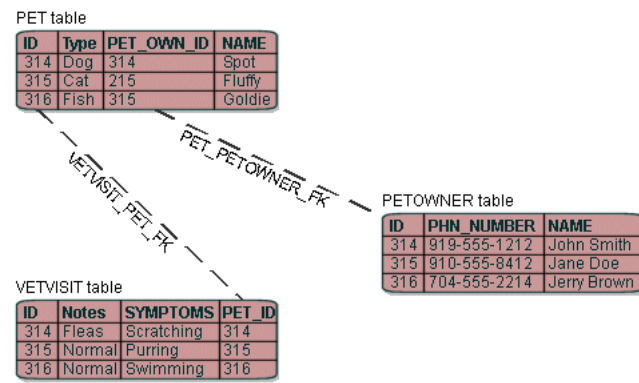


Figure 100-3 Example Data Model

Using Basic Unit of Work API

This chapter explains the essential unit of work API calls most commonly used throughout the development cycle:

- [Acquiring a Unit of Work](#)
- [Creating an Object](#)
- [Modifying an Object](#)
- [Associating a New Target to an Existing Source Object](#)
- [Associating a New Source to an Existing Target Object](#)
- [Associating an Existing Source to an Existing Target Object](#)
- [Deleting Objects](#)

For more information, see [Chapter 102, "Using Advanced Unit of Work API"](#).

Acquiring a Unit of Work

This example shows how to acquire a unit of work from a client session object.

```
Server server =  
    (Server) SessionManager.getManager().getSession(  
        sessionName, MyServerSession.class.getClassLoader()  
    );  
Session session = (Session) server.acquireClientSession();  
UnitOfWork uow = session.acquireUnitOfWork();
```

You can acquire a unit of work from any session type. For more information about acquiring sessions at run time, see ["Acquiring a Session at Run Time With the Session Manager"](#) on page 75-5.

Note that you do not need to create a new session and log in before every transaction. The recommended pattern is to acquire a client session per client access (or thread), and then acquire the necessary unit of work from this client session.

The unit of work is valid until the `commit` or `release` method is called. After a commit or release transaction, a unit of work is not valid even if the transaction fails and is rolled back.

A unit of work remains valid after the `commitAndResume` method is called as described in ["Resuming a Unit of Work After Commit"](#) on page 102-14.

When using a unit of work with JTA, you can also use the advanced API `getActiveUnitOfWork` method as described in ["Integrating the Unit of Work With an External Transaction Service"](#) on page 102-20.

Creating an Object

When you create new objects in the unit of work, use the `registerObject` method to ensure that the unit of work writes the objects to the database at commit time.

The unit of work calculates commit order using foreign key information from one-to-one and one-to-many mappings. If you encounter constraint problems during a commit transaction, verify your mapping definitions. The order in which you register objects with the `registerObject` method does not affect the commit order.

[Example 101-1](#) and [Example 101-2](#) show how to create and persist a simple object (without relationships) using the clone returned by the unit of work `registerObject` method.

Example 101-1 Creating an Object: Preferred Method

```
UnitOfWork uow = session.acquireUnitOfWork();
Pet pet = new Pet();
Pet petClone = (Pet)uow.registerObject(pet);
petClone.setId(100);
petClone.setName("Fluffy");
petClone.setType("Cat");
uow.commit();
```

[Example 101-2](#) shows a common alternative.

Example 101-2 Creating an Object: Alternative Method

```
UnitOfWork uow = session.acquireUnitOfWork();
Pet pet = new Pet();
pet.setId(100);
pet.setName("Fluffy");
pet.setType("Cat");
uow.registerObject(pet);
uow.commit();
```

Both approaches produce the following SQL:

```
INSERT INTO PET (ID, NAME, TYPE, PET_OWN_ID) VALUES (100, 'Fluffy', 'Cat', NULL)
```

[Example 101-1](#) is preferred: it gets you into the pattern of working with clones and provides the most flexibility for future code changes. Working with combinations of new objects and clones can lead to confusion and unwanted results.

Modifying an Object

In [Example 101-3](#), a `Pet` is read prior to a unit of work: the variable `pet` is the cache copy clone for that `Pet`. Inside the unit of work, register the cache copy to get a working copy clone. We then modify the working copy clone and commit the unit of work.

Example 101-3 Modifying an Object

```
// Read in any pet
Pet pet = (Pet)session.readObject(Pet.class);
UnitOfWork uow = session.acquireUnitOfWork();
Pet petClone = (Pet) uow.registerObject(pet);
petClone.setName("Furry");
uow.commit();
```


In [Example 101-4](#), we take advantage of the fact that you can query through a unit of work and get back clones, saving the registration step. However, the drawback is that we do not have a handle to the cache copy clone.

If we wanted to do something with the updated `Pet` after the commit transaction, we would have to query the session to get it (remember that after a unit of work is committed, its clones are invalid and must not be used).

Example 101-4 Modifying an Object: Skipping the Registration Step

```
UnitOfWork uow = session.acquireUnitOfWork();
    Pet petClone = (Pet) uow.readObject(Pet.class);
    petClone.setName("Furry");
uow.commit();
```

Both approaches produce the following SQL:

```
UPDATE PET SET NAME = 'Furry' WHERE (ID = 100)
```

Take care when querying through a unit of work. All objects read in the query are registered in the unit of work and therefore will be checked for changes at commit time. Rather than do a `ReadAllQuery` through a unit of work, it is better for performance to design your application to do the `ReadAllQuery` through a session, and then register in a unit of work only the objects that need to be changed.

Associating a New Target to an Existing Source Object

There are two ways to associate a new target object with an existing source object with one-to-many and one-to-one relationships:

- [Associating Without Reference to the Cache Object](#)
- [Associating With Reference to the Cache Object](#)

Deciding which approach to use depends on whether or not your code requires a reference to the cache copy clone of the new object after the unit of work is committed, and on how adaptable to change you want your code to be.

Associating Without Reference to the Cache Object

[Example 101-5](#) shows the first way of associating a new target with an existing source.

Example 101-5 Associating Without Reference to the Cache Object

```
UnitOfWork uow = session.acquireUnitOfWork();
    Pet petClone = (Pet)uow.readObject(Pet.class);

    PetOwner petOwner = new PetOwner();
    petOwner.setId(400);
    petOwner.setName("Donald Smith");
    petOwner.setPhoneNumber("555-1212");

    VetVisit vetVisit = new VetVisit();
    vetVisit.setId(500);
    vetVisit.setNotes("Pet was shedding a lot.");
    vetVisit.setSymptoms("Pet in good health.");
    vetVisit.setPet(petClone);

    petClone.setPetOwner(petOwner);
    petClone.getVetVisits().addElement(vetVisit);
```

```
uow.commit();
```

This executes the proper SQL:

```
INSERT INTO PETOWNER (ID, NAME, PHN_NBR) VALUES (400, 'Donald Smith', '555-1212')
UPDATE PET SET PET_OWN_ID = 400 WHERE (ID = 100)
INSERT INTO VETVISIT (ID, NOTES, SYMPTOMS, PET_ID) VALUES (500, 'Pet was shedding
a lot.', 'Pet in good health.', 100)
```

When associating new objects to existing objects, the unit of work treats the new object as if it were a clone. That is, after the commit transaction:

```
petOwner != session.readObject(petOwner)
```

For a more detailed discussion of this fact, see ["Using registerNewObject"](#) on page 102-2).

Note: You cannot use `UnitOfWork` methods `registerObject`, `registerNewObject`, or `registerExistingObject` with an aggregate object (see ["Relational Aggregate Descriptors"](#) on page 27-2). Doing so will raise a `ValidationException` or other errors at commit time. For more information, see ["Working with Aggregates"](#) on page 102-6.

Therefore, after the unit of work commit transaction, the variables `vetVisit` and `petOwner` no longer point to their respective cache objects; they point at working copy clones.

If you need the cache object after the unit of work commit transaction, you must query for it or create the association with a reference to the cache object (as described in ["Associating With Reference to the Cache Object"](#) on page 101-4).

Associating With Reference to the Cache Object

[Example 101-6](#) shows how to associate a new target with an existing source with reference to the cache object.

Example 101-6 Associating With Reference to the Cache Object

```
UnitOfWork uow = session.acquireUnitOfWork();
Pet petClone = (Pet)uow.readObject(Pet.class);

PetOwner petOwner = new PetOwner();
PetOwner petOwnerClone = (PetOwner)uow.registerObject(petOwner);
petOwnerClone.setId(400);
petOwnerClone.setName("Donald Smith");
petOwnerClone.setPhoneNumber("555-1212");

VetVisit vetVisit = new VetVisit();
VetVisit vetVisitClone = (VetVisit)uow.registerObject(vetVisit);
vetVisitClone.setId(500);
vetVisitClone.setNotes("Pet was shedding a lot.");
vetVisitClone.setSymptoms("Pet in good health.");
vetVisitClone.setPet(petClone);

petClone.setPetOwner(petOwnerClone);
petClone.getVetVisits().addElement(vetVisitClone);
uow.commit();
```

Now, after the unit of work commit transaction:

```
petOwner == session.readObject(petOwner)
```

This means that we have a handle to the cache copy after the commit transaction, rather than a clone.

[Example 101-7](#) shows how to use unit of work method `registerNewObject` to add a new object when a bidirectional relationship exists. For more information, see ["Using registerNewObject"](#) on page 102-2.

Note: You cannot use `UnitOfWork` methods `registerObject`, `registerNewObject`, or `registerExistingObject` with an aggregate object (see ["Relational Aggregate Descriptors"](#) on page 27-2). Doing so will raise a `ValidationException` or other errors at commit time. For more information, see ["Working with Aggregates"](#) on page 102-6.

Example 101-7 Resolving Issues When Adding New Objects

```
// Get an employee read from the parent session of the unit of work
Employee manager = (Employee)session.readObject(Employee.class);

// Acquire a unit of work
UnitOfWork uow = session.acquireUnitOfWork();

// Register the manager to get its clone
Employee managerClone = (Employee)uow.registerObject(manager);

// Create a new employee
Employee newEmployee = new Employee();
newEmployee.setFirstName("Spike");
newEmployee.setLastName("Robertson");

/* INCORRECT: Do not associate the new employee with the original manager. This
will cause a QueryException when TopLink detects this error during commit */
//newEmployee.setManager(manager);

/* CORRECT: Associate the new object with the clone. Note that in this example,
the setManager method is maintaining the bidirectional managedEmployees
relationship and adding the new employee to its managedEmployees. At commit time,
the unit of work will detect that this is a new object and will take the
appropriate action */
newEmployee.setManager(managerClone);

/* INCORRECT: Do not register the newEmployee: this will create two copies and
cause a QueryException when TopLink detects this error during commit */
//uow.registerObject(newEmployee);

/* CORRECT:
In the above setManager call, if the managerClone's managedEmployees was not
maintained by the setManager method, then you should call registerObject before
the new employee is related to the manager. If in doubt, you could use the
registerNewObject method to ensure that the newEmployee is registered in the unit
of work. The registerNewObject method registers the object, but does not make a
clone */
uow.registerNewObject(newEmployee);
```

```
// Commit the unit of work
uow.commit();
```

Associating a New Source to an Existing Target Object

This section describes how to associate a new source object with an existing target object with one-to-many and one-to-one relationships.

TopLink follows all relationships of all registered objects (deeply) in a unit of work to calculate what is new and what has changed. This is known as **persistence by reachability**. In ["Associating a New Target to an Existing Source Object"](#) on page 101-3, we saw that when you associate a new target with an existing source, you can choose to register the object or not. If you do not register the new object, it is still reachable from the source object (which is a clone, hence it is registered). However, when you need to associate a new source object with an existing target, you must register the new object. If you do not register the new object, then it is not reachable in the unit of work, and TopLink will not write it to the database.

For example, the code shown in [Example 101-8](#) shows how to create a new `Pet` and associate it with an existing `PetOwner`.

Example 101-8 Associating a New Source to an Existing Target Object

```
UnitOfWork uow = session.acquireUnitOfWork();
PetOwner existingPetOwnerClone =
    (PetOwner)uow.readObject(PetOwner.class);

Pet newPet = new Pet();
Pet newPetClone = (Pet)uow.registerObject(newPet);
newPetClone.setId(900);
newPetClone.setType("Lizzard");
newPetClone.setName("Larry");
newPetClone.setPetOwner(existingPetOwnerClone);
uow.commit();
```

This generates the proper SQL:

```
INSERT INTO PET (ID, NAME, TYPE, PET_OWN_ID) VALUES (900, 'Larry', 'Lizzard', 400)
```

In this situation, you should register the new object and work with the working copy of the new object. If you associate the new object with the `PetOwner` clone without registering, it will not be written to the database. If you are in a situation where you want to associate the `PetOwner` clone with the new `Pet` object, use the advanced API `registerNewObject` as described in ["Using registerNewObject"](#) on page 102-2.

Note: You cannot use `UnitOfWork` methods `registerObject`, `registerNewObject`, or `registerExistingObject` with an aggregate object (see ["Relational Aggregate Descriptors"](#) on page 27-2). Doing so will raise a `ValidationException` or other errors at commit time. For more information, see ["Working with Aggregates"](#) on page 102-6.

If you fail to register the clone and accidentally associate the cache version of the existing object with the new object, then TopLink will generate an error which states that you have associated the cache version of an object ("from a parent session") with a clone from this unit of work. You must work with working copies in units of work.

Associating an Existing Source to an Existing Target Object

This section explains how to associate an existing source object with an existing target object with one-to-many and one-to-one relationships.

As shown in [Example 101-9](#), associating existing objects with each other in a unit of work is as simple as associating objects in Java. Just remember to only work with working copies of the objects.

Example 101-9 Associating an Existing Source to Existing Target Object

```
// Associate all VetVisits in the database to a Pet from the database
UnitOfWork uow = session.acquireUnitOfWork();
Pet existingPetClone = (Pet)uow.readObject(Pet.class);
Vector allVetVisitClones;
allVetVisitClones = (Vector)uow.readAllObjects(VetVisit.class);
Enumeration enum = allVetVisitClones.elements();
while(enum.hasMoreElements()) {
    VetVisit vetVisitClone =(VetVisit)enum.nextElement();
    existingPetClone.getVetVisits().addElement(vetVisitClone);
    vetVisitClone.setPet(existingPetClone);
};
uow.commit();
```

The most common error when associating existing objects is failing to work with the working copies. If you accidentally associate a cache version of an object with a working copy you will get an error at commit time indicating that you associated an object from a parent session (the cache version) with a clone from this unit of work.

[Example 101-10](#) shows another example of associating an existing source to an existing target object.

Example 101-10 Associating Existing Objects

```
// Get an employee read from the parent session of the unit of work
Employee employee = (Employee)session.readObject(Employee.class)

// Acquire a unit of work
UnitOfWork uow = session.acquireUnitOfWork();
Project project = (Project) uow.readObject(Project.class);

/* When associating an existing object (read from the session) with a clone, we
must make sure we register the existing object and assign its clone into a unit of
work */

/* INCORRECT: Cannot associate an existing object with a unit of work clone. A
QueryException will be thrown */
//project.setTeamLeader(employee);

/* CORRECT: Instead register the existing object then associate the clone */
Employee employeeClone = (Employee)uow.registerObject(employee);
project.setTeamLeader(employeeClone);
uow.commit();
```

Deleting Objects

To delete objects in a unit of work, use the `deleteObject` or `deleteAllObjects` method. When you delete an object that is not already registered in the unit of work, the unit of work registers the object automatically.

When you delete an object, TopLink deletes the object's privately owned child parts, because those parts cannot exist without the owning (parent) object. At commit time, the unit of work generates SQL to delete the objects, taking database constraints into account.

When you delete an object, you must take your object model into account. You may need to set references to the deleted object to null (for an example, see ["Using privateOwnedRelationship"](#) on page 101-8).

This section explains how to delete objects within a unit of work, including the following:

- [Using privateOwnedRelationship](#)
- [Explicitly Deleting from the Database](#)
- [Understanding the Order in Which Objects Are Deleted](#)

Using privateOwnedRelationship

Relational databases do not have garbage collection like a Java Virtual Machine (JVM) does. To delete an object in Java you just remove the reference to the object. To delete a row in a relational database, you must explicitly delete it. Rather than tediously manage when to delete data in the relational database, use the mapping attribute `privateOwnedRelationship` to have TopLink manage the garbage collection in the relational database for you.

As shown in [Example 101-11](#), when you create a mapping using Java, use its `privateOwnedRelationship` method to tell TopLink that the referenced object is privately owned: that is, the referenced child object cannot exist without the parent object.

Example 101-11 *Specifying a Mapping as Privately Owned*

```
OneToOneMapping petOwnerMapping = new OneToOneMapping();
petOwnerMapping.setAttributeName("petOwner");
petOwnerMapping.setReferenceClass(com.top.uowprimer.model.PetOwner.class);
petOwnerMapping.privateOwnedRelationship();
petOwnerMapping.addForeignKeyFieldName("PET.PET_OWN_ID", "PETOWNER.ID");
descriptor.addMapping(petOwnerMapping);
```

When you create a mapping using TopLink Workbench, you can select the **Private Owned** check box under the **General** tab.

When you tell TopLink that a relationship is privately owned, you are specifying that:

- If the source of a privately owned relationship is deleted, then delete the target.
- If you remove the reference to a target from a source, then delete the target.

Do not configure privately owned relationships to objects that might be shared. An object should not be the target in more than one relationship if it is the target in a privately owned relationship.

The exception to this rule is the case when you have a many-to-many relationship in which a relation object is mapped to a relation table and is referenced through a one-to-many relationship by both the source and the target. In this case, if the one-to-many mapping is configured as privately owned, then when you delete the source, all the association objects will be deleted.

Consider the example shown in [Example 101-12](#).

Example 101–12 Privately Owned Relationships

```
// If the Pet-PetOwner relationship is privatelyOwned
// then the PetOwner will be deleted at uow.commit()
// otherwise, just the foreign key from PET to PETOWNER will
// be set to null. The same is true for VetVisit
UnitOfWork uow = session.acquireUnitOfWork();
    Pet petClone = (Pet)uow.readObject(Pet.class);
    petClone.setPetOwner(null);
    VetVisit vvClone =
        (VetVisit)petClone.getVetVisits().firstElement();
    vvClone.setPet(null);
    petClone.getVetVisits().removeElement(vvClone);
uow.commit();
```

If the relationships from `Pet` to `PetOwner` and from `Pet` to `VetVisit` are not privately owned, this code produces the following SQL:

```
UPDATE PET SET PET_OWN_ID = NULL WHERE (ID = 150)
UPDATE VETVISIT SET PET_ID = NULL WHERE (ID = 350)
```

If the relationships are privately owned, this code produces the following SQL:

```
UPDATE PET SET PET_OWN_ID = NULL WHERE (ID = 150)
UPDATE VETVISIT SET PET_ID = NULL WHERE (ID = 350)
DELETE FROM VETVISIT WHERE (ID = 350)
DELETE FROM PETOWNER WHERE (ID = 250)
```

Explicitly Deleting from the Database

If there are cases where you have objects that will not be garbage collected through privately owned relationships (especially root objects in your object model), then you can explicitly tell `TopLink` to delete the row representing the object using the `deleteObject` API, as shown in [Example 101–13](#).

Example 101–13 Explicitly Deleting

```
UnitOfWork uow = session.acquireUnitOfWork();
    pet petClone = (Pet)uow.readObject(Pet.class);
    uow.deleteObject(petClone);
uow.commit();
```

The preceding code generates the following SQL:

```
DELETE FROM PET WHERE (ID = 100)
```

Understanding the Order in Which Objects Are Deleted

The unit of work does not track changes or the order of operations. It is intended to insulate you from having to modify your objects in the order the database requires.

By default, at commit time, the unit of work correctly puts in order all insert and update operations using the constraints defined by your schema. After all insert and update operations are done, the unit of work will issue the necessary delete operations.

Constraints are inferred from one-to-one and one-to-many mappings. If you have no such mappings, you can add additional constraint knowledge to `TopLink` as described in ["Controlling the Order of Delete Operations"](#) on page 102-16.

Using Advanced Unit of Work API

This chapter explains the advanced unit of work API calls and techniques commonly used later in the development cycle, including the following:

- [Registering and Unregistering Objects](#)
- [Declaring Read-Only Classes](#)
- [Writing Changes Before Commit Time](#)
- [Using Conforming Queries and Descriptors](#)
- [Merging Changes in Working Copy Clones](#)
- [Resuming a Unit of Work After Commit](#)
- [Reverting a Unit of Work](#)
- [Using a Nested or Parallel Unit of Work](#)
- [Using a Unit of Work With Custom SQL](#)
- [Controlling the Order of Delete Operations](#)
- [Using Optimistic Read Locking With `forceUpdateToVersionField`](#)
- [Implementing User and Date Auditing with the Unit of Work](#)
- [Integrating the Unit of Work With an External Transaction Service](#)
- [Integrating the Unit of Work With CMP](#)
- [Database Transaction Isolation Levels](#)
- [Troubleshooting a Unit of Work](#)

For more information about the available methods for the `UnitOfWork`, see *Oracle TopLink API Reference*.

Registering and Unregistering Objects

The unit of work provides the following object registration options:

- [Creating and Registering an Object in One Step](#)
- [Using `registerNewObject`](#)
- [Using `registerAllObjects`](#)
- [Using Registration and Existence Checking](#)
- [Working with Aggregates](#)
- [Unregistering Working Clones](#)

Creating and Registering an Object in One Step

[Example 102-1](#) shows how to use the unit of work `newInstance` method to create a new `Pet` object, register it with the unit of work, and return a clone, all in one step. If you are using a factory design pattern to create your objects (and have specified this in the query builder), the `newInstance` method will use the appropriate factory.

Example 102-1 *Creating and Registering an Object in One Step*

```
UnitOfWork uow = session.acquireUnitOfWork();
    Pet petClone = (Pet)uow.newInstance(Pet.class);
    petClone.setId(100);
    petClone.setName("Fluffy");
    petClone.setType("Cat");
uow.commit();
```

Using `registerNewObject`

This section examines how to use the `registerNewObject` method, including the following:

- [Registering a New Object with `registerNewObject`](#)
- [Associating New Objects With One Another](#)

Note: You cannot use `UnitOfWork` methods `registerObject`, `registerNewObject`, or `registerExistingObject` with an aggregate object (see ["Relational Aggregate Descriptors"](#) on page 27-2). Doing so will raise a `ValidationException` or other errors at commit time. For more information, see ["Working with Aggregates"](#) on page 102-6.

Registering a New Object with `registerNewObject`

The `registerNewObject` method registers a new object as if it was a clone. At commit time, the unit of work creates another instance of the object to be the cache version of that object.

Use `registerNewObject` in situations where:

- You do not need a handle to the cache version of the object after the commit transaction and you do not want to work with clones of new objects.
- You must pass a clone into the constructor of a new object, then register the new object.

[Example 102-2](#) shows how to register a new object with the `registerNewObject` method.

Example 102-2 *Registering a New Object with the `registerNewObject` Method*

```
UnitOfWork uow = session.acquireUnitOfWork();
    PetOwner existingPetOwnerClone =
        PetOwner(uow.readObject(PetOwner.class));

    Pet newPet = new Pet();
    newPet.setId(900);
    newPet.setType("Lizzard");
    newPet.setName("Larry");
    newPet.setPetOwner(existingPetOwnerClone);
```

```
uow.registerNewObject(newPet);
uow.commit();
```

By using `registerNewObject`, the variable `newPet` should not be used after the unit of work is committed. The new object is the clone and if you need the cache version of the object, you need to query for it. If you needed a handle to the cache version of the `Pet` after the unit of work has been committed, then you should use the first approach described in ["Associating a New Source to an Existing Target Object"](#) on page 101-6. In that example, the variable `newPet` is the cache version after the unit of work is committed.

Associating New Objects With One Another

At commit time, `TopLink` can determine if an object is new or not. As described in ["Associating a New Target to an Existing Source Object"](#) on page 101-3, if a new object is reachable from a clone, you do not need to register it. `TopLink` effectively uses the `registerNewObject` method on all new objects it can reach from registered objects.

When working with new objects, remember the following rules:

- Only reachable or registered objects will be persisted.
- New objects or objects that have been registered with `registerNewObject` are considered to be working copies in the unit of work.
- If you call `registerObject` with a new object, the clone, and the argument, is considered the cache version.

[Example 102-3](#) shows how to associate new objects with the `registerNewObject` method.

Example 102-3 Associating New Objects with the `registerNewObject` Method

```
UnitOfWork uow = session.acquireUnitOfWork();
Pet newPet = new Pet();
newPet.setId(150);
newPet.setType("Horse");
newPet.setName("Ed");

PetOwner newPetOwner = new PetOwner();
newPetOwner.setId(250);
newPetOwner.setName("George");
newPetOwner.setPhoneNumber("555-9999");

VetVisit newVetVisit = new VetVisit();
newVetVisit.setId(350);
newVetVisit.setNotes("Talks a lot");
newVetVisit.setSymptoms("Sore throat");

newPet.getVetVisits().addElement(newVetVisit);
newVetVisit.setPet(newPet);
newPet.setPetOwner(newPetOwner);

uow.registerNewObject(newPet);
uow.commit();
```

However, after the unit of work commit, do not use the objects in the variables `newPet`, `newPetOwner`, and `newVetVisit` because they are technically copies from the unit of work.

If you need a handle to the cache version of these objects, query for them or use the unit of work as shown in [Example 102-4](#).

