



SIEBEL ANALYTICS PERFORMANCE TUNING GUIDE

VERSION 7.5

12-BD363V

JULY 2002

Siebel Systems, Inc., 2207 Bridgepointe Parkway, San Mateo, CA 94404
Copyright © 2002 Siebel Systems, Inc.
All rights reserved.
Printed in the United States of America

No part of this publication may be stored in a retrieval system, transmitted, or reproduced in any way, including but not limited to photocopy, photographic, magnetic, or other record, without the prior agreement and written permission of Siebel Systems, Inc.

The full text search capabilities of Siebel eBusiness Applications include technology used under license from Fulcrum Technologies, Inc. and are the copyright of Fulcrum Technologies, Inc. and/or its licensors.

Siebel, the Siebel logo, TrickleSync, TSQ, Universal Agent, and other Siebel product names referenced herein are trademarks of Siebel Systems, Inc., and may be registered in certain jurisdictions.

Other product names, designations, logos, and symbols may be trademarks or registered trademarks of their respective owners.

U.S. GOVERNMENT RESTRICTED RIGHTS. Programs, Ancillary Programs and Documentation, delivered subject to the Department of Defense Federal Acquisition Regulation Supplement, are “commercial computer software” as set forth in DFARS 227.7202, Commercial Computer Software and Commercial Computer Software Documentation, and as such, any use, duplication and disclosure of the Programs, Ancillary Programs and Documentation shall be subject to the restrictions contained in the applicable Siebel license agreement. All other use, duplication and disclosure of the Programs, Ancillary Programs and Documentation by the U.S. Government shall be subject to the applicable Siebel license agreement and the restrictions contained in subsection (c) of FAR 52.227-19, Commercial Computer Software - Restricted Rights (June 1987), or FAR 52.227-14, Rights in Data—General, including Alternate III (June 1987), as applicable. Contractor/licensor is Siebel Systems, Inc., 2207 Bridgepointe Parkway, San Mateo, CA 94404.

Proprietary Information

Siebel Systems, Inc. considers information included in this documentation and in Siebel eBusiness Applications Online Help to be Confidential Information. Your access to and use of this Confidential Information are subject to the terms and conditions of: (1) the applicable Siebel Systems software license agreement, which has been executed and with which you agree to comply; and (2) the proprietary and restricted rights notices included in this documentation.

Contents

Introduction

How This Guide Is Organized	8
Revision History	8

Chapter 1. Performance Tuning

Overview of Performance Tuning	10
Working Assumptions	12
Suggested Approach	12
Siebel Analytics	13
Underlying Database Servers	14
The Extract, Load, and Transform Process	16

Chapter 2. Siebel Analytics

Analytics Considerations	18
Enable Query Caching	19
Benefits	19
Costs	20
Reference Materials	20
Add Aggregate Tables	21
Aggregates Shipped with Siebel Analytics	22
Identify Candidates	22
Basic Guidelines	24
Adding and Managing Aggregate Tables	25
Packaged Aggregate Tables	26

Subset Large Physical Dimension Tables	27
Subset Dimension Tables	27
Review Configuration Parameters	35
Siebel Analytics Configuration Parameters	35

Chapter 3. The Database Servers

Overview of Database Indexing	38
More Than One Kind of Index	38
Database Optimizers	38
Index Selection	39
B-Tree Indexes	39
Bitmap Indexes	48
Vendor Index Evaluation Tools	51
Determinants of Index Efficiency	52
Periodic Tuning Tasks	55
Periodic Index Evaluation	55
Refresh Optimizer Statistics	57
Reorganize Indexes	60
Drop Indexes	62
Review Configuration Parameters	64
Computer Host	65
Miscellaneous Tips	65

Chapter 4. The ETL Process

Remove Unused Batches	68
Example Company	68
Rearrange Batches for Balance	77
Example	77

Drop and Recreate Indexes	79
Create and Drop Batch Scripts	79
Dummy Sessions and Initialization	81
Batches for the Initial Extract	82
Split Index Script Files for Parallelism	89

Introduction

This document, the *Siebel Analytics Performance Tuning Guide*, shows you how to improve the query performance of Siebel Analytics applications by addressing performance issues that yield the highest payoff.

This guide is useful for those who perform one or more of the following roles:

Database Administrators	Persons who administer the database system, including data loading; system monitoring, backup, and recovery; space allocation and sizing; and user account management.
Siebel Application Administrators	Persons responsible for planning, setting up, and maintaining Siebel applications.
Siebel Application Developers or Configurators	Persons who plan, implement, and configure Siebel applications, possibly adding new functionality.
Siebel System Administrators	Persons responsible for the whole implementation, including installing, maintaining, and upgrading Siebel applications.

This guide assumes that you are knowledgeable in the areas of relational databases, decision support systems, dimensional design, and the operating system which supports the Siebel Analytics components.

How This Guide Is Organized

This guide shows you how to improve the performance of the Siebel Analytics server by taking steps which have proven to be effective in a large number of cases. The suggested actions represent what database and data warehouse performance analysts consider standard practices.

This guide consists of four chapters:

- [Chapter 1, “Performance Tuning,”](#) summarizes the principal actions you can take to improve the performance of Siebel Analytics.
- [Chapter 2, “Siebel Analytics,”](#) describes how you can improve the performance of Siebel Analytics using Query Caching, Aggregates, and Configuration parameters.
- [Chapter 3, “The Database Servers,”](#) describes how to improve performance by evaluating and tuning the underlying databases.
- [Chapter 4, “The ETL Process,”](#) describes how to improve the performance of the process that extracts, transforms, and loads data into the underlying databases.

Revision History

Siebel Analytics Performance Tuning Guide, Version 7.5

This chapter provides an overview of the three areas of performance tuning described in this document:

- [Siebel Analytics on page 13](#)
- [Underlying Database Servers on page 14](#)
- [The Extract, Load, and Transform Process on page 16](#)

Overview of Performance Tuning

Siebel Analytics allows users to access data distributed across multiple independent databases through a single, unified, and easily understood business model.

Therefore, when you tune Siebel Analytics for optimal performance, you must also tune the underlying database servers, the communication network, and the extract, transform, and load routines. Otherwise, performance will fall below optimal.

You must also make sure that sufficient hardware resources are available to support the entire workload. That means, a sufficient amount of memory, a sufficient number of processors, a sufficient amount of physical disk space with the requisite characteristics, and ample network bandwidth to support the entire workload.

This chapter summarizes specific actions you can take to improve the performance of Siebel Analytics, the underlying database servers, and the extract, transform, and load routines. The actions are organized into three areas:

- Siebel Analytics server
- Underlying database servers
- Extract, transform, and load routines

Subsequent chapters describe these actions in more detail and provide examples.

All the actions described represent steps which have proven effective in a large number of cases, and most database administrators and performance analysts consider these actions as part of their standard practices.

Keep in mind that these actions are intended as guidelines, and the results depend highly on your particular workload, data shape, and requirements. Before summarizing the performance actions, the next section reviews a few working assumptions as well as a suggested approach to performance tuning. You should also review the logical and physical database schemas from a performance point of view. [Figure 1](#) shows the three performance areas addressed in this guide.