Example 102-4 Associating New Objects with the `newObjectMethod` and Retaining a Handle to the Cache Objects

```
UnitOfWork uow = session.acquireUnitOfWork();
Pet newPet = new Pet();
Pet newPetClone = (Pet)uow.registerObject(newPet);
newPetClone.setId(150);
newPetClone.setType("Horse");
newPetClone.setName("Ed");

PetOwner newPetOwner = new PetOwner();
PetOwner newPetOwnerClone =
    (PetOwner)uow.registerObject(newPetOwner);
newPetOwnerClone.setId(250);
newPetOwnerClone.setName("George");
newPetOwnerClone.setPhoneNumber("555-9999");

VetVisit newVetVisit = new VetVisit();
VetVisit newVetVisitClone =
    (VetVisit)uow.registerObject(newVetVisit);
newVetVisitClone.setId(350);
newVetVisitClone.setNotes("Talks a lot");
newVetVisitClone.setSymptoms("Sore throat");

newPetClone.getVetVisits().addElement(newVetVisitClone);
newVetVisitClone.setPet(newPetClone);
newPetClone.setPetOwner(newPetOwnerClone);
uow.commit();
```

Using `registerAllObjects`

The `registerAllObjects` method takes a `Collection` of objects as an argument and returns a `Collection` of clones. This lets you register many objects at once as shown in [Example 102-5](#).

Note: You cannot use `UnitOfWork` methods `registerObject`, `registerNewObject`, or `registerExistingObject` with an aggregate object (see "[Relational Aggregate Descriptors](#)" on page 27-2). Doing so will raise a `ValidationException` or other errors at commit time. For more information, see "[Working with Aggregates](#)" on page 102-6.

Example 102-5 Using `registerAllObjects`

```
UnitOfWork uow = session.acquireUnitOfWork();
Collection toRegister = new Vector(2);
VetVisit vv1 = new VetVisit();
vv1.setId(70);
vv1.setNotes("May have flu");
vv1.setSymptoms("High temperature");
toRegister.add(vv1);

VetVisit vv2 = new VetVisit();
vv2.setId(71);
```

```

vv2.setNotes("May have flu");
vv2.setSymptoms("Sick to stomach");
toRegister.add(vv2);

uow.registerAllObjects(toRegister);
uow.commit();

```

Using Registration and Existence Checking

When you register an object with the unit of work, TopLink runs an existence check to determine whether or not the object exists. TopLink uses this information at commit time to determine whether to perform an insert or an update operation. You can specify the default existence checking policy for a project as a whole (see ["Configuring Existence Checking at the Project Level"](#) on page 22-10) or on a per-descriptor basis (["Configuring Cache Existence Checking at the Descriptor Level"](#) on page 28-50). By default, TopLink uses the check cache existence checking policy. If you use any existence checking policy other than check cache, then you can use the way you register your objects to your advantage to reduce the time it takes TopLink to register an object.

This section explains how to use one of the following existence checking policies to accelerate object registration:

- [Check Database](#)
- [Assume Existence](#)
- [Assume Nonexistence](#)

Check Database

If you configure a class's descriptor with an existence checking policy of check database, TopLink will check the database for existence for all instances of that class registered in a unit of work. However, if you know that an object is new or existing, rather than use the basic `registerObject` method, you can use `registerNewObject` or `registerExistingObject` to bypass the existence check. TopLink will not check the database for existence on objects that you have registered with these methods. It will automatically perform an insert operation if `registerNewObject` is called, or an update operation if `registerExistingObject` is called.

Assume Existence

If you configure a class's descriptor with an existence checking policy of assume existence, TopLink will assume that all instances of that class registered with a unit of work exist and TopLink will always perform an update operation to the database on all such registered objects; even new objects that you registered with `registerObject` method. However, if you use the `registerNewObject` method on the new object, TopLink knows to perform an insert operation in the database even though the existence checking policy says assume existence.

Assume Nonexistence

If you configure a class's descriptor with an existence checking policy of assume nonexistence then TopLink assumes that all instances of that class registered with a unit of work do not exist and will always perform an insert operation on the database, even on objects read from the database. However, if you use the `registerExistingObject` method on existing objects, TopLink knows to perform an update operation on the database.

Working with Aggregates

Aggregate mapped objects should never be registered in a TopLink unit of work—doing so will generate an exception. Aggregate cloning and registration is automatic based on the owner of the aggregate object. In other words, if you register the owner of an aggregate, the aggregate is automatically cloned. When you get a working copy of an aggregate owner, its aggregate is also a working copy.

When working with aggregates, you should always use an aggregate *within the context of its owner*:

- If you get an aggregate from a working clone owner, then the aggregate is a working clone.
- If you get an aggregate from a cache version owner, then the aggregate is the cache version.

For more information about aggregate objects, see ["Relational Aggregate Descriptors"](#) on page 27-2.

Unregistering Working Clones

The unit of work `unregisterObject` method lets you unregister a previously registered object from a unit of work. An unregistered object will be ignored in the unit of work, and any uncommitted changes made to the object up to that point will be discarded.

In general, this method is rarely used. It can be useful if you create a new object, but then decide to delete it in the same unit of work (which is not recommended).

Declaring Read-Only Classes

You can declare a class as read-only within the context of a unit of work. Clones are neither created nor merged for such classes, thus improving performance. Such classes are ineligible for changes in the unit of work.

When a unit of work registers an object, it traverses and registers the entire object tree. If the unit of work encounters a read-only class, it does not traverse that branch of the tree, and does not register objects referenced by the read-only class, so those classes are ineligible for changes in the unit of work. The read-only classes are cached and must not be changed by the user.

Configuring Read-Only Classes for a Single Unit of Work

For example, suppose class A owns a class B, and class C extends class B. You acquire a unit of work in which you know only instances of class A will change: you know that no class Bs will change. Before registering an instance of class B, use the following:

```
myUnitOfWork.addReadOnlyClass(B.class);
```

You can then proceed with your transaction: registering class A objects, modifying their working copies, and committing the unit of work.

At commit time, the unit of work will not have to compare backup clones with the working clones for instances of class B (even if instances were registered explicitly or implicitly). This can improve unit of work performance if the object tree is very large.

Note that if you register an instance of class C, the unit of work does not create or merge clones for this object; any changes made to class C are not be persisted because class C extends class B and class B was identified as read-only.

To identify multiple classes as read-only, add them to a `Vector` and use the following code:

```
myUnitOfWork.addReadOnlyClasses(myVectorOfClasses);
```

Note that a nested unit of work inherits the set of read-only classes from the parent unit of work. For more information on using a nested unit of work, see ["Using a Nested or Parallel Unit of Work"](#) on page 102-15.

Configuring Default Read-Only Classes

To establish a default set of read-only classes for all units of work, use the project method `setDefaultReadOnlyClasses(Vector)`. After you call this method, all new units of work include the `Vector` of read-only classes.

Read-Only Descriptors

When you declare a class as read-only, the read-only declaration extends to its descriptors. You can declare a descriptor as read-only at development time, using either Java code or TopLink Workbench. This option improves performance by excluding the read-only descriptors from unit of work registration and editing.

To flag descriptors as read-only in Java code, call the `setReadOnly` method on the descriptor as follows:

```
descriptor.setReadOnly();
```

To declare a descriptor as read-only in TopLink Workbench, select the **Read Only** check box for the specific descriptor.

For more information, see ["Configuring Read-Only Descriptors"](#) on page 28-5.

Writing Changes Before Commit Time

By default, when you call the unit of work `commit` method, TopLink writes your changes to the data source and commits your changes.

Alternatively, you can perform a two-stage or partial commit transaction by using the unit of work `writeChanges` method prior to calling `commit` (either directly or by way of an external transaction service).

When you call the unit of work `writeChanges` method, the unit of work `commit` process begins, and all changes are written out to the data source. However, the data source transaction is not committed, nor will changes be merged into the shared session cache. To finalize your changes, the unit of work `commit` method must still be called (either directly or by way of an external transaction service).

After you call the unit of work `writeChanges` method, any attempt to register objects or execute object-level queries will throw an exception. You may execute report queries, noncaching queries, and data read and modify queries.

If any exception is thrown, the transaction will be rolled back (or marked rollback only) and you cannot recover the unit of work.

You can call this method only once. You cannot use this method to write out changes in an incremental fashion.

You can use the unit of work `writeChanges` method to address a variety of transaction issues, including the following:

- As an alternative to conforming (see ["Using Unit of Work Method writeChanges Instead of Conforming"](#) on page 102-12)
- To handle external transaction issues (see ["Using the Unit of Work to Handle External Transaction Timeouts and Exceptions"](#) on page 102-24)

Using Conforming Queries and Descriptors

Because queries are executed on the database, querying through a unit of work will not, by default, include new, uncommitted objects in a unit of work. The unit of work will not spend time executing your query against new, uncommitted objects in the unit of work unless you explicitly tell it to. If you have uncommitted changes, this can pose a problem in a unit of work. Uncommitted changes not yet written to the database cannot influence which result set gets returned.

Conforming is a query feature that lets you include new, changed, or deleted objects in queries within a unit of work prior to committing. This lets you to query against your relative logical or transaction view of the database.

Before you use conforming, be aware of its limitations (see ["Guidelines for Using Conforming"](#) on page 102-8) and make sure that conforming is actually necessary. For example, consider the alternative described in ["Conforming Query Alternatives"](#) on page 102-12.

This section explains the following:

- [Guidelines for Using Conforming](#)
- [Using Conforming Queries](#)
- [Using Conforming Descriptors](#)
- [Conforming Query Alternatives](#)

Note: By default, TopLink suppresses exceptions thrown during the memory search stage of conforming. For more information on handling exceptions during conforming, see ["Exceptions During Conforming"](#) on page 102-35.

Guidelines for Using Conforming

When using conforming, follow the guidelines that this section describes to ensure that conforming queries return the correct results:

- [Ensure That the Query Supports Conforming](#)
- [Consider how Conforming Affects Database Results](#)
- [Register New Objects and Instantiate Relationships](#)

Ensure That the Query Supports Conforming

Conforming is supported by the following queries:

- `ReadObjectQuery`
- `ReadAllQuery`
- Expressions
- EJB QL
- Query by example

- Query by selection object or primary key (only new or deleted objects apply)

Conforming is not supported by the following queries:

- `ReportQuery`
- `DataReadQuery`
- `DataReadQuery` (deleted objects can be conformed)
- `StoredProcedureCall` (deleted objects can be conformed)
- `EISCall` (deleted objects can be conformed)
- Expression or EJB QL queries that use database-specific functions, or subselects.

Consider how Conforming Affects Database Results

When conforming is used on a `ReadAllQuery`, the database result is first queried. If the unit of work has not yet committed any changes to the database, this result will not reflect the unit of work changes. The database results are then conformed in memory using the following criteria:

- Registered new objects that conform to the query are added.
- Modified existing objects that no longer conform are removed.
- Modified existing objects that conform are added.
- Deleted objects are removed.

Note: If new objects are not explicitly registered, they are not conformed. Also, if removed object are not explicitly deleted, they are not conformed.

If the query uses ordering, ordering of conformed results is not maintained and conformed instances are added to the front of the result. To apply ordering, store the result in memory using `Collections` method `sort`, or a `TreeSet` result collection class. When using conforming on a `ReadObjectQuery`, first query the unit of work for a conforming object: if the conforming object is found, it is returned and the database access is avoided; if the conforming object is not found, the database is queried. If the unit of work has not yet committed any changes to the database, this result does not reflect the unit of work changes. The database results are then conformed in memory using the following criteria:

- If the database result no longer conforms, null is returned.
- If the database result has been deleted, null is returned.

Note: If the database result returns multiple results, only the first result is considered, because it is an instance of the `ReadObjectQuery` and only a single result is expected. If the first result no longer conforms, null is returned, even if there were potential valid conforming results.

If you expect the query to return multiple results, use a `ReadObjectQuery`.

Register New Objects and Instantiate Relationships

If a new object is only related to an existing object, and not explicitly registered, queries for this object are not able to conform it. If you remove, but do not explicitly delete a privately owned object, this object cannot be conformed.

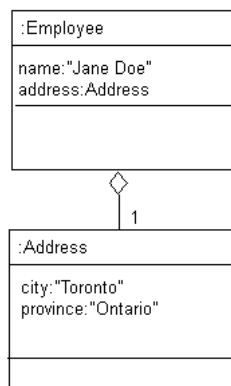
If a query traverses relationships (uses joins) and the related objects are changed, the query can only conform these objects if both of the following conditions are met:

- The source objects have been registered in the unit of work.
- The source objects' relationship has been instantiated.

TopLink provides a conforming option that forces an instantiation of indirection. However, you use this option with caution as it can cause an increased database access.

Consider the example shown in [Example 102-1](#). In this example, you have `Employee` objects with an `address` attribute configured for indirection (see "[Indirection](#)" on page 33-5) mapped by a one-to-one mapping to an `Address` object.

Figure 102-1 Conforming Example



You want to read all employees who live in Ottawa, but first, you need to modify some of the `Address` objects to change city from Toronto to Ottawa. Jane Doe is one such employee.

First, using the `UnitOfWork`, you read all `Address` objects and change some `city` attributes (including Jane's) from Toronto to Ottawa. Then you run a conforming query to get all employees who live in Ottawa. However, for the following reasons Jane is not included in the results, even though she now lives in Ottawa:

- Jane is not returned from the database because the transaction has not yet been committed and in the database, her address still says Toronto.
- Jane cannot be added to the conformed result in memory because she is not registered in the `UnitOfWork` cache.

Conforming only recognizes *explicit* changes. In this example, Jane Doe's `Employee` object was only *implicitly* changed. In order to be considered explicitly changed, an `Employee` must meet the following criteria:

- be registered in the `UnitOfWork`
- have its `address` attribute changed (in this example, indirection must be triggered for the `address` attribute)

The correct way to handle this example would be as follows:

1. Using the `UnitOfWork`, read in all employees.
All these `Employee` objects are now registered with the `UnitOfWork`
2. Using the same `UnitOfWork`, access the employees' addresses, instantiating the indirect relationships.
3. Modify the employees' addresses, changing some of the addresses to be in Ottawa.
4. Run the conforming query on employees with addresses inside Ottawa.
All employees with addresses in Ottawa are returned, including both employees that were in Ottawa originally and employees whose addresses were changed in this transaction.
5. Commit the transaction.

If you do not register all employees whose address may be changed, and instantiate their address relationship, the conforming query will not include Jane.

An alternate approach is to use short transactions: the safest conforming query is one made immediately after a commit. For example:

1. Using the `UnitOfWork`, read in all addresses outside of Ottawa.
2. Modify the addresses, changing some of the addresses to be in Ottawa
3. Commit the transaction.
4. Using the `UnitOfWork`, read in all employees inside Ottawa.

Using Conforming Queries

Assume that a single `Pet` of type `Cat` already exists on the database. Examine the code shown in [Example 102-6](#).

Example 102-6 Using Conforming Queries

```
UnitOfWork uow = session.acquireUnitOfWork();
Pet pet2 = new Pet();
Pet petClone = (Pet)uow.registerObject(pet2);
petClone.setId(200);
petClone.setType("Cat");
petClone.setName("Mouser");

ReadAllQuery readAllCats = new ReadAllQuery();
readAllCats.setReferenceClass(Pet.class);
ExpressionBuilder builder = new ExpressionBuilder();
Expression catExp = builder.get("type").equal("Cat");
readAllCats.setSelectionCriteria(catExp);

Vector allCats = (Vector)uow.executeQuery(readAllCats);

System.out.println("All 'Cats' read through UOW are: " + allCats);
uow.commit();
```

This produces the following output:

```
All 'Cats' read through UOW are: [Pet type Cat named Fluffy id:100]
```

If you tell the query `readAllCats` to include new objects:

```
readAllCats.conformResultsInUnitOfWork();
```

The output would be:

All 'Cats' read through UOW are: [Pet type Cat named Fluffy id:100, Pet type Cat named Mouser id:200]

Using Conforming Descriptors

TopLink's support for conforming queries in the unit of work can be specified at the descriptor level.

You can define a descriptor directly to always conform results in the unit of work so that all queries performed on this descriptor conform its results in the unit of work by default. You can specify this either within code or from TopLink Workbench.

You can configure a descriptor to always conform in the unit of work using TopLink Workbench or Java code.

To configure a descriptor to always conform in the unit of work in Java code, use `Descriptor` method `setShouldAlwaysConformResultsInUnitOfWork`, passing in an argument of `true`.

To configure a descriptor to always conform in the unit of work using TopLink, see ["Configuring Unit of Work Conforming at the Descriptor Level"](#) on page 28-7.

Conforming Query Alternatives

This section describes alternatives to conforming that may meet your needs without the performance penalty imposed by conforming. This section describes the following:

- [Using Unit of Work Method `writeChanges` Instead of Conforming](#)
- [Using Unit of Work Properties Instead of Conforming](#)

Using Unit of Work Method `writeChanges` Instead of Conforming

Using `UnitOfWork` method `writeChanges`, you can write uncommitted changes to the data source: you can execute report queries, noncaching queries, and data read and modify queries against these changes (see [Example 102-7](#)).

Example 102-7 Using Unit of Work Method `writeChanges`

```
UnitOfWork uow = session.acquireUnitOfWork();
    Pet pet = new Pet();
    Pet petClone = (Pet)uow.registerObject(pet);
    petClone.setId(100);
    petClone.setName("Fluffy");
    petClone.setType("Cat");

uow.writeChanges();

// Use uow to perform report, noncaching, and data read and modify queries
// against the changes made so far

uow.commit();
```

However, you can call `writeChanges` only once; any attempt to register objects or to execute object-level queries will throw an exception.

For more information, see ["Writing Changes Before Commit Time"](#) on page 102-7

Using Unit of Work Properties Instead of Conforming

Sometimes, you need to provide other code modules with access to new objects created in a unit of work. Conforming can be used to provide this access. However, the following alternative is significantly more efficient.

Somewhere a unit of work is acquired from a session and is passed to multiple modules for portions of the requisite processing:

```
UnitOfWork uow = session.acquireUnitOfWork();
```

In the module that creates the new employee, note the following:

```
Pet newPet = new Pet();
Pet newPetClone = (Pet)uow.registerObject(newPet);
uow.setProperty("NEW PET", newPet);
```

In other modules where `newPet` needs to be accessed for further modification, it can simply be extracted from the unit of work's properties:

```
Pet newPet = (Pet) uow.getProperty("NEW PET");
newPet.setType("Dog");
```

Conforming queries are ideal if you are not sure if an object has been created yet or the criteria is dynamic.

However, for situations where the quantity of objects is finite and well known, using unit of work properties is a simple and more efficient solution.

Merging Changes in Working Copy Clones

In a three-tier application, the client and server exchange objects using a serialization mechanism such as RMI or CORBA. When the client changes an object and returns it to the server, you cannot register this serialized object into a unit of work directly. On the server, you must merge the serialized object with the original object in the session cache.

Using the unit of work methods listed in [Table 102–1](#), you can merge a deserialized object into your session cache. Each method takes the serialized object as an argument and returns the original object.

Before doing so, you must ensure that the source object is in your session cache. Attempting to merge a deserialized object into a session cache that does not yet contain the object will result in a descriptor exception (see ["200: ATTEMPT_TO_REGISTER_DEAD_INDIRECTION"](#) on page 13-24). To avoid this, Oracle recommends that you first read the object instance that the deserialized object represents. If you are using a coordinated cache or your application is running in a cluster, the session you merge into may not yet contain your original object. By performing a read operation first, you guarantee that the object will be in the cache before you merge.

Table 102–1 Unit of Work Merge Methods

Method	Purpose	Used When
<code>mergeClone</code>	Merges the serialized object and all its privately owned parts (excluding non-private references from it to independent objects) into the working copy clone.	The client edits the object but not its relationships, or marks its independent relationships as transient.
<code>mergeCloneWithReferences</code>	Merges the serialized object and all references into the working copy clone.	The client edits the object and the targets of its relationships and has not marked any attributes as transient.

Table 102–1 (Cont.) Unit of Work Merge Methods

Method	Purpose	Used When
<code>shallowMergeClone</code>	Merges only serialized object changes to attributes mapped with direct mappings into the working copy clone.	The client edits only the object's direct attributes or has marked all of the object's relationships as transient.
<code>deepMergeClone</code>	Merges the serialized object and everything connected to it (the entire object tree where the serialized object is the root) into the working copy clone.	The client traverses all relationships of the objects and makes changes. Note: Use <code>deepMergeClone</code> with caution. If two different copies of an object are in the same tree, <code>TopLink</code> will merge one set of changes over the other. You should not have any transient attributes in any of your related objects.

Note that if your three-tier client is sufficiently complex, consider using the `TopLink` remote session (see ["Remote Sessions"](#) on page 75-30). It automatically handles merging and lets you use a unit of work on the client.

You can merge clones with both existing and new objects. Because clones do not appear in the cache and may not have a primary key, you can merge new objects only once within a unit of work. If you need to merge a new object more than once, call the unit of work `setShouldNewObjectsBeCached` method, and ensure that the object has a valid primary key; you can then register the object.

[Example 102–8](#) shows one way to update the original object with the changes contained in the corresponding serialized object (`rmiClone`) received from a client.

Example 102–8 Merging a Serialized Object

```
update(Object original, Object rmiClone) {
    original = uow.registerObject(original);
    uow.mergeCloneWithRefereneces(rmiClone);
    uow.commit();
}
```

For more information, see ["Indirection, Serialization, and Detachment"](#) on page 33-10.

Resuming a Unit of Work After Commit

At commit time, a unit of work and its contents expire: you must not use the unit of work nor its clones even if the transaction failed and rolled back.

However, `TopLink` offers an API that lets you continue working with a unit of work and its clones:

- `commitAndResume`: Commits the unit of work, but does not invalidate it or its clones.
- `commitAndResumeOnFailure`: Commits the unit of work. If the commit transaction succeeds, the unit of work expires. However, if the commit transaction fails, this method does not invalidate the unit of work or its clones. This method lets you modify the registered objects in a failed unit of work and retry the commit transaction.

You should resume a unit of work only in an application that makes repeated changes to the same, small dataset. Reusing the same unit of work while accessing different datasets may result in poor performance.

[Example 102-9](#) shows how to use the `commitAndResume` method.

Example 102-9 Using the `commitAndResume` Method

```
UnitOfWork uow = session.acquireUnitOfWork();
    PetOwner petOwnerClone =
        (PetOwner)uow.readObject(PetOwner.class);
    petOwnerClone.setName("Mrs. Newowner");
    uow.commitAndResume();
    petOwnerClone.setPhoneNumber("KL5-7721");
uow.commit();
```

The `commitAndResume` call produces this SQL:

```
UPDATE PETOWNER SET NAME = 'Mrs. Newowner' WHERE (ID = 400)
```

Then, the `commit` call produces this SQL:

```
UPDATE PETOWNER SET PHN_NBR = 'KL5-7721' WHERE (ID = 400)
```

Reverting a Unit of Work

Under certain circumstances, you may want to abandon some or all changes to clones in a unit of work, but not abandon the unit itself. The following options exist for reverting all or part of the unit of work:

- `revertObject`: Abandons changes to a specific working copy clone in the unit of work
- `revertAndResume`: Uses the backup copy clones to restore all clones to their original states, deregister any new objects, and reinstate any deleted objects.

Using a Nested or Parallel Unit of Work

You can use a unit of work within another unit of work (nesting), or you can use two or more units of work with the same objects in parallel.

Parallel Unit of Work

To start multiple units of work that operate in parallel, call the `acquireUnitOfWork` method multiple times on the session. The units of work operate independently of one another and maintain their own cache.

Nested Unit of Work

To nest units of work, call the `acquireUnitOfWork` method on the parent unit of work. This creates a child unit of work with its own cache. If a child unit of work commits, it updates the parent unit of work rather than the database. If the parent does not commit, the changes made to the child are not written to the database.

`TopLink` does not update the database or the cache until the outermost unit of work is committed. You must commit or release the child unit of work before you can commit its parent.

Working copy clones from one unit of work are not valid in another units of work: not even between an inner and outer unit of work. You must register objects at all levels of a unit of work where they are used.

[Example 102-10](#) shows how to use nested units of work.

Example 102–10 Using Nested Units of Work

```
UnitOfWork outerUOW = session.acquireUnitOfWork();
    Pet outerPetClone = (Pet)outerUOW.readObject(Pet.class);

    UnitOfWork innerUOWa = outerUOW.acquireUnitOfWork();
        Pet innerPetCloneA =
            (Pet)innerUOWa.registerObject(outerPetClone);
        innerPetCloneA.setName("Muffy");
    innerUOWa.commit();

    UnitOfWork innerUOWb = outerUOW.acquireUnitOfWork();
        Pet innerPetCloneB =
            (Pet)innerUOWb.registerObject(outerPetClone);
        innerPetCloneB.setName("Duffy");
    innerUOWb.commit();
outerUOW.commit();
```

Using a Unit of Work With Custom SQL

You can execute native SQL or invoke a stored procedure within a unit of work by using unit of work method `executeNonSelectingCall` or by executing a `DataModifyQuery`. This makes the unit of work begin its database transaction early and execute the call to the data immediately.

If you release the unit of work, it will roll back the database changes. If you commit the unit of work and the commit succeeds, the unit of work will commit the changes to the database.

You can execute a `DataModifyQuery` only in a unit of work or a database session. You cannot execute a `DataModifyQuery` in a client or server session directly.

You can execute a `DataReadQuery` or use session method `executeSelectingCall` in any session type because these do not modify data.

[Example 102–11](#) illustrates using `SQLCall` with the unit of work method `executeNonSelectingCall`.

Example 102–11 Using the `executeNonSelectingCall` Method

```
uow.executeNonSelectingCall(new SQLCall(mySqlString));
```

WARNING: Allowing an unverified SQL string to be passed into methods makes your application vulnerable to SQL injection attacks.

Controlling the Order of Delete Operations

"Deleting Objects" on page 101-7 explained that `TopLink` always properly arranges (orders) the SQL based on the mappings and foreign keys in your object model and schema. You can control the order of delete operations by the following:

- [Using the Unit of Work `setShouldPerformDeletesFirst` Method](#)
- [Using the Descriptor `addConstraintDependencies` Method](#)

Using the Unit of Work `setShouldPerformDeletesFirst` Method

By default, TopLink does insert and update operations first, before delete operations, to ensure that referential integrity is maintained. This is the preferred approach.

If you are forced to replace an object with unique constraints by deleting it and inserting a replacement, you may cause a constraint violation if the insert operation occurs before the delete operation. In this case, call `setShouldPerformDeletesFirst` to perform the delete operation before the insert operation.

Using the Descriptor `addConstraintDependencies` Method

The constraints used by TopLink to determine delete order are inferred from one-to-one and one-to-many mappings. If you do not have such mappings, you can add constraint knowledge to TopLink using the descriptor `addConstraintDependencies(Class)` method.

For example, suppose you have a composition of objects: A contains B (one-to-many, privately owned) and B has a one-to-one, nonprivate relationship with C. You want to delete A (and in doing so the included Bs) but before deleting the Bs, for some of them (not all) you want to delete the associated object C.

There are two possible solutions:

1. [Using `deleteAllObjects` Without `addConstraintDependencies`](#)
2. [Using `deleteAllObjects` with `addConstraintDependencies`](#)

Using `deleteAllObjects` Without `addConstraintDependencies`

In the first option, you do not identify the one-to-many (A to B) relationship as privately owned. When deleting an A object, you must delete all of its B objects, as well as any C objects, as shown in the following example:

```
uow.deleteObject(existingA);
uow.deleteAllObjects(existingA.getBs());
// delete one of the Cs
uow.deleteObject(((B) existingA.getBs().get(1)).getC());
```

This option produces the following SQL:

```
DELETE FROM B WHERE (ID = 2)
DELETE FROM B WHERE (ID = 1)
DELETE FROM A WHERE (ID = 1)
DELETE FROM C WHERE (ID = 1)
```

Using `deleteAllObjects` with `addConstraintDependencies`

In the second option, keep the one-to-many (A to B) relationship privately owned and add a constraint dependency from A to C, as shown in the following example:

```
session.getDescriptor(A.class).addConstraintDependencies(C.class);
```

Now the delete code would be:

```
uow.deleteObject(existingA);
// delete one of the Cs
uow.deleteObject(((B) existingA.getBs().get(1)).getC());
```

This option produces the following SQL:

```
DELETE FROM B WHERE (A = 1)
DELETE FROM A WHERE (ID = 1)
DELETE FROM C WHERE (ID = 1)
```

In both cases, the B object is deleted before A and C. The main difference is that the second option generates fewer SQL statements, as it knows that it is deleting the entire set of Bs related from A.

Using Optimistic Read Locking With forceUpdateToVersionField

If your descriptors are configured to use an optimistic version locking policy (see ["Optimistic Version Locking Policies"](#) on page 26-18) or field locking policy (see ["Optimistic Field Locking Policies"](#) on page 26-21), use the unit of work method `forceUpdateToVersionField` to solve either or both of the following problems:

- You want an `OptimisticLockingException` thrown at commit time if an object you read in a transaction has changed since it was registered even though you have not changed the object in your transaction (see ["Forcing a Check of the Optimistic Read Lock"](#) on page 102-18).
- You modify an object in a transaction in such a way that its version field is not updated but you want to alert other threads of the change by way of the version field (see ["Forcing a Version Field Update"](#) on page 102-19).

For example, you change a privately owned object that has its own database table so the parent object's version field is not, by default, updated. In this case, you can use `forceUpdateToVersionField` to update the parent's version field.

As an alternative to this approach, consider ["Optimistic Version Locking Policies and Cascading"](#) on page 26-19.

To remove `forceUpdateToVersionField` configuration from an object before a commit operation, use the unit of work method `removeForceUpdateToVersionField` (see ["Disabling forceUpdateToVersionField"](#) on page 102-21).

Forcing a Check of the Optimistic Read Lock

When you read an object with the unit of work, optimistic lock checking is not applied to that object at commit time unless you change the object. However, there are times when you want your transaction to fail if the state of an object has changed since it was read, even though you have not modified the object.

[Example 102-12](#) shows a transaction that updates a mortgage rate by multiplying the central bank prime rate by 1.25. The transaction forces an optimistic read lock on the central prime rate at commit time to ensure that the prime rate has not changed since the transaction began. Note that in this example, the transaction does not increment the version of the unchanged object (the central prime rate).

Example 102-12 Optimistic Read Lock with No Version Increment

```
try {
    UnitOfWork uow = session.acquireUnitOfWork();
    MortgageRate cloneMortgageRate = (MortgageRate)
        uow.registerObject(mortgageRate);
    CentralPrimeRate cloneCentralPrimeRate = (CentralPrimeRate)
        uow.registerObject(CentralPrimeRate);
    /* Change the Mortgage Rate */
    cloneMortgageRate.setRate(cloneCentralPrimeRate.getRate() * 1.25);
```

```

        /* Optimistic read lock check on Central prime rate with no version update */
        uow.forceUpdateToVersionField(cloneCentralPrimeRate, false);
        uow.commit();
    } catch (OptimisticLockException exception) {
        /* Refresh the out-of-date object */
        session.refreshObject(exception.getObject());
        /* Retry... */
    }
}

```

For another example that forces both optimistic locking and a version field update, see [Example 102–13](#) in ["Forcing a Version Field Update"](#) on page 102-19.

Forcing a Version Field Update

The unit of work considers an object changed when you modify its direct-to-field or aggregate object mapping attribute. Adding, removing, or modifying objects related to the source object does not render the source object changed for the purposes of the unit of work. In other words, when a relationship is changed in a one-to-many or one-to-one target foreign key mapping, by default, the version field (if any) of the affected object is not changed.

If you configure a descriptor to refresh the cache only if the database version is newer than the cache version (using descriptor method `onlyRefreshCacheIfNewerVersion`), and such a relationship changes, you will not be able to refresh the object at all. Because the version has not changed, the unit of work method `refreshObject` and even a query with `refreshIdentityMapResults` option set to `true` cannot refresh the object.

Using the unit of work method `forceUpdateToVersionField` passing in both the unit of work copy clone and `true` value will ensure that the object's version field is updated when such a change is made. It will also ensure that changes to the object before it is refreshed will result in optimistic locking exceptions, preventing the writing of stale data (see ["Forcing a Check of the Optimistic Read Lock"](#) on page 102-18).

[Example 102–13](#) and [Example 102–14](#) show transactions executing in separate threads that access the same customer object concurrently. The unit of work method `forceUpdateToVersionField` is used to ensure that changes to the customer object in one thread are detected by the other threads.

[Example 102–13](#) shows a transaction in which an *invoice* thread calculates an invoice for a customer. [Example 102–14](#) shows a transaction in which another thread, the *service* thread, adds a service to the same customer or modifies the current service. In either case, the *service* thread must inform the *invoice* thread, which adds the changes to the invoice.

Example 102–13 Optimistic Read Lock with Version Increment: Service Thread

```

/* The following code represents the service thread. Notice that the thread forces
a version update */
try {
    UnitOfWork uow = session.acquireUnitOfWork();
    Customer cloneCustomer = (Customer uow.registerObject(customer));
    Service cloneService = (Service uow.registerObject(service));
    /* Add a service to customer */
    cloneService.setCustomer(cloneCustomer);
    cloneCustomer.getServices().add(cloneService);
    /* Modify the customer version to inform other application that
    the customer has changed */
}

```

```

        uow.forceUpdateToVersionField(cloneCustomer, true);
        uow.commit();
    }
    catch (OptimisticLockException exception) {
        /* Refresh out-of-date object */
        session.refreshObject(exception.getObject());
        /* Retry... */
    }
}

```

Example 102–14 Optimistic Read Lock with Version Increment: Invoice Thread

/* The following code represents the invoice thread, and calculates a bill for the customer. Notice that it does not force an update to the version */

```

try {
    UnitOfWork uow = session.acquireUnitOfWork();
    Customer cloneCustomer = (Customer) uow.registerObject(customer);
    Invoice cloneInvoice = (Invoice) uow.registerObject(new Invoice());
    cloneInvoice.setCustomer(cloneCustomer);
    /* Calculate service charge */
    int total = 0;
    for(Enumeration enum = cloneCustomer.getServices().elements();
        enum.hasMoreElements();) {
        total += ((Service) enum.nextElement()).getCost();
    }
    cloneInvoice.setTotal(total);
    /* Force optimistic lock checking on the customer to guarantee a valid
       calculation */
    uow.forceUpdateToVersionField(cloneCustomer, false);
    uow.commit();
}
catch(OptimisticLockException exception) {
    /* Refresh the customer and its privately owned parts */
    session.refreshObject(cloneCustomer);
    /* If the customer's services are not privately owned then use a
       ReadObjectQuery to refresh all parts */
    ReadObjectQuery query = new ReadObjectQuery(customer);
    /* Refresh the cache with the query's result and cascade refreshing
       to all parts including customer's services */
    query.refreshIdentityMapResult();
    query.cascadeAllParts();
    /* Refresh from the database */
    query.dontCheckCache();
    session.executeQuery(query);
    /* Retry... */
}
}

```

Disabling forceUpdateToVersionField

The `forceUpdateToVersionField` configuration you apply to an object stays in effect for the lifetime of your unit of work. After you commit your transaction, `forceUpdateToVersionField` configuration no longer applies.

To remove `forceUpdateToVersionField` configuration from an object before commit time, use the unit of work method `removeForceUpdateToVersionField`. TopLink will not apply optimistic read locking to the object unless you change it in this transaction (that is, unless you modify its direct-to-field or aggregate object mapping attribute).

Implementing User and Date Auditing with the Unit of Work

Auditing datasource changes is a common requirement: when a user commits a change to the datasource, the application updates a field in the datasource to record the user who made the change and the date.

For example, suppose each row in your database schema includes fields `lastUpdateBy` (to record the user name of the user who commits a change) and `lastUpdateOn` (to record the date of the change).

You can use `UnitOfWork` method `setProperty` to record the name of the user who acquires the `UnitOfWork` and implement a descriptor event listener for `AboutToUpdateEvent` descriptor events that extracts the property and updates the `lastUpdateBy` and `lastUpdateOn` fields.

For more information, see the following:

- ["Acquiring a Unit of Work"](#) on page 101-1
- ["Configuring a Domain Object Method as an Event Handler"](#) on page 28-67
- ["Configuring a Descriptor Event Listener as an Event Handler"](#) on page 28-71

Integrating the Unit of Work With an External Transaction Service

To support transactions managed by an application server's external transaction service, `TopLink` supports external connection pools and external transaction controller classes for supported servers. This lets you incorporate external transaction service support into your application, and use the unit of work with transactions managed externally by the server. For more information, see ["Unit of Work Transaction Demarcation"](#) on page 100-2.

To integrate a unit of work with an external transaction service, you must enable the use of the following:

- an external transaction controller (see ["Configuring the Server Platform"](#) on page 77-16)
- an external connection pool (see ["Configuring External Connection Pooling"](#) on page 85-3)

After you configure external connection pool and external transaction controller support, you use a unit of work in your `TopLink` application as you would typically, with few exceptions. This section describes these exceptions as follows:

- [Acquiring a Unit of Work With an External Transaction Service](#)
- [Using a Unit of Work When an External Transaction Exists](#)
- [Using a Unit of Work When No External Transaction Exists](#)
- [Using the Unit of Work to Handle External Transaction Timeouts and Exceptions](#)

Acquiring a Unit of Work With an External Transaction Service

You use a unit of work to commit changes to a data source even when using an external transaction service. To ensure that only one unit of work is associated with a given transaction, use the `getActiveUnitOfWork` method to acquire a unit of work as shown in [Example 102-15](#).

Note: Although there are other ways to commit changes to a data source using an external transaction service, Oracle recommends using the `getActiveUnitOfWork` method.

The `getActiveUnitOfWork` method searches for an existing external transaction in the following way:

- If there is an active external transaction and a unit of work is already associated with it, return this unit of work.
- If there is an active external transaction with no associated unit of work, then acquire a new unit of work, associate it with the transaction, and return it.
- If there is no active external transaction in progress, return null.

If `TopLink` returns a unit of work that is not null, use it exactly as you would typically: the only exception is that you do not call the `commit` method (see ["Using a Unit of Work When an External Transaction Exists"](#) on page 102-22).

If `TopLink` returns a null unit of work, start an external transaction explicitly through the `UserTransaction` interface.

Example 102-15 Using a Unit of Work With an External Transaction Service

```
// Read in any pet
Pet pet = (Pet)clientSession.readObject(Pet.class);
UnitOfWork uow = clientSession.getActiveUnitOfWork();
if (uow == null) {
    throw new RuntimeException("No transaction started");
}
Pet petClone = (Pet)uow.registerObject(pet);
petClone.setName("Furry");
```

Using a Unit of Work When an External Transaction Exists

When `getActiveUnitOfWork` returns a unit of work that is not null, you are associated with an existing external transaction. Use the unit of work as usual.

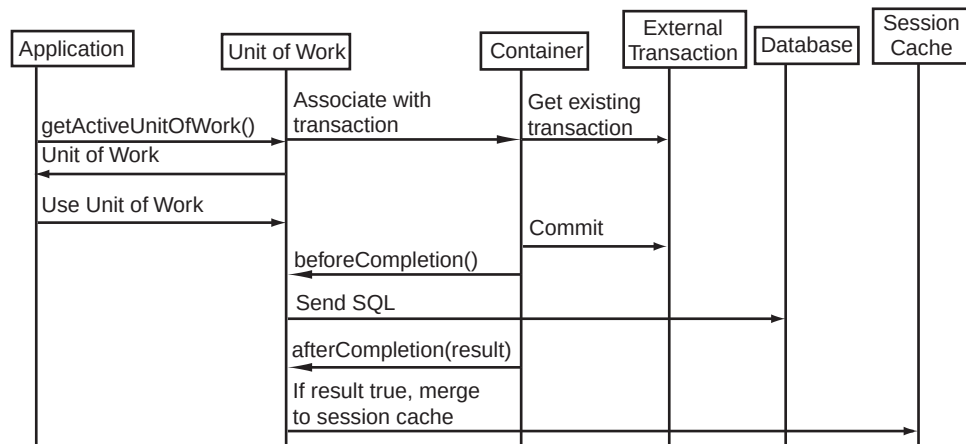
As the external transaction was not started by the unit of work, issuing a `commit` on it will not cause the external transaction to be committed. The unit of work will defer to the application or container that began the transaction. When the external transaction does get committed by the container, `TopLink` receives synchronization callbacks at key points during the commit transaction.

The unit of work sends the required SQL to the database when it receives the `beforeCompletion` callback.

The unit of work uses the `Boolean` argument received from the `afterCompletion` callback to determine if the commit was successful (`true`) or not (`false`).

If the commit transaction was successful, the unit of work merges changes to the session cache. If the commit transaction was unsuccessful, the unit of work discards the changes.

[Figure 102-2](#) shows the life cycle of a unit of work when an external transaction exists.

Figure 102–2 Unit of Work When an External Transaction Exists

Using a Unit of Work When No External Transaction Exists

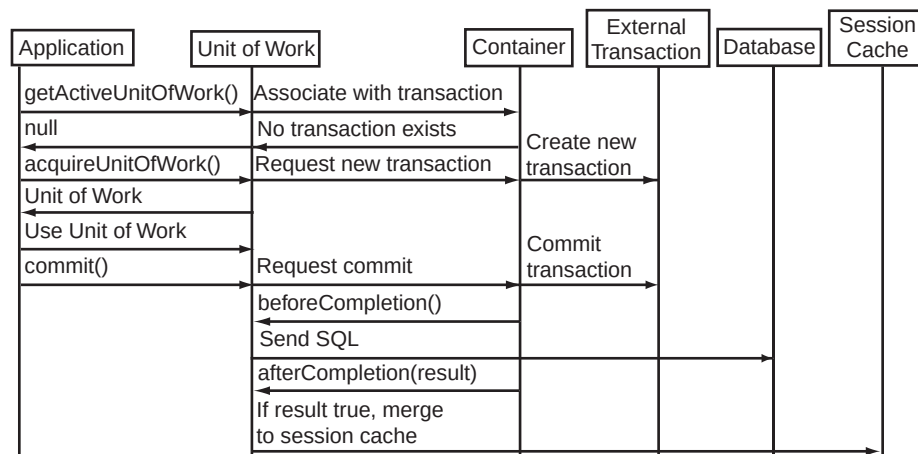
When the `getActiveUnitOfWork` method returns a null unit of work, there is no existing external transaction. You must start a new external transaction.

Do this either by starting an external transaction explicitly using the `UserTransaction` interface, or by acquiring a new unit of work using the `acquireUnitOfWork` method on the server session.

Use the unit of work as usual.

Once the modifications to registered objects are complete, you must commit the transaction either explicitly through the `UserTransaction` interface or by calling the unit of work `commit` method.

The transaction synchronization callbacks are then invoked on, and the database updates and cache merge occur based upon those callbacks.

Figure 102–3 Unit of Work When No External Transaction Exists

Using the Unit of Work to Handle External Transaction Timeouts and Exceptions

This section describes two common problems with external transactions:

- [External Transaction Commit Timeouts](#)

- [External Transaction Commit Exceptions](#)

External Transaction Commit Timeouts

When an external transaction is committed, the external transaction service expects each transaction owner to commit its portion of the overall transaction within a finite amount of time. If any individual transaction exceeds this timeout interval, the external transaction service will fail the specific transaction and roll it back (or mark it rollback only).

If your transaction is large and its commit transaction may exceed the external transaction service timeout interval, use `UnitOfWork` method `writeChanges` to write changes to the data source before committing the external transaction. This will reduce the time it takes for your part of the global transaction to commit.

For more information about the `UnitOfWork` method `writeChanges`, including restrictions and warnings, see ["Writing Changes Before Commit Time"](#) on page 102-7.

External Transaction Commit Exceptions

When you use the unit of work with an external transaction service, commit exceptions may not be thrown until long after your application thread calls its `UnitOfWork` method `commit` and returns. In this case, commit exceptions are thrown to the client of the container-managed transaction (CMT) call, forcing the client to handle this server-side failure.

You can use the `UnitOfWork` method `writeChanges` to write changes to the data source before the external transaction commits. This allows your application thread to catch and handle most exceptions that could be thrown at the time the external transaction service commits the global transaction.

For more information about the `UnitOfWork` method `writeChanges`, including restrictions and warnings, see ["Writing Changes Before Commit Time"](#) on page 102-7.

For more information on handling unit of work exceptions in general, see ["Handling Exceptions"](#) on page 102-34.

Integrating the Unit of Work With CMP

All modifications to persistent beans should be carried out in the context of a transaction.

Modifying entity beans without a transaction can lead to an inconsistent state, potentially corrupting the values in the TopLink cache. Because of this, TopLink does not support modifying a bean through its remote interface when no transaction is active. If you attempt to do so, TopLink simply does not write changes to the database.

Although TopLink does not let you modify an entity bean through its remote interface without a transaction, TopLink *does* let you invoke methods on its home interface that change the state in the underlying database without a transaction. For example, you may invoke `remove` and `create` methods on the home interface of an entity bean without a transaction.

To integrate TopLink transactions and the unit of work with container-managed persistence (CMP), you must consider the following:

- [CMP Transaction Attribute](#)
- [Local Transactions](#)
- [Nondeferred Changes](#)

CMP Transaction Attribute

To ensure that all modifications to persistent beans are carried out in the context of a transaction, transactional attributes must be properly specified in the bean deployment descriptors.

The transaction may be either client-controlled or container-controlled.

Client-controlled transactions are started explicitly by your application by way of the `javax.transaction.UserTransaction` interface.

Container-controlled transactions are started implicitly by the container to satisfy the transaction attribute configuration when a bean method is invoked in the absence of a client-controlled transaction.

[Table 102-2](#) shows what transaction (if any) an EJB method invocation uses depending on how its transaction attribute is configured and whether or not a client-controlled transaction exists at the time the method is invoked.

Oracle recommends that you do not make modifications to entity beans under conditions identified as "Use no transaction" in [Table 102-2](#). Oracle also recommends that you avoid using the `Supports` transaction attribute because it leads to a nontransactional state whenever the client does not explicitly provide a transaction.

Table 102-2 EJB Transaction State by Transaction Attribute

Transaction Attribute	Client-Controlled Transaction Exists	Client-Controlled Transaction Does Not Exist
<code>NotSupported</code>	Use no transaction	Use no transaction
<code>Supports</code>	Use client-controlled transaction	Use no transaction
<code>Required</code>	Use client-controlled transaction	Use container-controlled transaction
<code>RequiresNew</code>	Use client-controlled transaction	Use container-controlled transaction
<code>Mandatory</code>	Use client-controlled transaction	Exception raised
<code>Never</code>	Exception raised	Use no transaction

Depending on the CMP container you use, you may be able to write without a container-controlled transaction (see ["Integrating the Unit of Work With an External Transaction Service"](#) on page 102-21). In this case, TopLink automatically uses a transaction of its own, referred to as a local transaction (see ["Local Transactions"](#) on page 102-26).

Local Transactions

Some CMP containers (such as OC4J) support writing without an active JTA transaction.

If you execute a bean method outside a JTA transaction while the transaction attribute (see ["CMP Transaction Attribute"](#) on page 102-25) is set to `Supports`, `NotSupported`, or `Never`, TopLink performs the operation within a local unit of work and commits the unit of work at the end of the method. This unit of work is referred to as a local transaction.

The reason for this is because the update semantics in the EJB specification are left undefined for these scenarios, and a proper transactional model demands that a transaction be active before being able to modify data. TopLink also requires change operations to occur within a unit of work to ensure that the session cache remains consistent.

Nondeferred Changes

Some CMP containers (such as OC4J) support nondeferred changes: the ability to modify the data source immediately as you change the persistent fields of an entity bean.

Using nondeferred changes, you can achieve backwards compatibility with the native behavior of some CMP containers (such as OC4J) and you can accommodate advanced applications that rely on the database and entity changes being synchronized for such things as triggers or stored procedures based on transient state within the transaction, deletion and creation of rows with the same primary key, or other complex queries that depend on transient transaction state.

Nondeferred changes have the disadvantage of being the least efficient approach: they produce the greatest number of data source interactions.

By default, TopLink defers all changes until commit time. This is the most efficient approach that produces the least number of data source interactions.

For more information, see ["Nondeferred Changes"](#) on page 26-3.

Database Transaction Isolation Levels

Achieving a particular database transaction isolation level in a TopLink application is more involved than simply using the `DatabaseLogin` method `setTransactionIsolation`.

In a typical TopLink application and in J2EE applications that require persistence in general, a variety of factors affect when database transaction isolation levels apply and to what extent a particular database transaction isolation can be achieved.

This section describes these factors and provides guidelines on configuring and using TopLink to achieve each database transaction isolation level to the extent possible given these factors.

This section includes the following:

- [General Factors Affecting Transaction Isolation Level](#)
- [Read Uncommitted Level](#)
- [Read Committed Level](#)
- [Repeatable Read Levels](#)
- [Serializable Read Levels](#)

General Factors Affecting Transaction Isolation Level

This section describes some of the important factors and variables that may affect the degree to which your TopLink application can achieve a particular database transaction isolation level. These factors include the following:

- [External Applications](#)
- [TopLink Coordinated Cache](#)
- [DatabaseLogin Method `setTransactionIsolation`](#)
- [Reading Through the Write Connection](#)
- [Managing Cache Access](#)
- [CMP and External Transactions](#)

External Applications

In many cases, your TopLink application is not the only application that can update to the database. External, non-TopLink applications, can also update the database at any time.

In this case, your TopLink application must use the `ObjectLevelReadQuery` method `refreshIdentityMapResult` (see ["Refreshing the Cache"](#) on page 96-35) or `Descriptor` methods `alwaysRefreshCache` and `disableCacheHits` (see ["Configuring Cache Refreshing"](#) on page 28-32).

For more information, see ["Managing Cache Access"](#) on page 102-29.

If the external application can update a version field in the database, your TopLink application could use `alwaysRefreshCache` in conjunction with `Descriptor` method `onlyRefreshCacheIfNewerVersion` to ensure that refresh operations are performed only when required. Another, recommended way to achieve this, is to use the descriptor isolated cache option (see ["Cache Isolation"](#) on page 90-9), as well as cache invalidation (see ["Cache Invalidation"](#) on page 90-8).

TopLink Coordinated Cache

Consider multiple TopLink applications (each running on its own application server instance) configured to use a distributed, coordinated cache (as described in ["Understanding Cache Coordination"](#) on page 90-10). A TopLink application instance first commits changes to its own cache before the change is distributed to other caches. Because cache coordination is not instantaneous, there is a possibility that one TopLink application instance may read an older version of an object from its cache before a cache coordination message is received.

To provide your TopLink application with the most up-to-date version of an object use the descriptor isolated cache option (see ["Cache Isolation"](#) on page 90-9), as well as cache invalidation (see ["Cache Invalidation"](#) on page 90-8).

You can also avoid stale data by using `Descriptor` methods `alwaysRefreshCache` and `disableCacheHits`. For more information on the `disableCacheHits` method, see ["Managing Cache Access"](#) on page 102-29.

Caution: Using `Descriptor` methods `alwaysRefreshCache` and `disableCacheHits` will result in frequent database hits. Use only when absolutely necessary.

DatabaseLogin Method setTransactionIsolation

Use the `DatabaseLogin` method `setTransactionIsolation` to configure the database transaction isolation level that TopLink applies to any database connection it obtains, for example:

```
databaseLogin.setTransactionIsolation(DatabaseLogin.TRANSACTION_SERIALIZABLE);
```

This method sets the transaction isolation level used for both database read and write operations on the database connections obtained from either an internal or external connection pool (see ["Connection Pools"](#) on page 75-3), for both internal transactions and external transactions as in the case of CMP.

However, with TopLink, by default read operations use a different database connection than write operations, typically obtained from an external connection pool, or may use the cache, bypassing the database entirely. Thus, with TopLink, by default, read operations are always performed outside the transaction or unit of work, even if

you perform the read operation within a transaction or unit of work. Although database transaction isolation applies to both read and write connections, the read is not performed as part of the transaction. Therefore, the read operation overrides the transaction isolation set on the database.

Depending on the level of transaction isolation you are trying to achieve, you may require that the same transaction isolation be applied to both read and write operations. You must take special action to make TopLink use the same connection for both read and write operations. For more information, see ["Reading Through the Write Connection"](#) on page 102-28.

Reading Through the Write Connection

Recall that TopLink, by default, performs read operations with a different database connection than used for write operations ("[DatabaseLogin Method setTransactionIsolation](#)" on page 102-28). However, from the perspective of database transaction isolation, there is a one-to-one relationship between transaction and database connection: that is, all database operations (including read operations) must use the same database connection in order to achieve a particular database transaction isolation level.

In general, when TopLink performs a read operation, if a write connection already exists, TopLink will use the write connection for the read operation. This is called "reading through the write connection." If a write connection does not yet exist, TopLink will acquire another connection and use that for the read operation.

You can configure TopLink to allocate a write connection early using any of the following:

- [Pessimistic Locking Query](#)
- [Unit of Work Method beginTransactionEarly](#)
- [ConnectionPolicy Method setShouldUseExclusiveConnection](#)

Caution: Depending on the database transaction isolated level reading through the write connection may lock the object being read. This will affect performance and reduce concurrency. Oracle recommends that you do not use these advanced techniques unless strict database transaction isolation is absolutely necessary.

For more information, see ["CMP and External Transactions"](#) on page 102-30.

Pessimistic Locking Query When you use pessimistic locking (ObjectLevelReadQuery methods `acquireLocks` or `acquireLocksWithoutWaiting` or Session method `refreshAndLockObject`), TopLink does the following:

- Allocates a write connection used for both read and write operations.
- Always reads from the database.
- Always updates the cache with the database version.

Unit of Work Method beginTransactionEarly This method is advanced API. If you call `beginTransactionEarly` on an instance of a unit of work, all read operations should be performed through that instance of the unit of work.

This method starts a database transaction immediately: any objects you read will lock data in the database before commit time, reducing concurrency.

ConnectionPolicy Method `setShouldUseExclusiveConnection` Client sessions can access the data source using a connection pool or an exclusive connection. To use an exclusive connection, acquire your client session using a `ConnectionPolicy` (see "[Acquiring a Client Session That Uses Exclusive Connections](#)" on page 78-7).

If you are using isolated client sessions (see "[Isolated Client Sessions](#)" on page 75-20), you can use exclusive connections for reading isolated data. In this case, you can configure `TopLink` to acquire an exclusive connection from the write connection pool and use it for both writing and reading isolated data. However, `TopLink` still acquires a shared connection from the read connection pool for reading nonisolated data.

For more information, see "[Exclusive Write Connections](#)" on page 77-22.

Managing Cache Access

By default, `TopLink` uses the shared session cache as much as possible. Doing so increases concurrency and improves performance. However, to achieve a particular transaction isolation level, you may need to avoid the cache using some or all the following:

- [Isolated Client Session Cache](#)
- [ReadObjectQuery](#)
- [ReadAllQuery](#)
- [Descriptor Method `disableCacheHits`](#)
- [DatabaseQuery Method `dontMaintainCache`](#)

Isolated Client Session Cache This method always goes to the database for the initial read operation of an object whose descriptor is configured as isolated. By avoiding the shared session cache, you do not need to use the more complicated descriptor and query APIs to disable cache hits or always refresh. For more information about isolated client sessions, see "[Isolated Client Sessions](#)" on page 75-20. This is particularly useful for achieving serializable transaction isolation (see "[Serializable Read Levels](#)" on page 102-31).

ReadObjectQuery This API goes to the database unless it is a primary key-based query, in which case it will go to the cache first. For information on how to avoid the cache entirely in this case, see "[Descriptor Method `disableCacheHits`](#)" on page 102-30.

ReadAllQuery This API always goes to the databases. For information on how to avoid the cache entirely in this case, see the description of the `Descriptor` method `alwaysRefreshCache` in "[Cache Refresh API](#)" on page 90-14.

Descriptor Method `disableCacheHits` This API allows for cache hits on primary key, read-object queries to be disabled. This can be used with the `Descriptor` method `alwaysRefreshCache` to ensure queries always go to the database.

DatabaseQuery Method `dontMaintainCache` This is a query-level means of preventing objects from being added to the shared session cache. Using an isolated client session (see "[Isolated Client Session Cache](#)" on page 102-30) is a simpler approach to achieving the same ends.

CMP and External Transactions

In general, the transaction isolation information in this section applies to both CMP and non-CMP applications, with one exception:

When using a TopLink application with CMP, Oracle recommends that you configure your container to use separate read and write connection pools, and to associate only the write connections with an external transaction. This means the read connections do not participate in the transaction.

However, because TopLink treats EJB finders as just another type of query, you can use your descriptor configuration to exploit the options described in ["Reading Through the Write Connection"](#) on page 102-28. For example, if you configure a descriptor to use pessimistic locking (see ["Configuring Locking Policy"](#) on page 28-73), then when its finder is invoked it will allocate a write connection early and both read and write operations will use the same connection.

Refer to ["Externally Managed Transactional Data Sources"](#) on page 84-1 for more information on external transactions with transactional data sources.

Read Uncommitted Level

Oracle does not recommend using this transaction isolation level.

In general, a read uncommitted operation is not necessary. Using TopLink, a transaction isolation of read committed gives you better performance than read uncommitted but with greatly improved data integrity.

Read Committed Level

Using the unit of work guarantees that you will read only committed data in the shared session cache or committed data in the database.

Repeatable Read Levels

To achieve repeatable read operations, you must use a unit of work, you must register all objects in the unit of work (both objects you intend to modify and objects you intend only to read), and you must use `ObjectLevelReadQuery` method `conformResultsInUnitOfWork` or `Descriptor` method `alwaysConformResultsInUnitOfWork`.

By doing so, each time you query a registered object, you will get the version of the object as it currently is in your unit of work.

Serializable Read Levels

To achieve serializable transaction isolation with TopLink, Oracle recommends that you use an isolated client session (see ["Isolated Client Sessions"](#) on page 75-20) as follows:

1. Configure the database transaction isolation as serializable.
2. Configure objects as isolated (see ["Configuring Cache Isolation at the Project Level"](#) on page 22-19 or ["Configuring Cache Isolation at the Descriptor Level"](#) on page 28-44).
3. Use the `UnitOfWork` method `beginTransactionEarly` (see ["Unit of Work Method `beginTransactionEarly`"](#) on page 102-29).

If you are only concerned about the write aspect of serializable, optimistic locking is sufficient.

To prevent phantom read transactions (that is, when a transaction detects that new records that have been added to the database after the transaction started), use the `ReadQuery` method `cacheQueryResults`.

Troubleshooting a Unit of Work

This section examines common unit of work problems and debugging techniques, including the following:

- [Avoiding the Use of Post-Commit Clones](#)
- [Determining Whether or Not an Object Is the Cache Object](#)
- [Dumping the Contents of a Unit of Work](#)
- [Handling Exceptions](#)
- [Validating a Unit of Work](#)

Avoiding the Use of Post-Commit Clones

A common unit of work error is holding on to clones after commit time. Typically the clones are stored in a static variable and the developer incorrectly believes that this object is the cache copy. This leads to problems when another unit of work makes changes to the object and what the developer believes is the cache copy is not updated (because a unit of work updates only the cache copy, not old clones).

Consider the error in [Example 102–16](#). In this example you get a handle to the cache copy of a `Pet` and store it in the static `CACHE_PET`. We get a handle to a working copy clone and store it in the static `CLONE_PET`. In a future unit of work, the `Pet` is changed.

Developers who incorrectly store global references to clones from units of work often expect them to be updated when the cache object is changed in a future unit of work. Only the cache copy is updated.

Example 102–16 Incorrect Use of Handle to Clone

```
// Read a Pet from the database, store in static
CACHE_PET = (Pet)session.readObject(Pet.class);

// Put a clone in a static. This is a bad idea and is a common error
UnitOfWork uow = session.acquireUnitOfWork();
    CLONE_PET = (Pet)uow.readObject(Pet.class);
    CLONE_PET.setName("Hairy");
uow.commit();
//Later, the pet is changed again
UnitOfWork anotherUow = session.acquireUnitOfWork();
    Pet petClone = (Pet)anotherUow.registerObject(CACHE_PET);
    petClone.setName("Fuzzy");
anotherUow.commit();

// If you incorrectly stored the clone in a static and thought it should be
// updated when it is later changed, you would be wrong: only the cache copy is
// updated; NOT OLD CLONES
System.out.println("CACHE_PET is" + CACHE_PET);
System.out.println("CLONE_PET is" + CLONE_PET);
```

The two `System.out` calls produce the following output:

```
CACHE_PET isPet type Cat named Fuzzy id:100
```


CLONE_PET isPet type Cat named Hairy id:100

Determining Whether or Not an Object Is the Cache Object

In "[Modifying an Object](#)" on page 101-2, it was noted that it is possible to read any particular instance of a class by executing:

```
session.readObject(Class);
```

There is also a `readObject` method that takes an object as an argument: this method is equivalent to doing a `ReadObjectQuery` on the primary key of the object passed in. For example, the following code is equivalent to the code in the subsequent example:

```
session.readObject(pet);
```

The following is equivalent to the preceding code:

```
ReadObjectQuery query = new ReadObjectQuery();
query.setReferenceClass(Pet.class);
ExpressionBuilder builder = new ExpressionBuilder();
Expression exp = builder.get("id").equal(pet.getId());
query.setSelectionCriteria(exp);
session.executeQuery(query);
```

Also note that primary key-based queries, by default, will return what is in the cache without going to the database. As a result, you can use very quick and simple method for accessing the cache copy of an object as shown in [Example 102-17](#).

Example 102-17 Testing If an Object Is the Cache Object

```
//Here is a test to see if an object is the cache copy
boolean cached = CACHE_PET == session.readObject(CACHE_PET);
boolean cloned = CLONE_PET == session.readObject(CLONE_PET);
System.out.println("Is CACHE_PET the Cache copy of the object: " + cached);
System.out.println("Is CLONE_PET the Cache copy of the object: " + cloned);
```

This code produces the following output:

```
Is CACHE_PET the Cache copy of the object: true
Is CLONE_PET the Cache copy of the object: false
```

Dumping the Contents of a Unit of Work

The unit of work has several debugging methods to help you analyze performance or track down problems with your code. The most useful is `printRegisteredObjects`, which prints all the information about known objects in the unit of work. Use this method to see how many objects are registered and to make sure objects you are working on are registered.

To use this method, you must have log messages enabled for the session that the unit of work is from. Session log messages are disabled by default. To enable log messages, use the session `logMessages` method. To disable log messages, use the session `dontLogMessages` method as shown in [Example 102-18](#).

Example 102-18 Dumping the Contents of a Unit of Work

```
session.logMessages(); // Enable log messages
UnitOfWork uow = session.acquireUnitOfWork();
```



```
Pet petClone = (Pet)uow.readObject(Pet.class);
petClone.setName("Mop Top");

Pet pet2 = new Pet();
pet2.setId(200);
pet2.setName("Sparky");
pet2.setType("Dog");
uow.registerObject(pet2);

uow.printRegisteredObjects();
uow.commit();
session.dontLogMessages(); // Disable log messages
```

This example produces the following output:

UnitOfWork identity hashCode: 32373

Deleted Objects:

All Registered Clones:

```
Key: [100] Identity Hash Code:13901 Object: Pet type Cat named Mop Top id:100
Key: [200] Identity Hash Code:16010 Object: Pet type Dog named Sparky id:200
```

New Objects:

```
Key: [200] Identity Hash Code:16010 Object: Pet type Dog named Sparky id:200
```

Handling Exceptions

This section explains how to handle the following:

- [Exceptions at Commit Time](#)
- [Exceptions During Conforming](#)

Exceptions at Commit Time

TopLink exceptions are instances of `RuntimeException`, which means that methods that throw them do not have to be placed in a try-catch block.

However, the unit of work `commit` method is one that should be called within a try-catch block to deal with problems that may arise.

[Example 102–19](#) shows one way to handle unit of work exceptions:

Example 102–19 Handling Unit of Work Commit Exceptions

```
UnitOfWork uow = session.acquireUnitOfWork();
Pet petClone = (Pet)uow.registerObject(newPet);
petClone.setName("Assume this name is too long for a database constraint");
// Assume that the name argument violates a length constraint on the database.
// This will cause a DatabaseException on commit
try {
    uow.commit();
} catch (TopLinkException tle) {
    System.out.println("There was an exception: " + tle);
}
```

This code produces the following output:

```
There was an exception: EXCEPTION [ORACLEAS TOPLINK-6004]:
oracle.toplink.exceptions.DatabaseException
```

See [Database Exceptions \(4002 – 4018\)](#) section on page 13-27 in the [TopLink Exception Reference](#) chapter for more information on `DatabaseException`.

If you use optimistic locking, you must catch exceptions at commit time because the exception raised is the indication that there was an optimistic locking problem. Optimistic locking allows all users to access a given object, even if it is currently in use in a transaction or unit of work. When the unit of work attempts to change the object, the database checks to ensure that the object has not changed since it was initially read by the unit of work. If the object has changed, the database raises an exception, and the unit of work rolls back the transaction. For more information, see ["Locking and the Unit of Work"](#) on page 100-12.

If you are using an external transaction service, exceptions may be thrown long after your `UnitOfWork` code has returned. Using `UnitOfWork` method `writeChanges`, you can catch and handle most exceptions before the external transaction is committed. For more information, see ["External Transaction Commit Exceptions"](#) on page 102-24.

Exceptions During Conforming

You can conform query results in a unit of work across one-to-many relationships and a combination of both one-to-one and one-to-many relationships. [Example 102-20](#) illustrates a query across two levels of relationships, one-to-many and one-to-one.

Example 102-20 Querying Across Two Levels of Relationship

```
Expression exp =
    bldr.anyOf("managedEmployees").get("address").get("city").equal("Perth");
```

By default, any exceptions thrown during conforming are suppressed. However, you can use the `UnitOfWork` method `setShouldThrowConformExceptions` to make the unit of work throw all conforming exceptions. This method takes one `int` argument with the following values:

- 0—do not throw conform exceptions (default)
- 1—throw all conform exceptions

For more information on customizing exception handling when using conforming and in-memory queries, see ["Handling Exceptions Resulting From In-Memory Queries"](#) on page 96-34.

Validating a Unit of Work

The unit of work validates object references at commit time. If an object registered in a unit of work references other unregistered objects, this violates object transaction isolation, and causes TopLink validation to raise an exception.

Although referencing unregistered objects from a registered object can corrupt the session cache, there are applications in which you want to disable validation. TopLink offers the following APIs to toggle validation:

- `dontPerformValidation`: disables validation
- `performFullValidation`: enables validation

Validating the Unit of Work Before Commit Time

If the unit of work detects an error when merging changes into the session cache, it throws a `QueryException`. Although this exception specifies the invalid object and the reason it is invalid, it may still be difficult to determine the cause of the problem.

In this case, you can use the `validateObjectSpace` method to test registered objects and provide the full stack trace of all traversed objects. This may help you more easily find the problem. You can call this method at any time on a unit of work.

Glossary

This glossary contains terms and abbreviations that you should be familiar with when using Oracle TopLink.

attribute

A variable of a class or object. In TopLink, an *attribute* describes all instance variables of a class. Every attribute contains a single mapping. Attributes store primitive data such as integers, and simple Java types such as `String` or `Date`.

authentication

The means by which a data source validates a user's identity and determines whether or not the user has sufficient privileges to perform a given action.

bean class

The implementation of the bean. The bean is accessed from the client using the home and remote interfaces.

bean-managed persistence (BMP)

A scheme for persisting entity beans that requires the developer to manually code the methods that perform the persistence.

Compare to [container-managed persistence \(CMP\)](#).

branch class

Has a persistent superclass and also has subclasses. By default, queries performed on the branch class return instances of the branch class and any of its subclasses. However, the branch class can be configured so that queries on it return only instances of itself without instances of its subclasses.

Compare to [leaf class](#).

class

A category of objects. Classes allow data and method to be grouped together.

class indicator field

A field in the table of the root class that indicates which subclass should be instantiated

client session broker

A collection of client sessions, one from each server session associated with the session broker.

connection pool

A collection of reusable connections to a single data source.

container-managed persistence (CMP)

A scheme for persisting entity beans that uses information supplied by the developer or deployer to perform the persistence

Compare to [bean-managed persistence \(BMP\)](#).

custom SQL

Refers to any non-TopLink-generated SQL used through TopLink. This includes hard-coded SQL and stored procedure calls.

data definition language (DDL)

The data definition part of the structured query language (SQL). TopLink Workbench can generate DDL creation scripts that can be used to create tables on the desired database.

database session

A database session provides a client application with a single data store connection, for simple, standalone applications in which a single connection services all data store requests for one user.

default mapping

A relational persistence framework term that refers to making the framework automatically generate the object descriptor metadata (including such things as mappings, login data, database platform, locking, and foreign keys). Default mapping is available for TopLink projects using EJB 2.0 CMP applications with OC4J.

dependent class path (IBM WebSphere)

Location where nonbean classes are specified. TopLink requires that the bean classes be included here since they are referenced by the project.

deployment descriptor

A set of XML files that provide the additional required information to install an EJB within its server. Typically, this includes security, transaction, relationship, and persistence information.

Compare with TopLink [descriptors](#).

descriptors

An TopLink object that describes how an object's attributes and relationships are to be represented in relational database table(s). An "TopLink descriptor" is not the same as a [deployment descriptor](#), although it plays a similar role.

direct access

By default, TopLink accesses public attributes directly when writing the attributes of the object to the database or reading the attributes of the object from the database.

Compare to [method access](#).

direct mapping

There are two basic ways of storing object attributes directly in a table:

-
- The information can be stored directly if the attribute type is comparable to a database type.
 - If there is no database primitive type that is logically comparable to the attribute's type, it must be transformed on its way to and from the database

TopLink provides five classes of direct mappings.

Compare to [relationship mapping](#).

Enterprise Java Beans (EJB)

EJB are server-side domain objects that fit into a standard component-based architecture for building enterprise applications with Java. They are objects that become distributed, transactional, and secure components. TopLink Workbench uses three types of EJB: [session beans](#), [entity beans](#), and [message-driven beans](#).

expressions

The TopLink equivalent of an SQL conditional clause. TopLink expressions are specified using the `Expression` and `ExpressionBuilder` classes.

entity beans

EJB that represent a persistent data object. TopLink uses two schemes for persisting entity beans: [bean-managed persistence \(BMP\)](#) and [container-managed persistence \(CMP\)](#).

fetch group

A performance enhancement that defines a subset of object attributes to be loaded initially and ensures that all other attributes are loaded on demand.

hub

A common connection point for devices in a network.

identity map

Used to cache objects for performance and to maintain object identity.

See also [object identity](#).

independent relationship

A relationship in which the source and target are public objects that exist independently; the destruction of one object does not necessarily imply the destruction of the other.

Compare to [private relationship](#).

indirection

An indirection object is one that acts as a stand-in for another object. In TopLink, indirection is implemented through value holders, which delay database access through acting as substitute for any object relationships.

inheritance

Describes how a child class inherits the characteristics of its parent class. TopLink supports multiple approaches to database implementations that preserve the inheritance relationship.

in-memory query

A query that is run against the shared session cache.

instantiate

Create an instance of a Java class.

J2C

The J2EE Connector architecture (J2C) adapter is a way to persist Java objects to a nonrelational data source, such as XML.

J2SE

The Java 2 Platform, Standard Edition (J2SE) is the core Java technology platform. It provides software compilers, tools, runtimes, and APIs for writing, deploying, and running applets and applications in Java.

J2EE

The Java 2 Platform, Enterprise Edition (J2EE) is an environment for developing and deploying enterprise applications. J2EE includes a set of services, APIs, and protocols for developing multitiered web-based applications.

J2EE Containers

A J2EE container is a run-time environment for Enterprise Java Beans (EJB) that includes such basic functions as security, life cycle management, transaction management, and deployment services. J2EE containers are usually provided by a J2EE server, such as Oracle Containers for J2EE.

Java Messaging Service (JMS)

The JMS API is a protocol for communication that provides asynchronous communication between components in a distributed computing environment.

Java Naming and Directory Interface (JNDI)

The JDBC specification recommends using a JNDI naming service to acquire a connection to a database. TopLink supports acquiring a database connection in this fashion. To take advantage of this feature, construct and configure an instance of `oracle.toplink.jndi.JNDIConnector` and pass it to the project login object using the `setConnector` method.

Java Transaction API (JTA)

The Java Transaction API (JTA) specifies the interfaces between a transaction manager, a resource manager, an application server, and transactional applications involved in a distributed transaction system.

leaf class

Has a persistent superclass in the hierarchy but does not have subclasses; queries performed on the leaf class can return only instances of the leaf class.

Compare to [branch class](#).

locking policy

A mechanism that ensures one user does not overwrite another users's work. TopLink descriptors support optimistic and pessimistic locking policies.

mappings

Describe how individual Java objects and attributes relate to a data source.

message-driven beans

An EJB that processes asynchronous Java Messaging Service (JMS) messages. For TopLink clients, a message-driven bean is simply a JMS consumer with no conversational state and no home or remote interfaces.

method access

The application registers accessor methods for the attribute.

Compare to [direct access](#).

named query

A TopLink query that is created and stored, by name, in a session for later retrieval and execution

object identity

Ensures that each object is represented by one and only one instance in the application; that is, multiple retrievals of the same object return references to the same object instance, not multiple copies of the same object. Violating object identity can corrupt the object model.

See also [identity map](#).

optimistic locking

Also known as write locking; allows unlimited read access to objects. A client can write an object to the database only if the object has not changed since it was last read.

Compare to [pessimistic locking](#).

packet

A piece of a message transmitted over a packet-switching network. One of the key features of a packet is that it contains the destination address in addition to the data.

packet time-to-live

A number of hops that session data packets can take before expiring. The default is 2.

See also [packet](#).

persist

In object technology, the storage of an Java object by a data source.

pessimistic locking

Objects are locked before they are edited, which ensures that only one client is editing the object at any given time.

Compare to [optimistic locking](#).

primary key

A field (or combination of fields) that uniquely identifies a record in the data source.

private relationship

A relationship in which the target object is considered to be a private component of the source object; the target object cannot exist without the source and is accessible only through the source object; furthermore, if the source object is destroyed, the target object is destroyed as well.

Compare to [independent relationship](#).

query manager

An object, owned by a descriptor, that controls the way the descriptor accesses the database. The query manager generates its own default SQL to access the database in a transparent manner.

query optimization

TopLink supports two forms of query optimization: joining and batch reading. Their purpose is to optimize database access through reducing the number of database calls required to read a group of objects.

relationship

In TopLink, a reference between two TopLink-enabled objects.

relationship mapping

Persistent objects use relationship mappings to store references to instances of other persistent classes. The appropriate mapping class is chosen primarily by the cardinality of the relationship. TopLink provides five classes of relationship mappings.

Compare to [direct mapping](#).

Remote Method Invocation (RMI)

A set of protocols that enable Java objects to communicate remotely with other Java objects.

remote session

A remote session is a client-side session that communicates over RMI with a corresponding client session and server session on the server side. Remote sessions handle object identity and marshalling and unmarshalling between client side and server side.

service channel

A name of the TopLink coordinated cache channel to which sessions subscribe in order to participate in the same coordinated cache.

session beans

EJB that represent a business operation, task, or process. TopLink can use session beans to make the regular Java objects they access persistent, or to wrap other legacy applications.

stale data

An artifact of caching, in which an object in the cache is not the most recent version committed to the data source.

TopLink session broker

A mechanism that enables client applications to transparently access multiple databases through a single TopLink session.

unit of work

A transactional TopLink session that allows for a transaction to occur at the object level not only the database level. Changes to objects are not visible globally until the unit of work is committed.

value holder

A wrapping object used by TopLink to delay database access.

A

access

- data access, 84-1
- direct, 35-32
- modifiers, 4-42
- optimizing data access, 11-14
- remote sessions, 75-31

access method

- direct, 35-14
- generating, 4-30
- mappings, 35-15
- method, 35-14
- specifying, 35-32

access modifiers, classes, 4-42

acquiring

- client sessions, 78-6
- sessions, at runtime, 75-5
- unit of work, 101-1

activating descriptors, 4-10

Add Named Query dialog, 28-12, 28-18, 28-20

Add New Class dialog, 4-40, 4-53

Add New Table button, 4-22

Add or Refresh Class button, 4-50

addConstraingDependencies(), 102-16

address

- multicast group, 91-4
- multicast port, 91-5

Add/Update Existing Tables from Database

- button, 4-23

advanced properties for descriptors, 22-7

After Load tab, 28-79

aggregate collection relational mappings

- and EJB, 36-11
- configuring, 44-1
- understanding, 36-10

aggregate descriptors

- about, 26-5
- aggregate object mapping, 26-5
- EIS projects, 26-8
- EJB 3.0, 26-6
- inheritance, 26-17
- relational projects, 26-5
- XML projects, 26-8

aggregate object relational mapping

- aggregate descriptors, and, 26-5

aggregate object relational mappings

- configuring, 47-1
- understanding, 36-12

aggregation, isolated client sessions, 75-24

AllFieldsLockingPolicy, 26-20

allows none, 28-18, 97-14

allows null, 28-18, 97-14

amending descriptors, 2-20, 26-5, 28-78

- see also* after load

announcement delay, 91-11

Ant, integrating with Oracle TopLink

- Workbench, 4-53

any collection XML mappings

- configuring, 72-1
- understanding, 65-29

any object XML mappings

- configuring, 71-1
- understanding, 65-27

application development

- deploying, 10-1
- mapping, 2-20
- querying, 2-13, 74-2
- troubleshooting, 15-1

application layer, remote sessions, 75-30

application servers

- EJB support, customizing, 7-25
- integrating with Oracle TopLink, 7-1
- logging, 75-9
- optimization, 11-30
- setter parameter type checking, 7-25
- single-object finder return type checking, 7-25, 7-26
- software requirements, 7-2
- target platforms, 2-5
- unknown primary key class support, 7-25

architectures

- application, 1-4
- BMP, 1-5, 2-33
- cache, 90-1
- choosing, 2-3
- CMP, 1-5, 2-27
- EIS, 2-4, 56-5
- EJB entity beans, 1-5, 2-27, 2-33
- EJB session bean facade, 1-5, 2-24
- locking, 2-10
- optimistic locking, 2-10

- Oracle TopLink, 1-1
- pessimistic locking, 2-10
- selecting, 2-5
- session brokers, 75-26
- sessions, 75-2
- three-tier, 1-4, 2-21
- two-tier, 1-6, 2-23
- unit of work, 100-1
- web services, 1-5, 2-36
- arguments, binding in query, 29-6
- array
 - dimensionality, 4-45, 4-49
 - object-relational mappings, 49-2
- AsOfClause, 99-2
- asynchronous change propagation, 91-2
- AttributeChangeTrackingPolicy
 - about, 100-8
 - OC4J CMP integration, 100-8
 - OC4J EJB 1.x CMP integration, 100-8
 - OC4J EJB 2.x CMP integration, 100-8
 - OC4J EJB 3.0 CMP integration, 100-8
 - other application servers, 100-8
- attributes
 - adding to descriptors, 4-44
 - array dimensionality, 4-45, 4-49
 - changes, tracking, 28-72
 - final, 4-45, 4-48
 - in TopLink Workbench, Navigator window, 4-9
 - lazy loading. *see* fetch groups
 - static, 4-45, 4-48
 - transforming, 35-29, 35-31, 36-15
 - transient, 4-45, 4-48
 - unmapping, 34-6
 - volatile, 4-45, 4-48
- Attributes tab, 4-44, 4-48
- Attunity Connect platform, 84-3
- auditing
 - authentication, 84-6
 - unit of work, 102-20
- authentication
 - about, 84-5
 - auditing, 84-6
 - proxy authentication, 84-5
 - simple JDBC authentication, 84-5
 - three-tier architecture, 84-5
 - two-tier architecture, 84-5
- Automap, 34-2
- automapping descriptors
 - about, 34-2
 - see also* mappings
- automatic table generation
 - about, 33-4
 - configuring, 8-12

B

- Base64 encoded strings, 35-25
- batch options
 - mappings, 37-7
 - writing, 11-15

- batch reading
 - in query objects, 98-10
 - read optimization, and, 11-19
- batch writing
 - about, 11-15, 11-26, 86-11
 - dynamic, 11-15
 - dynamic, `setMaxBatchWritingSize()`, 11-15
 - MySQL4 platform, 86-9
 - non-parameterized, 11-15
 - parameterized, 11-15
 - `setMaxBatchWritingSize()`, 11-15
- BEA WebLogic
 - deploying to, 10-2
 - deployment exceptions, 15-3
 - modifying persistence descriptor, 8-13
 - setting classpath, 7-15
 - setting shared library, 7-15
 - transport layer, 75-31
 - using a security manager, 7-20
- @Bean fetch=lazy, 33-9
- beans
 - session beans, 2-25
 - stateful beans, 2-25
 - stateless beans, 2-25
- bidirectional relationships
 - about, 36-2
 - generating, 4-31
 - in one-to-one mappings, 35-34
 - target keys, 36-5
 - with indirection, 35-35
- `bindAllParameters()` method, 23-7
- bindings
 - arguments, 29-6
 - input paramters, 98-19
 - JAXB, 20-10
 - LOB mappings, 34-3
 - see* parameter binding
- BLOB
 - mapping to, 34-2
- BMP
 - and EJB 1.1, 2-34
 - and EJB 2.0, 2-34
 - and TopLink, 2-33
 - deployment files, 8-9
 - descriptors, 26-3, 28-44
 - packaging for deployment, 9-5
- boolean logic in expressions, 97-3
- branch classes, 26-12, 26-13
- buttons. *see* toolbars
- bye array binding, LOB mappings, 34-3
- Byte array Base64, 35-25

C

- cache
 - about, 2-12, 2-18, 90-1
 - architecture, 90-1
 - configuring, 90-6, 99-21
 - coordination, 90-9, 91-1, 92-1, 93-1
 - descriptor level, 28-35

- disabling during read query, 96-34
- distributed, 90-9
- expiration, 22-19, 28-40, 99-22
- expression limitations, 96-31
- identity maps, using, 75-29
- in-memory queries, 96-30, 96-31
- internal query object cache, 96-36, 99-21, 99-22
- invalidation, 22-19, 28-40, 90-7, 90-8
- isolated client sessions, 75-24
- isolation, 90-6, 90-9
- object cache, 96-36
- object cascading refresh, 96-35
- object refresh, 96-35
- optimizing, 11-13
- project level, 22-13
- queries, 90-6, 96-29
- query cache, 96-36
- `readObject` method, and, 98-2
- refreshing, 28-27, 90-8, 96-35
- restrictions, 96-36
- service channel, 91-3
- sessions, 75-3, 75-32, 90-2
- stale data, 90-6
- storing query results, 96-36, 99-21
- type and size, 22-13, 28-35
- unit of work cache, 90-2
- cache coordination
 - about, 90-9
 - application server clustering, and, 7-4
 - avoiding stale data, 90-7
 - CMP projects, 8-11
 - EJB Entity Beans with BMP architecture, and, 2-36
 - explicit query refreshes, 90-7
 - JMS, 92-1
 - `orion-ejb-jar.xml`, 8-11
 - packet time-to-live, 91-15
 - permissions, 7-23
 - RMI, 93-1
- cache invalidation, avoiding stale data, 90-7
- cache synchronization. *see* cache coordination
- `cacheAllStatements()`, 23-7
- `cacheQueryResults()`, 28-25
- cache-synchronization property, 8-11
- Caching tab, 28-35, 28-38, 28-39, 28-42
- call
 - call queries, 96-16
 - EIS, 98-24
 - EJBQLCall, 98-23
 - SQLCall, 98-18
 - StoredFunctionCall, 98-22
 - StoredProcedureCall, 98-20
- Call object, queries, 96-3
- cascading
 - object refresh, 96-35
 - optimistic version locking, 26-18
 - write queries, compared to non-cascading, 96-14, 98-13
- catalog, database, 4-22
- `catchExcpetions()`, 77-19
- change policy
 - about, 28-70
 - attribute change tracking, configuring, 28-72
 - deferred change detection, configuring, 28-71
 - empty transaction, 100-6
 - object change tracking, configuring, 28-71
 - unit of work, 100-6
- change tracking
 - attribute, configuring, 28-72
 - deferred, configuring, 28-71
 - object, configuring, 28-71
- changed items, displaying in TopLink Workbench Navigator window, 4-9
- `ChangedFieldsLockingPolicy`, 26-20
- changing package names, 4-52
- `checkDatabase()`, 77-18
- checking in/out projects, 5-3
- `checkInstantiationPolicy()`, 77-19
- Choose a Schema Context dialog box, 31-3, 32-2
- Choose Query Key dialog box, 97-14
- Choose Relationships to Generate dialog box, 4-30, 4-31
- Choose Root Element dialog box, 31-4, 32-5
- class extraction method
 - about, 26-14
 - inheritance, 26-15
- class indicator
 - about, 26-13
 - class extraction method, 26-14
 - class indicator field, 26-13, 41-1
- class loader
 - host application, 2-5
 - loading session, 78-4
- class modifiers, 4-42
- Class tab, 4-41, 4-42, 4-43
- classes
 - access modifiers, 4-42
 - adding and refreshing, 4-50
 - branch, 26-13
 - creating, 4-40, 4-52
 - `CursoredStream`, optimizing, 99-18
 - Database Exception, 98-24
 - `DatabaseMapping`, 33-25
 - default null values, 35-36
 - `DeleteObjectQuery`, 98-13
 - `ExpressionBuilder`, 97-15
 - generating from database, 4-30
 - `InsertObjectQuery`, 98-13
 - `InsertObjectQuery`, 100-12
 - interfaces, 4-43
 - leaf, 26-13
 - merging files, 5-5
 - methods, adding, 4-47
 - naming, 4-41
 - non-descriptor classes, 4-51
 - object model, 2-16
 - Performance Profiler, 11-3
 - persistent requirements, 2-11
 - preferences, 4-16
 - refreshing, 4-50

- removing, 4-51
- root, 26-12
- troubleshooting, 14-22
- tutorial, 16-2
- unit of work, 100-12
- UpdateObjectQuery, 98-13, 98-14
- ValueHolderInterface, 2-11, 33-7, 36-9
- VariableOneToOneMapping, 36-6
- see also* specific class name
- classpath
 - adding, 22-3
 - BEA WebLogic, 7-15
 - configuring, 4-2, 7-15, 7-21
 - connector.jar, 4-2
 - custom Collection class, 4-3
 - DRIVER_CLASSPATH, Oracle TopLink Workbench, 4-2
 - IBM WebSphere, 7-21
 - J2C adapter, 4-2
 - JDBC driver, 4-2, 4-23
 - JDBC_CLASSPATH, 4-2
 - Oracle TopLink Workbench DRIVER_CLASSPATH, 4-2
 - relative, 22-3
 - setting for BEA WebLogic, 7-15
 - troubleshooting, 14-22
 - xdb.jar file, 4-3
- class-table relationships, 16-6
- client sessions
 - about, 75-1, 75-13, 75-25
 - acquiring, 78-6
 - configuration, 82-1
 - example, 75-15
 - shared resources, 75-15
- client-controlled transactions, 102-24
- client-server architecture. *See* two-tier
- CLOB
 - mapping to, 34-2
- clones
 - copying methods, 28-69
 - merging changes, 102-12
 - post-commit, avoiding, 102-30
 - unit of work, 100-2, 100-9
- Cloudscape platform, 84-3
- clustering, integrating TopLink with, 7-4
- CMP
 - and EJB 1.1, 2-27
 - and EJB 2.x, 2-27
 - and EJB 3.0, 2-27
 - and TopLink, 2-27
 - CMPPolicy, 26-3, 28-44
 - deploying, 8-9, 10-2
 - descriptors, inheritance, 26-4
 - external transactions, 100-3
 - isolated client sessions, 75-24
 - local transaction, 102-25
 - non-deferred write, 102-25
 - OC4J persistence, 7-5
 - packaging for deployment, 9-4
 - setter parameter type checking, 7-25
 - single-object finder return type checking, 7-26
 - transaction attribute, 102-24
 - unit of work, 100-3, 102-24
 - unknown primary key class support, 7-25
- code generation, optimizing, 11-8
- collapsing items in Navigator window, 4-9
- collection class
 - specifying, 98-9
 - specifying in query objects, 98-9
- collections
 - persistent requirements for mappings, 2-11
 - query results, 96-8
- comments
 - descriptors, 28-8, 28-9
 - mappings, 35-18
 - projects, 22-21
- commit
 - and Java Transaction API, 100-10
 - failure, resuming after, 102-14
 - resuming unit of work after, 102-14
- Communication Exceptions, 13-64
- composite collection EIS mappings
 - configuring, 61-1
 - example, 56-8
 - understanding, 56-7
- composite collection XML mappings
 - configuration, 70-1
 - configuring, 70-1
 - understanding, 65-25
- composite descriptors
 - about, 26-5
 - composite object mapping, 26-8
 - EIS projects, 26-8
 - XML projects, 26-8
- composite direct collection EIS mappings
 - configuring, 59-1
 - understanding, 56-6
- composite direct collection XML mappings
 - configuring, 68-1
 - understanding, 65-14
- composite EIS descriptors, 27-5
- composite object EIS mappings
 - composite descriptors, and, 26-8
 - configuring, 60-1
 - understanding, 56-7
- composite object mappings
 - composite descriptors, and, 26-8
- composite object XML mappings
 - configuration, 69-1
 - configuring, 69-1
 - understanding, 65-21
- composite primary key, 36-9
- concrete class. *see* container policy
- concurrency
 - about, 2-17
 - exceptions, 13-25
 - server session, 75-18
- Concurrency Exceptions, 13-25
- configurations
 - about, 76-2

- creating, 76-2
- development environment, 5-1
- new, 76-2
- Oracle JDeveloper, 5-1
- session, 76-2, 76-4, 76-6, 76-9
- conforming
 - about, 102-8
 - alternatives, UnitOfWork method
 - writeChanges, 102-11
 - alternatives, UnitOfWork properties, 102-12
 - descriptors, and, 102-11
 - in-memory queries, 102-9
 - queries, alternatives to, 102-11
- ConnectBy, 99-7
- connection policy
 - configuring, 77-19
 - exclusive connections, 77-19
 - lazy connection acquisition, 77-20
- connection pool
 - about, 84-7
 - connection count, 89-1
 - ConnectionPolicy, 75-20
 - external, 84-8
 - internal, 84-7, 84-8, 84-9
 - lazy connection allocation, 75-18, 78-9
 - named, 84-9
 - parameter binding, 11-16
 - prepared statement caching, 11-16
 - read, 84-8
 - sequence, 20-5, 84-8
 - server session, 75-18
 - sessions, and, 75-3
 - size, 89-1
 - write, 84-8
- Connection Specifications tab, 24-3
- Connection tab, 24-3
- connections
 - about, 84-6
 - connection pool, 84-7
 - exclusive write connection, 77-19
 - lazy acquisition, 77-20
 - reading through the write connection, 102-27
- connector.jar, 4-2, 20-7
- container configuration file, 8-5
- container policy
 - about, 35-26
 - custom Collection class, 4-3
 - sorting, in memory, 35-26
- container-controlled transactions, 102-24
- context
 - JAXB path, 20-14
 - menus, 4-5
 - schema, 31-2, 32-2
- Context.SECURITY_CREDENTIALS, 91-9
- Context.SECURITY_PRINCIPAL, 91-9
- Conversion Exception, 13-26
- Converter tab
 - object type mappings, 35-22
- converters
 - custom, 36-3

- object type, 35-22
- coordinated announcement delay, 91-11
- coordinated cache, 8-11
 - configuring, 91-1
 - naming service, 91-7
 - service channel, 91-3
- copy policy
 - about, 28-69
 - method, 28-69
 - setting, 28-70
- copying project objects, 5-6
- Copying tab, 28-70
- CORBA
 - Oracle TopLink transport layer support, 75-31
 - Transaction Service *see* OTS
- Create New Project button, 21-2
- Create New Project dialog box, 21-2
- Create New Session dialog, 76-4, 76-6, 76-8
- Create Project from JAXB dialog, 21-6
- Create Project from OC4J dialog, 7-9
- creating
 - configurations, 76-2
 - expressions, 97-12
 - sessions, 76-4, 76-6, 76-8, 88-1
- Crimson XML parser, 7-3
- cursored streams
 - example, 99-18
 - optimizing, 99-18
 - remote sessions, 75-32
 - usage example, 75-32
- cursors
 - as query results, 96-8
 - traversing scrollable, 99-16
- Custom Calls tab, 31-6
- custom SQL
 - Custom SQL tab, 29-7
 - unit of work, 102-15
- customization
 - about, 12-1
 - data types, 12-1
 - EIS, 12-1
 - mapping extensions, 12-1
 - overview, 2-15
 - XML, 12-1

D

- data access
 - about, 84-1
 - authentication, 84-5
 - connection pool, 84-7
 - connections, 84-6
 - optimizing, 11-14, 11-15
 - platforms, 84-3
- data level queries
 - example, 97-12
 - in expressions, 97-11
- data source platform
 - about, 84-3
 - Attunity Connect database, 84-3

- Cloudscape database, 84-3
- databases, 84-3
- DB2 database, 84-3
- EIS, 84-4
- HSQL database, 84-3
- Informix database, 84-3
- J2C adapter, 84-4
- JDBC drivers, 84-3
- Microsoft Access database, 84-3
- MySQL4 database, 84-3
- Oracle database, 84-3
- Oracle8 database, 84-3
- Oracle9 database, 84-3
- PointBase database, 84-3
- SQLAnywhere database, 84-3
- SQLServer database, 84-3
- SybasePlatform database, 84-3
- data sources
 - configuring, 85-1
 - nontransactional, 84-1
 - transactional, 84-1
 - troubleshooting, 14-23
- Database Exceptions, 13-27, 98-24
- database fields, configuring, 37-2
- database functions, in expressions, 97-3
- database login
 - parameter binding, 86-9
 - prepared statement caching, 86-9
- Database Preferences, 4-17
- database queries
 - about, 96-10, 98-4
 - fetch groups, 96-12
 - join reading, 96-12
 - object level modify query, 96-13, 96-14
 - object level read query, 96-11, 96-13
 - partial object query, 96-11
 - read all query, 96-11
 - read object query, 96-11
 - report query, 96-15
- database schema
 - tables, 4-22
 - tutorial, 16-4
- database sessions
 - about, 75-2, 75-28
 - cache, 75-32
 - configuration, 83-1
 - creating, 76-8
- database tables
 - about, 4-21
 - adding to database, 4-22
 - creating, 4-22
 - descriptors and classes, generating, 4-30
 - EJB entity generation, 4-31
 - fields, 4-25
 - generating, 4-29, 4-30, 4-32
 - importing, 4-22
 - Java source generation, 21-15
 - JDBC driver classpath, 4-23
 - properties, 4-25
 - references, 4-27, 4-28
 - removing, 4-24
 - renaming, 4-24
 - schema, 4-22
 - SQL generation, 4-29
 - TopLink Workbench, Navigator window, 4-9
- DatabaseException class, 98-24
- DatabaseLogin, 84-2
- DatabaseMapping class, 33-25
- DatabaseQuery, 96-3
- DatabaseRow, 36-15
- databases
 - catalog, 4-22
 - common problems, 14-23
 - connect to, 4-21
 - creating reference tables on, 4-27
 - custom drivers, 4-18
 - disconnect from, 4-21
 - drivers, 4-18
 - exceptions, 98-24
 - fields, configuring, 37-2
 - for project, 21-3
 - Java type conversion, 6-3
 - linking, 75-28
 - log out of, 4-22
 - logging into, 4-22, 23-7
 - logins, 86-1
 - mapping. *See* mappings
 - platform, 21-3, 23-2, 24-2, 86-1, 87-1
 - preferences, 4-17
 - schema, 4-22
 - schema manager, 6-1
 - tables, 4-21
 - TopLink Workbench, Navigator window, 4-9
 - troubleshooting, 14-23
 - type conversion, schema manager, 6-3
 - using with Oracle TopLink Workbench, 4-21
- DatabaseSession class
 - logging SQL and messages, 75-10
- DB2
 - platform, 84-3
 - schema manager type conversion, 6-3
- DBase platform, 84-3
- default mapping
 - about, 33-4
 - automatic table generation, and, 33-4
 - configuring, 8-12
 - default table generator, 6-6
- Default Mapping Exception, 13-75
- default table generator
 - default mapping, 6-6
 - table creator, creating, 6-4
- defaults
 - login level null values, 85-5
 - mapping level null values, 35-12
 - null values, 35-12, 35-36, 85-5
 - optimization, 11-8
 - root, 32-5
 - see also* preferences
- DefaultSequence, 86-6
- deferred change detection

- configuring, 28-71
- DeferredChangeDetectionPolicy, 100-7
- Delete All Interaction tab, 63-3
- deleteObject(), 29-8
- DeleteObjectQuery, 98-13
- deletes
 - controlling order, 102-16
 - delete operation, 98-4
 - queries, EIS mappings, 63-3
- demarcation of unit of work transactions, 100-2
- dependent objects
 - non-deferred write, 26-4
- deploy tool
 - about, 10-5
 - troubleshooting, 10-6
 - using with WebSphere Studio Application Developer, 10-5
- deploying
 - about, 2-14, 10-1
 - application server requirements, 7-1
 - BEA WebLogic, 10-2
 - CMP applications, 10-2, 10-4
 - database login, 23-7
 - entity beans overview, 2-14
 - generating XML for, 8-3
 - hot deployment, 10-4
 - IBM WebSphere, 10-3
 - Java applications, 10-1
 - JSP and Servlet applications, 10-1
 - modifying BEA WebLogic persistence
 - descriptor, 8-13
 - non-CMP applications, 10-4
 - packaging, 9-1
 - Session Bean applications, 10-1
 - troubleshooting, 15-1
- deployment descriptors, 26-3
- Deployment Exceptions, 13-67
- deployment exceptions
 - BEA WebLogic deployment, 15-3
 - Deployment Exceptions (14001-14033), 13-67
 - IBM WebSphere deployment, 15-12
- deployment files
 - BMP applications, 8-9
 - CMP applications, 8-9
 - creating, 8-1
 - descriptors, 26-3
 - EJB 3.0, 8-2, 9-1
 - JARs, troubleshooting, 15-1
 - Java applications, 8-8
 - JSP and Servlet applications, 8-8
 - Session Bean applications, 8-8
 - XML, generating, 8-3
- deployment XML, exporting, 21-14
- DeploymentXMLGenerator, 8-3
- Descriptor Event Listener, 28-60
- Descriptor Event Manager
 - about, 26-8
 - Descriptor Event Listener, 28-60
 - domain object methods, 28-57
 - event types, 28-57

- handlers, 26-8, 28-57, 28-60
- handlers, Descriptor Event Listener, 28-60
- understanding, 26-8
- descriptor events
 - about, 26-8
 - Descriptor Event Listener, 28-60
 - domain object methods, 28-57
 - handlers, 26-8, 28-57, 28-60
 - types of, 28-57
 - understanding, 26-8
- Descriptor Exceptions, 13-2, 13-29
- Descriptor Info tab, 28-3, 28-5, 28-7, 29-2, 29-4, 31-2, 31-4, 32-2, 32-4, 32-5, 32-6
- DescriptorEventListener, 28-60
- descriptors
 - about, 19-2, 26-1
 - advanced properties, default, 22-7
 - aggregate, 26-5, 27-2
 - aggregate, EJB 3.0 and, 26-6
 - aggregate, relational projects and, 26-5
 - amending, 2-20, 26-5, 28-78
 - API, 26-22
 - architecture, 26-2
 - attributes, adding, 4-44
 - automapping, 34-2
 - automatically mapping, 34-2
 - BMP, 26-3, 28-44
 - cache refreshing, 28-27
 - change policy, 28-70
 - child inheritance, 28-49
 - class, 27-2
 - CMP, 26-3, 28-44
 - CMPPolicy, 26-3, 28-44
 - comments, 28-8, 28-9
 - composite, 26-5, 26-8, 27-5
 - composite EIS, 27-5
 - configuring, 28-1
 - conforming, 102-8
 - creating, 27-1, 27-2, 27-5
 - custom EIS interactions for basic persistence, 31-6
 - custom SQL queries for basic persistence, 29-6
 - deactivating, 4-10
 - default mappings, 33-4
 - default root, 31-4
 - deployment information, 26-3
 - Descriptor Event Listener, 28-60
 - domain object methods, 28-57
 - EIS, 26-11, 27-5, 31-1
 - EIS projects, 26-8
 - EJB, 26-3, 28-44
 - EJB information, 26-3, 28-44
 - errors, 4-10, 4-11, 14-4
 - event handlers, 26-8, 28-57, 28-60
 - events, 26-8, 28-57
 - existence checking, 11-13, 28-42
 - fetch groups, 28-77
 - files, merging, 5-5
 - generating from database, 4-30
 - hierarchy, inheritance, 26-23
 - history policy, 28-73

- identity maps, 22-14, 28-35, 28-37, 28-39
- in Java, 26-22
- inactive, 4-10
- inheritance, 26-3, 26-12, 28-49, 28-50
- instantiation, 11-13
- interface, 27-2, 28-31, 28-33
- mapping, 28-3, 29-2, 33-4, 34-1, 34-2
- merging, 5-5
- named queries, 28-10
- nondesoriptor classes, 4-51
- object-relational, 26-11, 30-1
- optimizing, 11-13
- parent inheritance, 28-50
- projects, 19-2, 26-1
- query key interfaces, 28-31
- query timeout, 28-24, 28-26
- read only, 28-4
- registering with sessions, 75-12, 77-2, 77-3, 77-10
- relational, 26-11, 27-2, 29-1
- removing, 4-51
- returning policy, 28-65
- root EIS, 27-5
- root element, 32-5
- schema context, 31-2, 32-2, 32-3
- sequencing, 26-9, 29-3
- TopLink Workbench, Navigator window, 4-9
- types of, 26-1
- validating, 27-6
- XML, 26-12, 32-1
- XML projects, 26-8
- detachement indirection, 33-9
- developing applications with Oracle TopLink, 2-1
- development environments
 - about, 3-2
 - configuring, 5-1
 - database logins, 23-7
- development process
 - about, 2-1
 - additional support, 2-3
 - stages of, 2-1
 - with Oracle TopLink, 2-1
- development tools
 - about, 3-1
 - profiler, 11-2
 - schema manager, 6-1
- dimensionality, array, 4-45, 4-49
- direct access
 - about, 22-4, 35-14
 - specifying, 35-32
- direct collection relational mappings
 - configuring, 45-1
 - example, 36-11
 - understanding, 36-11
- direct collections
 - session broker limitations, 75-27
- direct EIS mappings
 - configuring, 58-1
 - understanding, 56-5
- direct field
 - in direct collection mappings, 45-2
- direct key fields, 46-2
- direct map relational mappings
 - configuring, 46-1
 - direct keys, 46-2
 - direct value, 46-1
 - understanding, 36-12
- direct mappings
 - generating deprecated, 22-12
 - with EJB, 36-4
- direct value fields, 46-1
- direct XML mappings
 - configuring, 67-1
 - understanding, 65-5
- directionality in mappings, 36-2
- direct-to-field mappings
 - ObjectTypemapping deprecated, 36-3
 - SerializedObjectMapping deprecated, 36-3
 - type conversions, 37-3
 - TypeConversionMapping deprecated, 36-3
- direct-to-field relational mappings
 - configuring, 38-1
 - options, 38-1
 - timestamp support, 37-3
 - understanding, 36-4
- direct-to-XMLType relational mappings
 - configuring, 39-1
 - understanding, 36-4
- Discovery Exception, 13-77
- DMS profiler
 - about, 11-4, 75-11
 - accessing with JMX, 11-7
 - and JMX
 - nouns, 11-4, 75-11
 - selecting, 77-11, 77-14, 77-17, 77-20
 - sensors, 11-4, 75-11
- document information in XML schemas, 4-34, 4-36, 4-38
- documentation
 - hosted, 4-14
 - See also* Help
- does exist write object, 11-27
- dontOptimizeDataConversion(), 11-15
- doPrivileged(), 7-4
- DRIVER_CLASSPATH
 - Oracle TopLink Workbench environment, 4-2
- drivers, custom database, 4-18
- dynamic batch writing
 - about, 11-15
 - setMaxBatchWritingSize(), 11-15
- dynamic fetch groups, querying with, 99-4

E

- Editor window, about, 4-4, 4-10
- EIS
 - about, 20-8
 - architecture, 2-4
 - call, 98-24
 - custom interactions for basic persistence, per descriptor, 31-6

- indexed records, configuring, 31-5
- interactions, 31-6, 96-6, 98-24
- mapped records, configuring, 31-5
- mappings, 20-8, 56-2
- projects, 21-3
- queries, 96-6
- record format, configuring, 31-5
- XML records, configuring, 31-5
- EIS descriptors
 - composite, 27-5
 - configuring, 31-1
 - default root, 31-4
 - locking policy, 28-62
 - root descriptor, 27-5
 - schema context, 31-2
 - setDataTypeName, 31-5
 - understanding, 26-11
- EIS mappings
 - about, 56-1, 56-2
 - architecture, 56-5
 - composite collection, 56-7, 61-1
 - composite direct collection, 56-6, 59-1
 - composite object, 56-7, 60-1
 - configuring, 57-1
 - direct, 56-5, 58-1
 - jaxb:class support, 56-3
 - list support, 56-3
 - one-to-many, 56-12, 63-1
 - one-to-many, key on source, 56-13
 - one-to-many, key on target, 56-15
 - one-to-one, 56-8, 62-1
 - one-to-one, key on source, 56-9
 - one-to-one, key on target, 56-10
 - transformation, 56-17, 64-1
 - types of, 56-1
 - union support, 56-3
 - xsd:list, 56-3
 - xsd:union, 56-3
- EIS projects
 - configuring, 24-1
 - connector.jar, 20-7
 - indexed records, 20-9
 - mapped records, 20-9
 - sequencing, 20-5
 - understanding, 20-7
 - XML records, 20-9
- EIS queries, 96-6
- EIS record types, supported, 56-2
- EISLogin, 84-2
- EJB
 - descriptors, 21-10
 - isolated client sessions, 75-24
 - setter parameter type checking, 7-25
 - single-object finder return type checking, 7-25
 - unknown primary key class support, 7-25
- EJB 1.1
 - and BMP, 2-34
 - and CMP, 2-27
- EJB 1.x
 - indirection, 33-9
 - serialization, 33-9
- EJB 2.x
 - and BMP, 2-34
 - and CMP, 2-27
 - default mapping, 33-4
 - indirection, 33-9
 - serialization, 33-9
- EJB 3.0
 - <J2EE-Container>-jar.xml file, 8-6
 - and CMP, 2-27
 - attribute change tracking policy, OC4J CMP integration, 7-5, 100-8
 - Bean annotation, fetch=lazy, 33-9
 - default mapping, 33-4
 - deployment files, 8-2, 9-1
 - deployment files, OC4J CMP integration, 7-5
 - detachment, 33-9
 - Embedded annotation, 26-6
 - packaging for deployment, 9-1
 - packaging, OC4J CMP integration, 7-5
 - projects.xml file, 8-3, 8-5
 - serialization, 33-9
 - sessions.xml file, 8-5
 - toplink-ejb.xml file, 8-8
 - value holder indirection, 33-9
- EJB descriptors, opening projects with, 21-10
- EJB entities
 - BMP indirection, 33-9
 - CMP hot deployment, 10-4
 - deployment overview, 2-14
 - EJB 1.x indirection, 33-9
 - EJB 2.x indirection, 33-9
 - EJB 3.0 indirection, 33-9
 - generating, 4-31
 - hot deployment, 10-4
 - indirection, 33-9
 - inheritance, 26-4, 26-17
 - inserting after ejbCreate, 26-4
 - inserting after ejbPostCreate, 26-4
 - mapping, 22-6
 - non-CMP hot deployment, 10-4
 - non-deferred write, understanding, 26-3, 102-25
 - sequencing, 20-21
- EJB entity beans
 - and EJB 1.1, 2-27, 2-34
 - and EJB 2.x, 2-27, 2-34
 - and EJB 3.0, 2-27
 - with BMP architecture, 1-5, 2-33
 - with CMP architecture, 1-5, 2-27
- EJB Exception Factory Exceptions, 13-58
- EJB finders
 - about, 96-23
 - Call finders, 96-26
 - creating, 99-8
 - DatabaseQuery finders, about, 96-26
 - default finders, about, 96-26
 - default finders, creating, 99-8
 - EJB QL finders, about, 96-27
 - ejb-jar.xml options, 99-9
 - ejbSelect method, 96-28

- ejbSelect, creating, 99-15
 - ejbSelect, using, 99-15
 - expression finders, about, 96-27
 - named query finders, about, 96-27
 - predefined, about, 96-24
 - primary key finders, about, 96-27
 - redirect finders, about, 96-28
 - redirect finders, using, 99-12
 - single-object finder return type checking, 7-25
 - SQL finders, about, 96-28
- EJB Info tab, 28-44
- EJB JAR XML Exception, 13-84
- EJB Preferences, 4-16
- EJB QL
 - exceptions, 13-55
 - queries, 29-6, 96-5
 - query language, 96-5
- EJB session bean facade architecture
 - about, 1-5, 2-24
 - understanding, 2-24
- EJB Session Beans, 75-31
- ejbc
 - about, 10-2
 - troubleshooting, 10-3
- ejbCreate, 26-4
- ejb-jar.xml file
 - about, 8-5, 21-15
 - configuring, 8-5
 - corresponding to Oracle TopLink Workbench functions, 21-15
 - EJB finder options, 99-9
 - location, 22-7
 - managing, 5-6
 - preferences, 4-16
 - synchronization under EJB 2.0, 8-5
 - updating from, 21-16
 - writing, 21-16
- ejbPostCreate, 26-4
- @Embeddable, 26-6
- @Embedded, 26-6
- empty unit of work transactions, 100-6
- encrypting login passwords, 21-13
- enhanced validation exceptions, 8-4
- Enterprise Information Systems. *see* EIS
- entity beans
 - deployment, 2-14
 - descriptor information, 26-3
 - direct mappings, 36-4
 - indirection, BMP, 33-9
 - indirection, EJB 1.x, 33-9
 - indirection, EJB 2.x, 33-9
 - indirection, EJB 3.0, 33-9
 - sequencing with, 20-21
- environment
 - configuring, 4-2
 - JAVA_HOME, 4-2
 - JDBC_CLASSPATH, 4-2
 - proxy, 4-13, 4-14, 21-16
- error codes
 - 1-176, 13-2
 - 1-99, 14-1
 - 100-199, 14-2
 - 200-399, 14-4
 - 400-599, 14-11
 - 500-699, 14-15
 - 700-799, 14-20
 - 800-899, 14-21
 - 10001-10047, 13-58
 - 12000-12004, 13-64
 - 13000-13020, 13-64
 - 14001-14027, 13-67
 - 15001-15024, 13-70
 - 17001-17006, 13-71
 - 18001-18002, 13-74
 - 19001-19003, 13-75
 - 22001-22004, 13-77
 - 22101-22105, 13-77
 - 3001-3007, 13-26
 - 4002-4018, 13-27
 - 5001-5008, 13-29
 - 6001-6098, 13-30
 - 7001-7104, 13-42
 - 72000-72023, 13-84
 - 8001-8010, 13-55
 - 9000-9009, 13-56
- errors
 - about, 13-1
 - codes and descriptions, 13-1, 14-1
 - descriptors, 4-10, 4-11
 - migration, 7-13
 - Oracle TopLink Workbench, 14-1
- Event Manager, 75-5
- events
 - about, 26-8
 - client session, 75-6
 - database access, 75-6
 - Descriptor Event Listener, 28-60
 - domain object methods, 28-57
 - handlers, 26-8, 28-57, 28-60
 - listeners, sessions, 75-7
 - session, 75-5
 - session manager, 75-6
 - sever session, 75-6
 - types of, 28-57
 - unit of work, 75-6
- examples
 - composite collection EIS mapping, 56-8
 - context menu, 4-5
 - cursor streams, 99-17
 - direct collection mappings, 36-11
 - direct-to-field mappings, 36-4
 - exception handler, 77-13
 - indirection, 33-6
 - inheritance, 26-12
 - Oracle TopLink Workbench, 4-3
 - performance optimization, 11-22, 11-24
 - proxy indirection in code, 35-9
 - READALL finders, 99-10
 - report query, 98-6
 - scrollable cursors, 99-17

- serialized mapping, 33-10
- stored procedure call, 98-21, 98-22
- transformation mapping, 36-15
- transformation XML mapping, 56-18, 65-31
- Unit of Work, 100-6, 101-7
- write, write all, 98-3
- exception handler
 - about, 75-12
 - example, 77-13
 - selecting, 77-12
- exceptions
 - chained, 75-10
 - communication exceptions, 13-64
 - conversion exceptions, 13-26
 - database exceptions, 13-27, 98-24
 - Default mapping exception, 13-75
 - deployment exceptions, 13-67
 - descriptor exceptions, 13-2
 - discovery exceptions, 13-77
 - EJB exceptions factory, 13-58
 - EJB JAR XML exceptions, 13-84
 - EJB QL exceptions, 13-55
 - enhanced validation, 8-4
 - `java.security.AccessControlException`, 15-12, 15-13
 - JMS processing exceptions, 13-74
 - Migration utility exception, 13-82
 - optimistic locking, 13-29
 - query exceptions, 13-30
 - remote command manager exceptions, 13-77
 - SDK data store exceptions, 13-71
 - SDK descriptor exceptions, 13-75
 - selecting exception handler, 77-12
 - session loader exceptions, 13-56
 - synchronization exceptions, 13-70
 - Transaction exception, 13-79
 - validation exceptions, 13-42
 - XML conversion exception, 13-80
 - XML data store exceptions, 13-64
- exclusive connections
 - about, 102-28
 - internal read connection pool, 89-6
 - isolated sessions, 75-20, 77-19
 - named queries, 28-24
- existence checking, 102-5
 - descriptors, 28-42
 - projects, 22-8
- expanding items in Navigator window, 4-9
- expiration of objects in the cache, 22-19, 28-40
- explicit query refreshes, cache coordination, 90-7
- exporting
 - deployment XML, 21-14
 - Java model, 21-14
 - Java source, 21-14
 - preferences, 4-12
 - projects, 21-13
- Expression Builder, 97-12, 97-15
- Expression Builder dialog box, 97-12
- Expression class, 97-1
- ExpressionMath class, 97-1

- expressions
 - about, 96-3, 97-1
 - allows none, 28-18, 97-14
 - allows null, 28-18, 97-14
 - building, 97-12
 - comparing with SQL, 97-1
 - components, 97-2
 - creating, 97-12
 - data level queries, 97-11
 - database functions, 97-3
 - in relationships, 97-5
 - in-memory queries, limitations, 96-31
 - mathematical functions, 97-4
 - multiple, 97-10
 - one-to-one mappings, 97-5
 - outer joins, 98-11
 - parallel expressions, 97-11
 - parameterized, 97-7
 - platform functions, 97-16
 - query keys, 97-9
 - subqueries and subselects, 97-10
 - user-defined functions, 97-4, 97-16
 - using Boolean logic, 97-3
 - XML Type functions, 97-4
 - see also* queries
- external
 - applications, 102-26
 - connection pools, 84-8
 - controller, transaction, 100-1
 - JDBC pools, 2-35
 - transactions, 100-1
- external transaction controller
 - configuration, sessions, 77-14
 - session, 100-2

F

- factory name, JMS connection, 92-2
- failure, resuming unit of work after commit, 102-14
- features of Oracle TopLink, 1-4
- fetch groups
 - about, 26-4, 28-77
 - configuring, 28-77, 99-3
 - default, 28-77, 99-3
 - disabling, 99-3
 - dynamic, 99-4
 - object level read query, 96-12
 - read optimization, and, 28-77
 - size, 11-17
 - static, 99-3
- field references, 37-8
- Field uses XML Schema "type" attribute
 - option, 33-12
- fields in database tables, 4-25
- Fields tab, 47-2
- field-to-object attribute transformation, 35-29, 35-31
- files
 - JAXB-specific, 20-10
 - TopLink-specific, 20-11
 - see also* specific file name

- final attributes, 4-45, 4-48
- findAll, using, 96-26
- finders
 - caching options, 96-37
 - disabling cache, 96-37
 - managing large result sets, 99-19
 - refreshing results, 96-38
 - see also* EJB finders
- flashback queries
 - about, 96-18
 - historical client sessions, 81-1
- forceUpdateToVersionField(), 102-17
- foreign keys
 - about, 2-17, 37-8
 - configuring in EIS mappings, 62-1
 - EIS mappings, 63-1
 - multiple tables, 29-14
 - one-to-many mappings, 36-7
 - one-to-one mappings, 35-34, 36-5
 - parameterized expressions, 97-7
 - references, 14-12, 14-21
 - target, 36-6, 37-9
 - troubleshooting, 14-12, 14-21
- full identity map, 90-3

G

- garbage collection, managing, 101-8
- General Preferences dialog, 4-13
- Generate Classes and Descriptors dialog, 4-30
- Generate EJB Entity Classes and Descriptors dialog, 4-31
- generating
 - access method, 4-30
 - deployment JARs, troubleshooting, 15-1
 - deprecated direct mappings, 22-12
 - see also* exporting
- getCatalogs(), 4-22
- getField(), 97-11
- getImportedKeys(), 4-22
- getParameter, 97-8
- getPrimaryKeys(), 4-22
- getTable(), 97-12
- getTables(), 4-22
- getTableTypes(), 4-22
- getValue(), 33-7
- getValue() method, 33-7

H

- hard cache weak identity map
 - about, 90-4
 - when to use, 90-5
- help
 - about, 4-12
 - displaying, 4-12
- Help button., 4-12
- Help Preferences, 4-14, 4-16
- hierarchical queries
 - about, 96-18

- described, 99-6
- hints, Oracle Hints in queries, 96-18
- historical client sessions
 - about, 75-1, 75-24
 - cache, 75-33
 - limitations of, 75-25
- historical queries, 99-2
 - about, 96-20
 - see also* AsOfClause
- history policy, configuring, 28-73
- holders, value, 33-6
- host URL, JMS topic, 92-3
- hosted
 - documentation, 4-14
 - XSD files, 8-2, 8-4
- hot deployment
 - about, 10-4
 - CMP applications, 10-4
 - non-CMP applications, 10-4
- HSQL platform, 84-3

I

- IBM Informix Database native sequencing, 20-19
- IBM WebSphere
 - deploy tool, 10-5
 - deploying to, 10-3
 - deployment exceptions, 15-12
 - setting classpath, 7-21
- identity
 - about, 2-16, 90-3
 - cache, and, 90-3
 - using cache to preserve, 90-3
 - see also* identity map
- identity map cache
 - disabling during a write query, 98-14
 - refresh in read query, 96-35
- identity maps
 - about, 28-35, 28-37, 28-39, 75-29
 - cascading refresh during read query, 96-35
 - descriptors, 22-14, 28-35
 - example, 96-35
 - full, 90-3
 - guidelines for choosing type, 90-4
 - hard cache weak identity map, 90-4, 90-5
 - isolated client sessions, 75-24
 - no identity map, 90-4
 - refreshing during read query, 96-35
 - soft cache weak identity map, 90-4, 90-5
 - soft cache weak identity map and read optimization, 11-19
 - specifying, 28-35, 28-37, 28-39
 - weak, 90-3
 - weak identity map and read optimization, 11-19
- Identity tab. *see* Caching tab
- impedance mismatch, solving, 1-2
- Implementors tab, 28-33
- Import Tables from Database dialog, 4-23
- importing
 - classes, 4-16

- preferences, 4-12
- inactive descriptors
 - about, 4-10
 - mapping to, 37-6, 57-3
- independent relationships, 35-16
- indexed records, 56-2
- indirection
 - about, 2-18, 33-5
 - bidirectional relationships, 35-35
 - choosing the correct type, 35-4
 - configuring, 35-3
 - EJB, 33-9
 - EJB 1.x CMP, 33-9
 - EJB 2.x CMP, 33-9
 - EJB 3.0 CMP, 33-9
 - example, 33-6
 - many-to-many mappings, 36-9
 - nontransparent, 2-11
 - one-to-many mappings, 35-34
 - proxy indirection, 33-8
 - remote sessions, 75-32
 - serialization, 33-9
 - transparent, 2-11, 33-8
 - value holder, 33-6
 - ValueHolderInterface, 2-11
 - see also* proxy indirection, transparent indirection
- Informix platform, 84-3
- inheritance
 - about, 2-17, 26-3, 26-12
 - aggregate classes, 26-17
 - aggregate collection mappings, 36-10
 - branch classes, 26-12, 26-13
 - child descriptors, 28-49
 - class extraction, 26-14, 26-15
 - class indicator, 26-13, 26-14
 - descriptors, 26-3, 26-12, 28-49, 28-50
 - finding subclasses, 26-13
 - instantiating subclasses, 26-13
 - isolated client sessions, 75-24
 - leaf classes, 26-12, 26-13, 99-4
 - primary keys, 26-15
 - queries, 96-22
 - querying on hierarchy, 99-4
 - relational parent, 28-50
 - root class, 26-12, 28-49, 28-51
 - root class subclasses, finding in inheritance, 26-13
 - supporting with multiple tables, 26-16
 - supporting with one table, 26-15
 - transformed to relational model, 11-11
 - using with EJB, 26-4, 26-17
- inheritance hierarchies
 - descriptors, 26-23
 - querying on, 99-4
- Inheritance tab, 28-49, 28-51
- inherited subclasses, mapping, 28-56
- in-memory query
 - about, 90-6
 - check cache using exact primary key, 96-30
 - check cache using primary key, 96-30
 - check database if not in cache, 96-30
 - conform results in unit of work, 96-31
 - conforming, 102-9
 - expression limitations, 96-31
 - supported, 96-30
 - using, 96-30
- inner join, 97-5
- insert operation, 98-3, 98-4
- insertObject(), 29-7
- instantiation policy
 - about, 28-68
 - setting, 28-68
- Instantiation tab, 28-68
- integrity checker, 27-6
 - about, 75-12
 - configuring, 77-18
- interactions
 - about, 98-24
 - creating, 31-6
- interface alias
 - about, 29-10
 - creating, 29-11
- Interface Alias tab, 29-11
- interfaces
 - classes, implementing, 4-43
 - customizing, 4-13
 - descriptors, 27-2, 28-31, 28-33
 - queries, 96-22
 - query keys, 28-31
 - querying on, 99-4
- internal connection pool
 - about, 84-7
 - named, 84-9
 - read, 84-8
 - sequence, 84-8
 - write, 84-8
- internal query object cache
 - about, 96-36
 - configuring, 99-21, 99-22
 - expiration, 99-22
 - restrictions, 96-36
- internal transactions, 100-1
- invalidation of objects in the cache, 22-19, 28-40, 90-8
- IP address for multicast group, 91-4
- isolated client sessions
 - about, 75-1, 75-19, 80-1
 - configuration, 80-1
 - life cycle, 75-22
 - limitations of, 75-23
 - session event handlers, 75-21
 - with Oracle Virtual Private Database (VPD), 75-20
- isolated session
 - cache, 75-33
 - ConnectionPolicy, 75-20
 - exclusive connections, 75-20
 - supported databases, 75-20
- isolation
 - cache, 90-6
 - transaction levels, 100-2
 - unit of work transactions, 100-4

Iterator interface, 99-16

J

J2C adapters

- about, 84-4
- configuring for Oracle TopLink Workbench, 4-2
- EISLogin, 84-2
- selecting, 84-4
- with EIS, 21-3

<J2EE-Container>-ejb-jar.xml file, 8-5

<J2EE-Container>-jar.xml file

EJB 3.0, 8-6

J2EE

- parameter binding, 11-16
- prepared statement caching, 11-16
- web applications, 1-4

Java

- database tables, 21-15
- exporting to, 21-14
- integration with any datasource, 1-2
- iterators, 99-16
- object model, 2-11

Java applications

- deploying, 10-1
- deployment files, 8-8
- packaging for deployment, 9-1

Java Cryptography Extension, 21-13

Java Management Extensions. *see* JMX

Java Naming and Directory Interface. *See* JNDI

Java Object Builder, 75-4

Java streams

- described, 99-17
- optimizing, 99-18
- support for, 99-17

Java Transaction API

- and unit of work commit, 100-10
- and unit of work rollback, 100-11
- see also* JTA

Java Transaction Service *see* JTA

JAVA_HOME, 4-2

java.security.AccessControlException, 15-12, 15-13

java.util.Collection interface, 35-26

java.util.Map interface, 35-26

java.util.Set interface, 35-26

javax.ejb.EntityBean interface, 4-31

JAXB

- creating projects from, 21-6
- files, 20-10
- generating project from the command line, 21-8
- jaxb:class, and EIS mappings, 56-3
- jaxb:class, and XML mappings, 65-4
- proxy configuration, 21-8
- tljxb.cmd, 21-8
- tljxb.sh, 21-8
- typesafe enumeration converter, 35-25
- understanding, 20-10
- validation, 20-14
- XML projects, 20-10

JAXB typesafe enumeration converter

- configuring, 35-25
- understanding, 33-24

JAXBContext, 20-14

JCE. *see* Java Cryptography Extension

JConnect, 11-14

JDBC

- adaptor for EIS, 56-5
- database gateway for EIS, 56-5
- driver classpath, 4-23
- JConnect, 11-14
- Sybase JConnect, 11-14

JDBC drivers

- about, 84-3
- configuring for Oracle TopLink Workbench, 4-2
- fetch size, 11-14, 11-17
- general properties, 11-14
- mapping LOBs, 34-3
- selecting, 84-3

JDBC pools

- external with EJB Entity Beans with BMP architecture, 2-35

JDBC_CLASSPATH, 4-2

JDeveloper. *See* Oracle JDeveloper

JMS

- connection factory name, 92-2
- coordinated cache, 92-1
- Processing Exceptions, 13-72, 13-74
- topic host URL, 92-3
- topic name, 92-1

JMX

- about
- and DMS profiler
- DMS profiler, 11-7

JNDI naming service, 91-7

joining

- about, 96-12
- expressions, and, 97-5
- mappings, and, 40-1
- one-to-many, about, 98-11
- one-to-one mappings, 40-1, 98-11
- optimizing reads, 11-19
- queries, and, 98-11
- QueryManager expressions, 99-4
- read queries, 11-19, 98-11

JSP and Servlet applications

- deploying, 10-1
- deployment files, 8-8
- packaging for deployment, 9-2

JTA

- about, 100-3
- and unit of work, 100-3
- isolated client sessions, 75-24
- unit of work, 100-1

JTA/JTS

- using with EJB Entity Beans with BMP architecture, 2-36

just-in-time reading. *see* indirection

K

- Key Converter tab, 46-3
- key pairs
 - database table reference, 4-28
 - troubleshooting, 14-12, 14-21
- keys
 - about, 2-16
 - foreign, 2-17, 35-34, 36-5
 - foreign, target, 36-6
 - inheritance, 26-15
 - multiple tables, 29-13
 - primary, 26-15, 29-13, 35-2, 36-6, 36-9, 41-3
 - read-only settings, 35-2
 - reference key field, 36-11, 36-12
 - variable class relationships, 36-6, 41-3

L

- large result sets, managing in finders, 99-19
- lazy attribute loading and read optimization, 28-77
- lazy connection
 - acquisition, 77-20
 - allocation, 75-18, 78-9
- lazy loading. *see* indirection
- lazy reading. *see* indirection
- leaf classes, 26-12, 26-13, 99-4
- life cycle of unit of work, 100-5
- LOB
 - mapping to, 34-2
- local
 - documentation, 4-14
 - transactions, 102-25
- locked files, 5-6
- locking policy
 - AllFieldsLockingPolicy, 26-20
 - ChangedFieldsLockingPolicy, 26-20
 - configuring, 28-62
 - field locking, 26-20
 - optimistic, 26-9, 26-18, 26-20
 - optimistic version locking, 26-17
 - OptimisticLockException, 26-18, 26-21
 - pessimistic locking policy, 26-9, 26-21
 - SelectedFieldsLockingPolicy, 26-21
 - stale data, and, 90-6
 - three-tier architectures, optimistic locking and, 26-21
 - three-tier architectures, pessimistic locking and, 26-22
 - TimestampLockingPolicy, 26-18
 - understanding, 26-9, 26-17
 - version locking, 26-18
 - VersionLockingPolicy, 26-18
- Locking tab, 28-63
- log into database, 4-22
- Log Out of Database, 4-22
- log out of database, 4-22
- logging
 - application server, 75-9
 - chained exceptions, 75-10
 - java.util.logging, 75-9

- log level, 75-10
- Oracle Enterprise Manager 10g, 75-10
- output, 75-10
- permissions, 7-24
- sessions, 75-7, 77-4, 77-5
- TopLink native logging, 75-8
- types, 75-8
- Logging tab, 77-5
- login
 - CMP deployment, 20-3
 - database, 23-5, 23-7, 86-1
 - deployment, 20-3
 - development, 20-4
 - platforms, and, 20-4
 - projects, and, 20-3, 74-1, 84-2
 - role in project, 20-3, 84-2
 - session, 20-3, 77-4
 - session role, non-CMP, 20-3
- logMessages method, 75-7
- look and feel, specifying, 4-13

M

- Manage Non-Descriptor Classes dialog, 4-51
- management, source control, 5-3
- manager, session events, 75-5
- many-to-many mappings
 - relation table, 36-9
 - relation tables, 43-1
 - session broker limitations, 75-27
- many-to-many relational mappings
 - configuring, 43-1
 - EJB, 36-9
 - understanding, 36-8
- mapped records, 56-3
- mapping extensions
 - custom data types, 12-1
 - JAXB typesafe enumeration converter, 33-24, 35-25
 - object type converter, 33-12, 35-22
 - serialized object converter, 33-10, 35-18
 - simple type translator, 33-12, 35-23
 - transformation mappings, 33-14
 - type conversion converter, 33-11, 35-20
- mappings
 - about, 2-15, 2-16, 19-1, 19-2, 33-1, 34-1
 - access types, 35-15
 - aggregate collection mappings and EJB, 36-11
 - anyType mapping, 66-3
 - as part of the application development process, 2-20
 - automatic, 34-2
 - batch options, 37-7
 - class hierarchy, 33-25
 - comments, 35-18
 - configuring, 34-1, 35-1
 - database field, 37-4
 - default mapping, 33-4
 - deprecated, generating, 22-12
 - direct access, 11-13, 35-14

- directionality, 36-2
- EIS mappings, 20-8, 33-27, 56-1, 57-1
- EJB 2.0 entities, 22-6
- errors, 14-11
- example, 33-3
- extensions, about, 33-10
- hierarchy, 33-25
- inactive descriptors, 37-6, 57-3
- indirection, 11-13, 33-5, 35-3
- isolated client sessions, 75-23
- manually configuring, 34-1
- many-to-many, 36-9
- many-to-many, with EJB, 36-9
- method access, 35-14, 35-32
- null values, 35-12, 35-36
- object-relational, 49-1, 50-1
- one-to-many object, with EJB, 36-8
- one-to-one with EJB, 36-6
- optimizing, 11-13
- OX mappings, 33-27
- projects, and, 19-2
- read only, 35-2, 35-3
- relation tables, 43-1, 43-2
- relational, 36-1, 37-1
- removing, 34-6
- to tables, 28-3, 29-2
- TopLink Workbench, Navigator window, 4-9
- types of, 33-1
- XML mappings, 65-1, 66-1
- mathematical functions, in TopLink
 - expressions, 97-4
- menu bar, 4-5
- menus
 - about, 4-4, 4-5
 - context menus, 4-5
 - menu bar, 4-5
- merging
 - changes in clones, 102-12
 - Oracle TopLink Workbench project files, 5-4
 - project files, 5-4
- messages, error, 13-1, 14-1
- metadata
 - about, 2-12, 2-19
 - advantages, 2-19
 - creating, 2-20, 2-21
 - mapping and configuration, 19-1
 - project metadata, 2-20
 - session metadata, 2-21
- Metalink, 2-3
- method access
 - about, 22-4, 35-14
 - setting, 35-32
- methods
 - adding, 4-47
 - getValue(), 33-7
 - setValue(), 33-7
 - wrapper policy, 28-76
 - see also* specific method name
- Microsoft Access
 - platform, 84-3

- schema manager type conversion, 6-3
- Microsoft SQL Database native sequencing, 20-19
- migrating
 - error messages, 7-13
 - OC4J persistence to TopLink, 7-5
 - Oracle TopLink Workbench projects, 21-10
 - troubleshooting, 7-13
- Migration Utility Exception, 13-82
- model source, exporting, 21-14
- modifiers, class, 4-42
- multicast group address, coordinated cache, 91-4
- multicast port, coordinated cache, 91-5
- multiple sessions, 75-28, 78-2
- multiple tables
 - about, 29-13
 - specifying for descriptors, 29-13
- multiplicity in relationships, resolving, 7-14
- multi-processing, 11-29
- Multitable Info tab, 28-48, 29-13
- mutable mappings, 35-33
- .mwp file, 4-1, 21-2
- MySQL4
 - batch writing, 86-9
 - platform, 84-3
 - primary key restrictions, 20-15
 - schema manager type conversion, 6-3

N

- named connection pools, 84-9
- named queries
 - about, 28-10, 77-21, 96-15
 - configuring, 28-10, 77-21
 - descriptor level, 28-10
 - exclusive connections, 28-24
 - options, advanced, 28-24
 - parameter binding, 28-22
 - prepared statement caching, 28-22
 - redirect query, 96-16
 - session level, 77-21
 - using, 98-17
 - when not to use, 96-16
 - when to use, 96-16
- namespaces
 - about, 20-5, 20-22
 - configuring, 4-38
- naming service
 - coordinated cache, 91-7
 - JNDI, 91-7
 - RMI, 91-9
- native sequencing
 - IBM Informix Database, 20-19
 - Microsoft SQL Database, 20-19
 - Microsoft SQL Server, 6-6
 - non-Oracle database, 20-19
 - Oracle Database, 20-18, 20-20
 - Oracle Database SEQUENCE object, 20-19
 - Sybase Database, 6-6, 20-19
- Navigator window
 - about, 4-4, 4-9

- attribute and mapping, 4-9
- database, 4-9
- database tables in, 4-21
- descriptor, 4-9
- example, 4-9
- package, 4-9
- project, 4-9
- unsaved or changed item, 4-9
- NCHAR, 33-12
- NCLOB, 33-12
- neediness warnings. *See* troubleshooting
- nested table object-relational mappings
 - configuring, 55-1
 - understanding, 49-3
- nested unit of work, 100-9, 102-15
- new projects, 21-2
- New Reference dialog box, 4-27
- New Session button, 76-4, 76-6, 76-8
- New Sessions Configuration, 76-2
- New Table dialog box, 4-22
- newInstance method, 102-2
- no identity map, 90-4
- non-cascading write queries
 - compared to cascading, 96-14, 98-13
 - creating using dontCascadeParts ()
 - method, 96-14, 98-13
- non-deferred write
 - configuring, 102-11
 - dependent objects, 26-4
 - understanding, 26-3, 102-7, 102-25
- nonintrusive persistence, 2-18
- nonpersistent projects, 20-2
- nonrelational projects, 20-2
- nontransactional data sources, 84-1
- nontransparent indirection, 2-11
- nouns
 - DMS profiler, 11-4, 75-11
- null values
 - default, 35-12, 35-36, 85-5
 - in expressions, 97-13
 - login level, 85-5
 - mapping level, 35-12
- NVARCHAR2, 33-12

O

- object array object-relational mappings
 - configuring, 54-1
 - understanding, 49-2
- object cache, 96-36
- object cache, sessions, 75-3
- object change tracking
 - configuring, 28-71
- object identity, 75-29
 - about, 2-16, 90-3
 - cache, and, 90-3
 - using cache to preserve, 90-3
 - see also* identity map
- object indirection
 - read optimization, as, 11-19

- object level modify query
 - about, 96-13, 96-14
- object level read query
 - about, 96-11, 96-13
 - fetch groups, 96-12
 - join reading, 96-12
 - partial object query, 96-11
 - read all query, 96-11
 - read object query, 96-11
- object model
 - about, 2-16
 - generating with t ljaxb.cmd, 21-9
 - optimization, 11-8
 - Oracle TopLink requirements, 2-11
 - tutorial, 16-2
- object type converter
 - about, 12-1, 33-12
 - configuring, 35-22
- object type mappings
 - configuring, 35-22
- ObjectLevelChangeTrackingPolicy, 100-7
- object-relational descriptors
 - configuring, 30-1
 - locking policy, 28-62
 - understanding, 26-11
- object-relational mappings
 - about, 49-1
 - array, understanding, 49-2
 - configuring, 50-1
 - nested table, 49-3, 55-1
 - object array, 49-2, 54-1
 - overview, 2-15
 - reference, 49-2, 52-1, 53-1
 - structure, 49-2, 51-1
- object-relational projects
 - about, 20-6
 - sequencing, 20-5
- objects
 - cascading refresh in cache, 96-35
 - creating and registering, 102-2
 - query, 98-6
 - refreshing in cache, 96-35
 - registering and unregistering, 102-1
- ObjectTypingMapping
 - see* ObjectTypeConverter
- OC4J. *See* Oracle Containers for J2EE
- one-to-many EIS mappings
 - configuring, 63-1
 - key on source, 56-13
 - key on target, 56-15
 - understanding, 56-12
- one-to-many relational mappings
 - configuring, 42-1
 - understanding, 36-7
- one-to-one EIS mappings
 - configuring, 62-1
 - key on source, 56-9
 - key on target, 56-10
 - understanding, 56-8
- one-to-one relational mappings

- configuring, 40-1
 - expressions, 97-5
 - joining, 40-1
 - understanding, 36-5
 - with EJB, 36-6
- online help, 4-14
- Open Project button, 21-10
- opening projects, 21-10
- operators
 - boolean logic, 97-3
- optimistic locking
 - about, 26-9
 - application architecture, 2-10
 - cascading locking policy, 26-18, 28-65
 - database exception, 98-24
 - exceptions, 13-29
 - field locking policy, about, 26-20
 - rollbacks, 26-20
 - version locking policy, 26-17, 26-18, 28-65
 - with `forceUpdateToVersionField()`
 - method, 102-17
- optimistic locking policy
 - field locking, about, 26-20
 - version locking, 26-17, 26-18, 28-65
- `OptimisticLockException`, 26-18, 26-21
- optimization
 - about, 11-1
 - application bottlenecks, 11-2
 - application server, 11-30
 - batch reading, 11-17
 - batch writing, 11-15
 - CMP partial object queries, 11-17
 - code generation, 11-8
 - data access, 11-14, 11-15
 - data format, 11-15
 - database, 11-30
 - descriptors, 11-13
 - DMS profiler, 11-4, 75-11
 - existence checking, 11-13
 - fetch groups, 11-17
 - fetch size, JDBC, 11-17
 - general, 11-8
 - inheritance, 26-16
 - instantiation, 11-13
 - JDBC driver, 11-14, 11-17
 - join reading, 11-17
 - mappings, 11-13
 - named queries, 11-17
 - object model, 11-8
 - overview, 2-15
 - parameter binding, 11-15
 - partial object queries, 11-17
 - prepared statement caching, 11-15
 - profiler, 11-4, 75-11
 - optimization
 - TopLink Profiler, 11-2
 - queries, 11-16
 - reading, 11-19
 - `ReadQuery` method `setMaxRows`, 11-18
 - schema, 11-8
 - `setMaxRows`, 11-18
 - understanding, 11-1
 - unit of work, 11-30
 - writing, 11-26
- Oracle
 - development support, 2-3
 - remote session support, 75-31
- Oracle Containers for J2EE
 - creating projects from, 7-9
 - migrating to TopLink, 7-5
- Oracle Database
 - date and timestamp mappings, 37-3
 - native sequencing, 20-20
 - Oracle Database
 - SEQUENCE object, 20-19
 - platform, 84-3
 - schema manager type conversion, 6-3
- Oracle extensions
 - hierarchical queries, 99-6
 - Oracle Hints, 99-6
- Oracle Hints, using with TopLink queries, 99-6
- Oracle JDeveloper
 - configuring with Oracle TopLink, 5-1
 - TopLink sessions, 5-3
- Oracle TopLink
 - about, 1-1, 4-1
 - application architectures, 1-4
 - architectures, 1-1
 - deploy tool for IBM WebSphere, 10-5
 - development, 2-1, 3-1
 - features, 1-4
 - integrating with application server, 7-1, 7-2
 - mapping types, 33-1
 - optimization, 11-1
 - packaging your application, 9-1
 - public source, 12-2
 - runtime components, 3-2
 - understanding, 1-1
- Oracle TopLink Sessions Editor. *see* sessions
- Oracle TopLink Workbench
 - about, 4-1
 - Ant integration, 4-53
 - classpath, 4-2
 - creating projects, 21-1
 - development process, 4-1
 - `DRIVER_CLASSPATH`, 4-2
 - environment, 4-2
 - error messages, 14-1, 14-2
 - `JDBC_CLASSPATH`, 4-2
 - parts of, 4-3
 - preferences, 4-12
 - project, 4-1, 21-2
 - proxy, 4-13, 4-14, 21-16
 - sample, 4-3
 - table creator, creating, 6-4
 - upgrading projects, 21-10
- Oracle Virtual Private Database (VPD)
 - isolated client sessions, 75-20
 - proxy authentication, 75-21, 84-6, 86-13
- `oracle.sql.TimeStamp`, 37-3

- order
 - query keys, 37-8
 - relational mappings, 37-8
- OrderSibling, 99-7
- orion-ejb-jar.xml file
 - about, 8-9
 - entity-deployment attribute pm-name, 8-9
 - modifying for Oracle TopLink, 8-9
 - persistence-manager attribute
 - class-name, 8-10
 - persistence-manager attribute
 - descriptor, 8-10
 - persistence-manager attribute name, 8-9
 - persistence-manager subentry
 - forpm-properties
 - cache-synchronizations, 8-11
 - customization-class, 8-11
 - db-platform-class, 8-11
 - default-mapping, 8-11
 - project-class, 8-10
 - remote-relationships, 8-11
 - session-name, 8-10
- OTN (Oracle Technology Network), 1-4, 2-3
- OTS (Object Transaction Service)
 - about, 100-3
 - unit of work, 100-3
- outer joins
 - in expressions, 98-11
 - inheritance, 28-47
- output parameter event in stored procedures, 98-22
- OX mappings
 - about, 33-27
 - extensions, simple type translator, 33-13, 33-14
 - read conversions, 33-13
 - write conversions, 33-14

P

- package names
 - generating, 4-30
 - renaming, 4-52
 - TopLink Workbench, Navigator window, 4-9
 - see also* classes
- packaging for deployment
 - about, 9-1
 - BMP applications, 9-5
 - CMP applications, 9-4
 - EJB 3.0, 9-1
 - Java applications, 9-1
 - JSP and Servlet applications, 9-2
 - Session Bean applications, 9-3
- packet time-to-live cache coordination, 91-15
- parallel
 - expressions, 97-11
 - unit of work, 100-9
- parameter binding
 - about, 11-15
 - byte arrays, 11-15
 - configuring, 23-7, 28-22, 86-9, 98-17
 - database login level, 86-9

- descriptor level, 28-22
- external connection pools, 11-16
- internal connection pools, 11-16
- J2EE, 11-16
- named queries, 28-22
- optimizing, 11-15
- project level, 23-7
- queries, 98-17
- streams, 11-15
- strings, 11-15
- trouble shooting, 11-15
- parameterized batch writing
 - about, 11-15
 - setMaxBatchWritingSize(), 11-15
- parameterized expressions
 - about, 97-7
 - example, 97-8
- parameterized SQL
 - enabling on queries, 98-17
 - Oracle TopLink optimization features, 11-26
 - See also* parameter binding
- parser conflicts, XML, 7-3
- partial object reading optimization, 11-19
- passwords, encryption, 21-13
- performance optimization
 - about, 11-1
 - application bottleneck, 11-2
 - examples, 11-20
 - JConnect method isClosed, 11-14
 - using Performance Profiler, 11-2
- Performance Profiler
 - about, 11-2
 - class, 11-3
- persistence
 - about, 2-18
 - BEA WebLogic deployment, 8-13
 - by reachability, 101-6
 - components of, 2-12
 - descriptor, 8-13
 - implementation options, 2-11
 - manager, 7-4
 - OC4J, 7-5
 - projects, 20-2
 - types, 22-5
 - using a persistence layer, 2-13
- persistence manager
 - default, 7-4
 - migration, 7-4
 - restrictions, 7-4
- persistent classes
 - requirements, 2-11
 - types, 22-7
- pessimistic locking
 - about, 26-9, 26-21
 - application architecture, 2-10
 - policy, 26-21
- phantom reads, preventing, 102-30
- platforms
 - Attunity Connect database, 84-3
 - Cloudscape database, 84-3

- data source, 84-3
- database, 21-3, 23-2, 24-2, 84-3, 86-1, 87-1
- DB2 database, 84-3
- EIS, 84-4
- functions in expressions, 97-16
- HSQL database, 84-3
- Informix database, 84-3
- J2C adapter, 84-4
- JDBC drivers, 84-3
- Microsoft Access database, 84-3
- MySQL4 database, 84-3
- Oracle database, 84-3
- Oracle8 database, 84-3
- Oracle9 database, 84-3
- parser, XML, 7-2
- PointBase database, 84-3
- projects, and, 20-4
- server, 77-15
- session configuration, 77-15
- SQLAnywhere database, 84-3
- SQLServer database, 84-3
- SybasePlatform database, 84-3
- XML parser, 7-2
- see also* target platforms
- PointBase platform, 84-3
- pop-up menus. *see* context menus
- ports
 - multicast group, 91-5
 - permissions, 7-23, 7-24
- post-commit clones, avoiding, 102-30
- Potential EJB Descriptors dialog box, 21-10
- pre-allocating sequence numbers, 20-20, 23-4, 86-5, 87-2
- preferences
 - class import, 4-16
 - database, 4-17
 - EJB, 4-16
 - general, 4-13
 - help, 4-14
 - importing and exporting, 4-13
 - Oracle TopLink Workbench, 4-12
 - sessions, 4-18, 4-19
- Preferences - Class dialog, 4-16
- Preferences - EJB dialog, 4-16
- Preferences - General dialog, 4-13
- Preferences - Help dialog, 4-14
- Preferences - Mappings dialog, 4-15
- Preferences dialog, 4-13
- prepared statement caching
 - about, 11-15
 - configuring, 23-7, 28-22, 86-9, 98-17
 - database login level, 86-9
 - descriptor level, 28-22
 - external connection pools, 11-16
 - internal connection pools, 11-16
 - J2EE, 11-16
 - named queries, 28-22
 - optimizing, 11-15
 - project level, 23-7
 - queries, 98-17
 - query level, 98-17
- preserving XML documents, 32-6
- primary key
 - about, 2-16
 - composite, 36-9
 - inheritance, 26-15
 - multiple tables, 29-13
 - primkey in ejb-jar.xml file, 21-15
 - read-only settings, 35-2
 - restrictions, 20-15
 - setting, 4-26, 28-3, 29-2
 - unit of work, 100-11
 - unknown, 28-45
 - variable class relationships, 36-6, 41-3
- private relationships, 35-16
- Problems window
 - about, 4-4, 4-11
 - sample, 4-11
 - see also* error messages
- profiler
 - about, 77-10
 - development tool, 11-2
 - DMS, 11-4, 75-11
 - Oracle TopLink, 11-2, 11-3, 75-11
 - selecting, 77-11, 77-14, 77-17, 77-20
- Project - Multiple Projects tab, 77-10
- Project Status Report dialog box, 21-13
- projects
 - about, 19-1, 20-1, 20-2, 21-2
 - architecture, 20-2
 - cache type and size, 22-13, 28-35
 - comments, 22-21
 - configuring, 22-1
 - copying objects, 5-6
 - creating, 7-9, 21-1, 21-2, 21-3, 21-6
 - deployment login, and, 20-3
 - deployment overview, 2-14
 - descriptors, 19-2
 - development login, and, 20-4
 - direct access to mapped fields, 22-4
 - EIS, 20-7, 20-9, 24-1
 - errors, 14-2
 - existence checking, 22-8
 - exporting, 21-13, 21-14
 - for sessions, 77-9
 - indexed records, 20-9
 - Java, 21-3
 - JAXB, 21-6
 - locked, 5-6
 - login, 20-3, 74-1, 84-2
 - login, and, 20-3
 - mapped field access, default, 22-4
 - mapped records, 20-9
 - mapping projects, creating, 21-2
 - mappings, 19-2, 21-2
 - merging, 5-4
 - metadata, 2-20
 - method access to mapped fields, 22-4
 - model, exporting, 21-14
 - .mwp file, 4-1

- nondesoriptor classes, 4-51
- non-persistent, 20-2
- nonrelational, 20-2
- object-relational, types supported, 20-6
- OC4J, creating from, 7-9
- open, 21-10
- Oracle TopLink Workbench, 21-2
- packages, renaming, 4-52
- persistence type, 20-2, 22-7
- platforms, and, 20-4
- prior TopLink versions, 21-10
- recently opened, 21-10
- relational, 20-2, 20-6, 23-1
- renaming, 21-12
- reopening, 21-10
- saving, 21-11
- sequencing, 20-4, 20-5, 23-3, 86-4
- session login, and, 20-3
- sharing, 5-6
- status report, 21-12
- team development, 5-3
- TopLink Workbench, Navigator window, 4-9
- types of, 20-1, 21-2, 26-1
- updating from `ejb-jar.xml`, 21-16
- upgrading from 2.x or 3.x, 21-10
- writing `ejb-jar.xml` file, 21-16
- XML, 20-9, 25-1
- XML records, 20-9
- `projects.xml` file
 - about, 8-2
 - EJB 3.0, 8-3, 8-5
 - schema, 8-2
 - XSD file, 8-2
- propagation mode, cache, 91-2
- proxies. *see* wrapper policy
- proxy authentication
 - about, 84-5
 - applications, 84-5
 - Oracle Virtual Private Database, 75-21, 84-6, 86-13
 - session events, 75-6
 - use cases, 84-5
- proxy indirection
 - about, 33-8
 - example, 35-9
 - restrictions, 33-8
- proxy settings, preferences, 4-13
- public source code, 12-2

Q

- qualified names, database tables, 4-23, 4-24
- queries
 - about, 96-1
 - application development process, 2-13, 74-2
 - building, 96-6
 - cache, 96-29
 - `Call` queries, 96-3, 96-16
 - cascading, 96-14, 98-13
 - concepts, 96-2

- conforming, 102-8
- database queries, 96-10, 98-4
- `DatabaseQuery`, 96-3
- descriptor query manager, 96-3
- EIS interactions, 96-6
- EJB finders, 96-23
- EJB QL query language, 96-5
- `ejb-jar.xml` file, 28-10, 29-6, 31-6
- executing, 96-7
- expressions, 96-3
- fetch groups, 96-12
- flashback queries, 81-1, 96-18
- hierarchical queries, Oracle extensions, 96-18
- hints, Oracle extensions, 96-18
- historical, 96-20, 99-2
- interface and inheritance queries, 96-22
- joining, 96-12
- languages, about, 96-4
- named queries, 96-15
- object level modify query, 96-13, 96-14
- object level read query, 96-11, 96-13
- on inheritance hierarchies, 99-4
- on interfaces, 99-4
- optimizing, 11-16
- Oracle database features, 99-5
- Oracle extensions, 96-17, 96-18
- parameter binding, 98-17
- partial object query, 96-11
- performance, 11-16
- prepared statement caching, 98-17
- query keys, 96-4
- read all query, 96-11
- read object query, 96-11
- redirect queries, 96-19
- remote sessions, 75-31
- report, 11-21, 96-8
- report query, 96-15
- results, 96-8
- returning policy, 96-17
- session queries, 96-9, 98-1
- SQL query language, 96-4
- stored functions, Oracle extensions, 96-18
- subqueries, 97-10
- summary queries, 96-3
- timeout, 28-24, 28-26, 98-10
- types, 96-1
- `UpdateAll`, 96-14
- XML query language, 96-5
- Queries tab, 28-26, 28-28
- query by example, 98-6
- query cache, 96-36
- Query Exception, 13-30
- query keys
 - about, 28-29, 28-33, 96-4
 - adding, 28-31, 28-34
 - and expressions, 97-9
 - creating, 28-31, 28-34
 - direct mappings, 28-30
 - generating, 28-30
 - in expressions, 97-9

- interface descriptors, 28-31, 28-33
- Java implementation, 28-31
- modifying, 28-30
- order, 37-8
- relationship mappings, 28-30
- specifying, 28-31, 28-34
- unmapped attributes, 28-30, 28-31
- Query Keys tab, 28-31
- query object query. *See* DatabaseQuery
- query objects
 - batch reading, 98-10
 - cache expiration, 99-22
 - caching results, 96-36, 99-21
 - examples, 98-5
 - ordering for ReadAll queries, 98-8
 - report query, 11-21
 - specifying collection class, 98-9
- query results
 - about, 96-8
 - caching, 96-36
 - collections, 96-8
 - cursors, 96-8
 - reports, 96-8
 - streams, 96-8
- query timeout example, 98-10
- QueryManager
 - about, 26-8
 - joining expressions, 99-4
- QuerySequence, 86-8

R

- read access
 - providing in sessions, 75-15
- read all operation, 98-2
- read conversions
 - simple type translator, 33-13
- read only
 - descriptors, 28-4
 - files, 5-6
 - mappings, 35-2, 35-3
- read operation, 98-2
- read optimization
 - about, 11-19
 - batch reading, 11-19
 - fetch groups, 28-77
 - joining, 11-19
 - lazy attribute loading, 28-77
 - object indirection, 11-19
 - partial object reading, 11-19
 - report query, 11-20
 - soft cache weak identity map, 11-19
 - unit of work, 11-19
 - weak identity map, 11-19
- read queries
 - cascading refresh of identity maps, 96-35
 - identity map cache refresh, 96-35
 - refreshing identity maps, 96-35
- ReadAll finders
 - executing, 99-10
- ReadAll queries
 - ordering in query objects, 98-8
- readAllObjects()
 - about, 29-8
 - example, 98-2
- reading
 - ejb-jar.xml, 21-16
 - just-in-time reading, 33-8
 - whole XML documents, 39-1
- reading through the write connection, 102-27
- read-locking, 26-9
- readObject()
 - example, 98-2
- recently opened projects, 21-10
- redirect queries
 - about, 96-19, 99-1
 - creating, 99-1
 - finders, 99-13
 - using, 99-12
- reference key field, 36-11, 36-12
- reference object-relational mappings
 - configuring, 52-1, 53-1
 - understanding, 49-2
- ReferenceMapping class, 49-2
- references
 - database tables, 4-27, 4-28
 - foreign keys, 14-12, 14-21
- Refresh from Database button, 4-24
- refresh policy
 - cache, 90-8
 - EJB finders, 90-8
- refreshing
 - cache, 28-27
 - classes, 4-50
 - refreshObject(), 98-2
 - remote sessions, 75-31
 - sessions, 78-5
- registering objects, 102-1
- registerNewObject method, 102-2
- reimporting schemas, 4-36
- relation tables
 - about, 36-9
 - many-to-many mappings, 43-1
 - mappings, 43-2
- relational descriptors
 - associated table, 29-2
 - configuring, 29-1
 - locking policy, 28-62
 - understanding, 26-11
- relational mappings
 - about, 36-1
 - aggregate collection, 36-10, 44-1
 - aggregate object, 36-12, 47-1
 - configuring, 37-1
 - converters, 36-3
 - direct collection, 36-11, 45-1
 - direct map, 36-12, 46-1
 - direct-to-field, 36-4, 38-1
 - direct-to-XMLType, 36-4, 39-1
 - many-to-many, 36-8, 43-1

- one-to-many, 36-7, 42-1
- one-to-one, 36-5, 40-1
- options, 37-1
- order, 37-8
- transformation, 36-3, 36-15, 48-1
- variable one-to-one, 36-6, 41-1
- relational projects
 - about, 20-2
 - configuring, 23-1
 - object-relational databases, 20-6
 - relational databases, 20-6
 - understanding, 20-6
- relationships
 - about, 2-17
 - bidirectional, 4-31, 35-34, 36-5
 - expressions, 97-5
 - in `ejb-jar.xml` file, 21-16
 - unexpected multiplicity, 7-14
- relative locations
 - about, 22-2
 - classpath, 22-3
- Remote Command Manager Exception, 13-77
- remote connection using RMI, example, 76-10
- remote sessions
 - about, 75-2, 75-29
 - application layer, 75-30
 - creating, 76-10
 - limitations of, 75-31
 - securing access, 75-31
 - server layer, 75-31
 - transport layer, 75-30
 - unit of work, 75-32
- Remove Class button, 4-51
- Remove Table button, 4-24
- removing sessions from brokers, 82-1
- Rename dialog, 4-24
- renaming
 - packages, 4-52
 - projects, 21-10, 21-12
- reopening projects, 21-10
- report query
 - about, 96-15
 - query objects, 11-21
 - read optimization, and, 11-20
 - using, 11-21, 96-8
- reports
 - project status, 21-12
 - query results, 96-8
 - see also* status reports
- resuming unit of work
 - after commit, 102-14
 - after commit failure, 102-14
- returning policy
 - configuring, 28-65
 - SQLCall, 96-17
- Returning tab, 28-66
- RMI
 - coordinated cache, 93-1
 - naming service, 91-9
 - remote session support, 75-31

- rollback
 - and Java Transaction API, 100-11
 - overview, 100-11
 - with optimistic locking, 26-20
- root class
 - about, 26-12
 - inheritance, 26-12
- root EIS descriptor, 27-5
- root element, descriptor, 32-5
- runtime
 - acquiring sessions, 75-5
 - components, 3-2
 - services, configuring sessions, 77-14
 - troubleshooting, 15-14

S

- Save All button, 21-11
- Save button, 21-11
- saving projects, 22-2
- schema context
 - descriptors without, 32-3
 - EIS descriptors, 31-2
 - XML descriptors, 32-2
- Schema Document Info tab, 4-36, 4-38
- schema manager
 - about, 6-1
 - automatic table creation, 6-6
 - creating a table creator, 6-4
 - DB2, 6-3
 - default table generator, 6-4
 - Java table creator, 6-4
 - MS Access, 6-3
 - MySQL, 6-3
 - Oracle, 6-3
 - Oracle TopLink Workbench table creator, 6-4
 - sequencing, 6-3, 6-6
 - Sybase, 6-3
 - table creator, 6-2, 6-4, 6-6
 - table definition, 6-2
 - type conversion, 6-3
 - usage, 6-1
- Schema Structure tab, 4-34
- schemas
 - about, 2-16
 - context for EIS descriptors, 31-2
 - context for XML descriptor, 32-2
 - data storage, 2-16
 - database, 4-22
 - default root for EIS descriptors, 31-4
 - details, 4-34
 - document information, 4-36, 4-38
 - errors, 14-20
 - importing, 4-35
 - optimizing, 11-8
 - properties, 4-36
 - reimporting, 4-36
 - schema manager, 6-1
 - structure, 4-34
 - XML schemas, 4-34

- SCM. *see* source control management
- scripts
 - SQL, generating, 4-29
 - see also* SQL
- scrollable cursor
 - traversing, 99-16
 - using for ReadAllQuery, 99-16
- SDK
 - Data Store Exception, 13-71
 - Descriptor Exception, 13-75
- security
 - cache coordination, permissions, 7-23
 - data source access, permissions, 7-24
 - disabling doPrivileged(), 7-25
 - doPrivileged(), 7-4
 - EJB, permissions, 7-24
 - enabling doPrivileged(), 7-25
 - J2EE application, permissions, 7-24
 - java.util.logging, permissions, 7-24
 - loading project.xml, permissions, 7-23
 - loading sessions.xml, permissions, 7-23
 - logging, permissions, 7-24
 - password encryption, 21-13
 - permissions by feature, 7-23
 - permissions when doPrivileged()
 - disabled, 7-24
 - port permissions, 7-23, 7-24
 - SecurityManager integration, 7-4
 - system properties, permissions, 7-23
 - understanding permissions, 7-22
 - with BEA WebLogic, 7-20
- Select Classes dialog box, 4-50
- SelectedFieldsLockingPolicy, 26-21
- sensors, DMS profiler, 11-4, 75-11
- SEQ_COUNT column in sequence table, 20-20
- sequence connection pools, 84-8
- sequence numbers, write optimization
 - features, 11-26
- SEQUENCE objects in Oracle native
 - sequencing, 20-19
- sequencing
 - about, 20-14
 - BEA WebLogic, 20-21
 - BEA WebLogic single column sequence
 - table, 20-17, 23-3, 86-4
 - configuring, 20-15
 - connection pools, 84-8
 - default, 86-5, 86-6
 - DefaultSequence class, 86-6
 - descriptors, 26-9, 29-3
 - entity beans, 20-21
 - IBM WebSphere, 20-21
 - isolated client sessions, 75-24
 - Java configuration, 20-15
 - Microsoft SQL Server, 6-6
 - native, 20-18, 20-19, 20-20
 - non-Oracle database native, 20-19
 - Oracle Database native, 20-18, 20-20
 - Oracle TopLink Workbench configuration, 20-15
 - overriding default, 86-6
 - platform default, 86-5
 - preallocation, 11-28, 20-20
 - QuerySequence class, 86-8
 - schema manager, 6-6
 - SEQ_COUNT, 20-20
 - sequence type, configuring, 23-3, 86-4
 - sessions, and, 75-13
 - stored procedures, 20-18, 86-7
 - Sybase Database, 6-6
 - table, 20-16
 - table, default column and table names, 20-17
 - unary table, 20-17
 - with stored procedures, 86-7
- serialization
 - descriptor exceptions, 102-13
 - EJB 1.x, 33-9
 - EJB 2.x, 33-9
 - EJB 3.0, 33-9
 - indirection, 33-9
 - merging into session cache with unit of
 - work, 102-12
- serialized object converter
 - about, 12-2, 33-10
 - configuring, 35-18
- serialized object mappings, 35-19
- SerializedObjectMapping. *see*
 - SerializedObjectConverter
- server layer, 75-31
- server platform
 - external transaction controller, 77-14
 - runtime services, 77-14
 - session configuration, 77-14
 - session event listener, 77-17
- Server Platform tab, 77-15
- server sessions
 - about, 75-1, 75-13, 75-15
 - cache, 75-32
 - connection options, 75-18
- service channel, coordinated cache, 91-3
- session accessor
 - about, 2-30
 - managing dependent objects with, 2-30
- session beans
 - about, 2-25
 - deploying, 8-8, 9-3, 10-1
 - model, 2-23, 2-25
 - remote session support for, 75-31
- session brokers
 - about, 75-2, 75-25
 - adding sessions to, 82-1
 - alternatives, 75-28
 - architecture, 75-26
 - configuration, 82-1
 - limitations of, 75-27
 - renaming, 82-1
 - two-phase commits, 75-27
 - two-stage commits, 75-27
- session configuration file
 - about, 76-1
 - loading alternative, 78-4

- preferences, 4-18, 4-19
 - see also* sessions.xml file
- session customizer, 75-4
- Session Event Manager, 75-5
- session events
 - example, 75-5, 75-6
 - isolated sessions, and, 75-5
 - manager, 75-5
 - proxy authentication, and, 75-6
- Session Loader Exceptions, 13-56
- Session Manager
 - about, 78-1, 78-2
 - acquiring, 78-2, 78-3
 - defaults, 78-3
 - destroying sessions, 78-10
 - J2EE defaults, 78-3
 - sessions, acquiring, 75-5
 - storing sessions, 78-10
- sessions
 - about, 2-12, 74-1, 75-1, 75-2, 76-1
 - acquiring at runtime, 75-5, 78-1
 - adding to session brokers, 82-1
 - additional mapping projects, 77-9
 - API, 75-33
 - application server logging, 75-9
 - architecture, 75-2
 - cache, 75-32, 90-2
 - client, 75-13
 - configuring, 77-1, 77-17
 - connection policy, 77-19
 - creating, 2-21, 76-1, 76-4, 76-6, 76-8, 88-1
 - customization, 75-4
 - database sessions, 75-28, 76-8
 - destroying in Session Manager, 78-10
 - errors, 14-21
 - event listeners, 75-7, 77-17
 - events, 75-6
 - external transaction controller, 77-14, 100-2
 - historical client sessions, 75-24
 - in sessions.xml file, 75-4
 - isolated client sessions, 75-19, 80-1
 - loading with alternative class loader, 78-4
 - logging, 75-7, 75-9, 75-10, 77-4, 77-5
 - logging into, 78-9
 - logging out of, 78-10
 - logins, 77-4
 - management of, 78-1
 - metadata, about, 2-21
 - multiple sessions, 75-28, 78-2
 - named queries, 77-21
 - object cache, 75-3
 - optimizing, 11-13
 - preferences, 4-18, 4-19
 - queries, 96-7, 96-9, 98-1
 - refreshing, 78-5
 - registering descriptors, 75-12, 77-2, 77-3, 77-10
 - remote sessions, 75-29, 76-10
 - sequencing, about, 75-13
 - server, 75-13
 - server platform, 77-14, 77-15
 - SQL and messages, 75-10
 - storing in Session Manager, 78-10
 - three-tier architecture, 75-14
 - transformation mappings, 36-15
 - types, 75-1
 - unit of work, 75-1, 75-19
- sessions.xml file
 - about, 8-3
 - acquiring, 75-5
 - CMP applications, 8-4, 75-4
 - configuring with JDeveloper, 5-3
 - creating, 76-1
 - default location, 75-5
 - EJB 3.0, 8-5
 - loading alternative configuration file, 78-4
 - non-CMP applications, 8-4
 - schema, 8-4
 - sessions, 75-4
 - XSD file, 8-4
- setAdditionalJoinExpression(), 99-5
- setenv.cmd file, 4-2
- setenv.sh file, 4-2
- setMaxBatchWritingSize(), 11-15
- setMultipleTableJoinExpression(), 99-5
- setShouldPerformDeletesFirst(), 102-16
- Settings tab, 28-26, 28-28
- setValue() method, 33-7
- shared library
 - setting for BEA WebLogic, 7-15
- shared library for BEA WebLogic, 7-15
- simple type translators
 - about, 12-2, 33-12
 - configuring, 35-23
 - in Java, 35-25
 - read conversions, 33-13
 - write conversions, 33-14
- soft cache weak identity map
 - about, 90-4
 - when to use, 90-5
- sorting, in memory, 35-26
- source code, public, 12-2
- source control management
 - projects, 5-3
 - with Oracle TopLink Workbench, 5-3
 - see also* team development
- source table, reference, 4-28
- source.jar, 12-2
- splash screen, 4-13
- SQL
 - call, 98-18
 - comparing with expressions, 97-1
 - custom queries for basic persistence, per descriptor, 29-6
 - EJBQLCall, 98-23
 - generating from database tables, 4-29
 - parameter binding, 11-15
 - parameterized, 11-26, 98-17
 - prepared statement caching, 11-15
 - queries, 96-4
 - scripts with binding arguments, 29-6

- SQLCall, 98-18
 - StoredFunctionCall, 98-22
 - StoredProcedureCall, 98-20
 - unit of work, 102-15
- SQL Creation Script dialog box, 4-29
- SQL DISTINCT, 11-24
- SQL Exception, 13-27
- SQLAnywhere platform, 84-3
- SQLCall
 - about, 96-17
 - binding input parameters, 98-19
 - input parameters, 98-19
 - input-output parameters, specifying, 98-20
 - output parameters, 98-19
 - Returning Policy, 96-17
 - using, 98-18
- SQLServer platform, 84-3
- stages of development with Oracle TopLink, 2-1
- stale data
 - cache, 90-6, 90-7
 - coordination, cache, 90-7
 - invalidating the cache, 90-7
 - locking policy, and, 90-6
 - per-class cache configuration, 90-7
 - per-query cache refresh, 90-7
- StartWith, 99-7
- stateful
 - beans, 2-25
 - comparing with stateless, 2-25
- stateless
 - comparing with stateful, 2-25
- static attributes, 4-45, 4-48
- static fetch groups, querying with, 99-3
- status report, generating, 21-12
- stored functions
 - about, 96-18
 - using, 98-22
- stored procedures
 - output parameter event, 98-22
 - queries, 98-20
 - sequencing, and, 20-18, 86-7
- streams
 - as query results, 96-8
 - cursor, 75-32, 99-18
- string bindings, CLOB mappings, 34-4
- structure object-relational mappings
 - configuring, 51-1
 - understanding, 49-2
- StructureMapping class, 49-2
- subqueries
 - multiple expressions, 97-10
 - subselects in expressions, 97-10
- subselects, in expressions, 97-10
- summary queries, 96-3
- superclass, 4-41
- Sybase
 - database schema manager type conversion, 6-3
 - native sequencing, 20-19
 - platform, 84-3
- Synchronization Exceptions, 13-70

- synchronous change propagation, 91-2
- system properties
 - oracle.j2ee.security.usedoprivileged, 7-25
 - oracle.j2ee.toplink.security.usedoprivileged, 7-25
 - toplink.cts.collection.checkParameters, 7-25
 - toplink.xml.platform, 7-3

T

- table creator
 - about, 6-2
 - creating, 6-4
 - using, 6-6
- Table Creator dialog, 6-4
- table generation. *see* automatic table generation
- table sequence
 - about, 20-16
 - default column and table names, 20-17
- table-class relationships, 16-6
- tables
 - adding database, 4-22
 - associating with relational descriptors, 29-2
 - database, 4-21
 - defining schema, 6-2
 - errors, 14-15
 - generating, automatic, 33-4
 - import filter, 4-23
 - mapping to descriptors, 28-3, 29-2
 - merging files, 5-5
 - multiple, 29-13
 - primary key, 4-26
 - references, 37-8
 - relation tables for mappings, 43-2
 - TableDefinition class, 6-2
 - see also* database tables
- target foreign keys
 - about, 36-6, 37-9
 - configuring, 37-9
- target platforms
 - about, 2-5
 - choosing, 2-3
- target tables
 - in direct collection mappings, 45-1
 - reference, 4-28
- team development, 5-3
- technical support, 2-3
- three-tier architecture
 - about, 1-4, 2-21
 - authentication, 84-5
 - migrating to scalable architecture, 75-29
 - overview, 2-21
 - sessions, 75-14
- timestamp support
 - about, 37-3
 - direct to field mappings, 37-3
 - Oracle Database, 37-3
 - TIMESTAMP timezone, 37-4
- TimestampLockingPolicy, 26-18
- timezone, with TIMESTAMP, 37-4

- tljxb.cmd file, 21-9
- toolbars, 4-4, 4-6
- topic name, 92-1
- TopLink. *see* Oracle TopLink
- TopLink expressions. *see* expressions
- TopLink profiler
 - about, 11-2
 - selecting, 77-11, 77-14, 77-17, 77-20
- TopLink Workbench. *see* Oracle TopLink Workbench
- toplink-ejb-jar.xml file, 8-6, 19-2
- toplink-ejb.xml file, 8-8
- Transaction Exception, 13-79
- transactional data sources, 84-1
- transactions
 - client-controlled, 102-24
 - CMP, 100-3, 102-24, 102-25
 - container-controlled, 102-24
 - demarcation, 100-2
 - external transaction controller, 77-14, 100-2
 - external, integrating, 100-2, 102-20
 - isolated client sessions, 75-24
 - isolation, 100-2, 100-4
 - JTA, 100-3
 - JTS, 100-3
 - local, 102-25
 - local, CMP, 102-25
 - OTS, 100-3
 - overview, 2-13, 74-3, 100-1
 - see also* unit of work
- transformation EIS mappings
 - configuring, 64-1
 - understanding, 56-17
- transformation mappings
 - about, 33-14
 - attribute transformation, 35-29, 35-31
 - mutable, 35-33
- transformation relational mappings
 - configuring, 48-1
 - understanding, 36-15
- transformation XML mappings
 - configuring, 73-1
 - understanding, 65-31
- transient attributes, 4-45, 4-48
- transparent indirection
 - about, 33-8
 - persistent class requirements, 2-11
- transport layer, 75-30
- troubleshooting
 - BEA WebLogic deployment, 15-3
 - data sources, 14-23
 - deploy tool, 10-6
 - deployment, 15-1
 - ejbc, 10-3
 - IBM WebSphere deployment, 15-12
 - migration from OC4J persistence, 7-13
 - Oracle TopLink Workbench, 14-1
 - unit of work, 102-30, 102-34
- tutorial
 - about, 16-1
 - database schema, 16-4
 - planning, 16-1
 - two-phase commits, 75-27
 - two-stage commits, 75-27
 - two-tier architecture
 - about, 1-6, 2-23
 - authentication, 84-5
 - understanding, 2-23
 - type conversion
 - automatic, 37-2
 - NCHAR, 33-12
 - NCLOB, 33-12
 - NVARCHAR2, 33-12
 - oracle.sql.TimeStamp, 37-3
 - schema manager, 6-3
 - String to TIMESTAMP, 33-12
 - TIMESTAMP to String, 33-12
 - type conversion converter
 - about, 12-2, 33-11
 - configuring, 35-20
 - provided by direct-to-field mappings, 37-3
 - TypeConversionMapping
 - see* TypeConversionConverter
 - types of mappings, 33-1
 - typesafe enumeration, in EIS mappings, 56-3

U

- unary table sequence
 - about, 20-17
 - BEA WebLogic single column sequence
 - table, 20-17, 23-3, 86-4
- undeployment, 10-4
- unexpected relationship multiplicity, 7-14
- unidirectional relationships, 36-2
- unit of work
 - about, 100-1, 100-2, 100-4
 - acquiring, 101-1
 - API, 100-11
 - architecture, 100-1
 - auditing, 102-20
 - benefits of, 100-4
 - cache, 90-2
 - change policy, 100-6
 - clones, 100-9
 - CMP integration, 100-3
 - commit and Java Transaction API, 100-10
 - commit, writing changes before, 102-7, 102-11, 102-23
 - conform results of in-memory query, 96-31
 - creating objects, 101-2
 - deleting objects, 101-7
 - example, 100-6
 - external transaction controller, 77-14, 100-2
 - external transactions, 100-2, 102-20
 - integrating with CMP, 102-24
 - isolation, 100-4
 - JTA integration, 100-3
 - JTS integration, 100-3
 - life cycle, 100-5
 - modifying objects, 101-2

- mutable mappings, 35-33
- nested, 100-9, 102-15
- `newInstance` method, 102-2
- optimization, 11-30, 100-11
- OTS integration, 100-3
- parallel, 100-9, 102-15
- pre-commit validation, 102-34
- primary keys, 100-11
- proxy indirection, 33-8
- queries, 100-12
- read optimization, 11-19
- read-only classes, 102-6, 102-7
- `registerNewObject` method, 102-2
- remote sessions, 75-32
- resuming, 102-14
- reverting, 102-14
- rollback, 100-11
- sessions, 75-1, 75-19
- transaction demarcation, 100-2
- transactions, 100-4
 - unit of work
 - demarcation, 100-2
- troubleshooting, 102-30
- validating objects, 102-34
- with custom SQL, 102-15
- write optimization, 11-26
- writing changes before commit, about, 102-7
- writing changes before commit, and external
 - transaction exceptions, 102-23
- writing changes before commit, and external
 - transaction timeouts, 102-23
- writing changes before commit, as alternative to
 - conforming, 102-11
- unmapping, 34-6
- unregistering objects, 102-1
- unsaved items, displaying in TopLink Workbench
 - Navigator window, 4-9
- update
 - operation, 98-3, 98-4
 - projects from prior versions, 21-10
- `UpdateAll` query, 96-14
- `updateObject()`, 29-8
- Use XML Schema "type" attribute,
 - configuring, 35-23
- `useBatchWriting()`, 86-11
- user-defined functions, in expressions, 97-16

V

- validating
 - descriptors, 27-6
 - JAXB, 20-14
 - projects, 21-12
 - unit of work, 102-34
- Validation Exceptions, 13-42
- Value Converter tab, 46-4
- value holders
 - about, 33-6
 - `ValueHolder` class, 33-6
- `ValueHolderInterface` class, 2-11, 33-7, 36-9

- variable one-to-one relational mappings
 - class indicator, 41-1
 - configuring, 41-1
 - primary key, unique, 41-3
 - understanding, 36-6
- `VariableOneToOneMapping` class, 36-6
- Varray in Oracle database. *see* array mappings
- version control, 5-6
- Version Control Assistance dialog box, 5-6
- `VersionLockingPolicy`, 26-18
- volatile attributes, 4-45, 4-48
- VPD. *see* Oracle Virtual Private Database

W

- warning icon, 4-10, 4-11
- weak identity map, 90-3
- web browser, specifying, 4-14
- web services architecture, 1-5, 2-36
- WebLogic. *see* BEA WebLogic
- `weblogic-ejb-jar.xml` file
 - described, 8-5
 - modifying for Oracle TopLink, 8-13
 - unsupported tags, 8-14
- WebSphere. *see* IBM WebSphere
- welcome screen, 4-14
- wildcard, 66-3
- wrapper policy
 - about, 28-75
 - implementing in Java, 28-76
- write
 - conversions, simple type translator, 33-14
 - write all operation, 98-3
- write query
 - disabling identity map cache, 98-14
 - non-cascading, 98-13
 - objects, 98-12
 - overview, 98-12
- write-locking, 26-9
- writing
 - batch, 11-26, 86-11
 - `ejb-jar.xml` file, 21-16
 - optimization, 11-26
 - sessions write access, 75-16

X

- `xdb.jar` file, 4-3
- XML
 - descriptor, schema context, 32-2
 - generating deployment, 8-3
 - mappings, concepts, 65-2
 - preserving data, 32-6
 - projects, 21-3
 - query language, 96-5
 - reading whole documents, 39-1
 - records, 56-3
 - schemas, 4-34
- XML Conversion Exception, 13-80
- XML Data Store Exceptions, 13-64

- XML descriptors
 - configuring, 32-1
 - schema context, 32-2
 - understanding, 26-12
- XML mappings
 - about, 65-1
 - any collection, 65-29, 72-1
 - any object, 65-27, 71-1
 - any type support, 65-4
 - composite collection, 65-25, 70-1
 - composite direct collection, 65-14, 68-1
 - composite object, 65-21, 69-1
 - concepts, 65-2
 - configuring, 66-1
 - default conversion pairs, customizing, 65-11
 - direct, 65-5, 67-1
 - extensions, 65-4
 - jaxb:class support, 65-4
 - list support, 65-3
 - reference descriptor, configuring, 66-2
 - transformation, 65-31, 73-1
 - types of, 65-1
 - union support, 65-3
 - xsd:list, 65-3
 - xsd:union, 65-3
- XML parser platform
 - about, 7-2
 - configuring, 7-3
 - creating, 7-3
 - Crimson, 7-3
 - default, 7-3
 - limitations, 7-3
 - parser conflicts, 7-3
 - toplink.xml.platform system property, 7-3
 - used by, application server, 7-2
 - used by, Oracle TopLink, 7-2
- XML projects
 - configuring, 25-1
 - JAXB support, 20-10
 - sequencing, 20-5
 - understanding, 20-9
- XML queries, 96-5
- XML schema
 - jaxb:class, and EIS mappings, 56-3
 - jaxb:class, and XML mappings, 65-4
 - jaxb:class, understanding, 33-20
 - type, 35-23
 - xs:any, understanding, 65-4
 - xs:anyType, understanding, 65-4
 - xsd:list, understanding, 33-17
 - xsd:union, understanding, 33-17
 - see also* schemas
- XML Type functions, 97-4
- XPath
 - by name, 33-16
 - by position, 33-15
 - mapping Java attributes, 35-10
 - support in OX mappings, 33-15
 - support in XML mappings, 56-3, 65-3
- XSD file

- projects.xml file, 8-2
- sessions.xml file, 8-4

Z

- zero-argument constructors
 - editing, 4-16

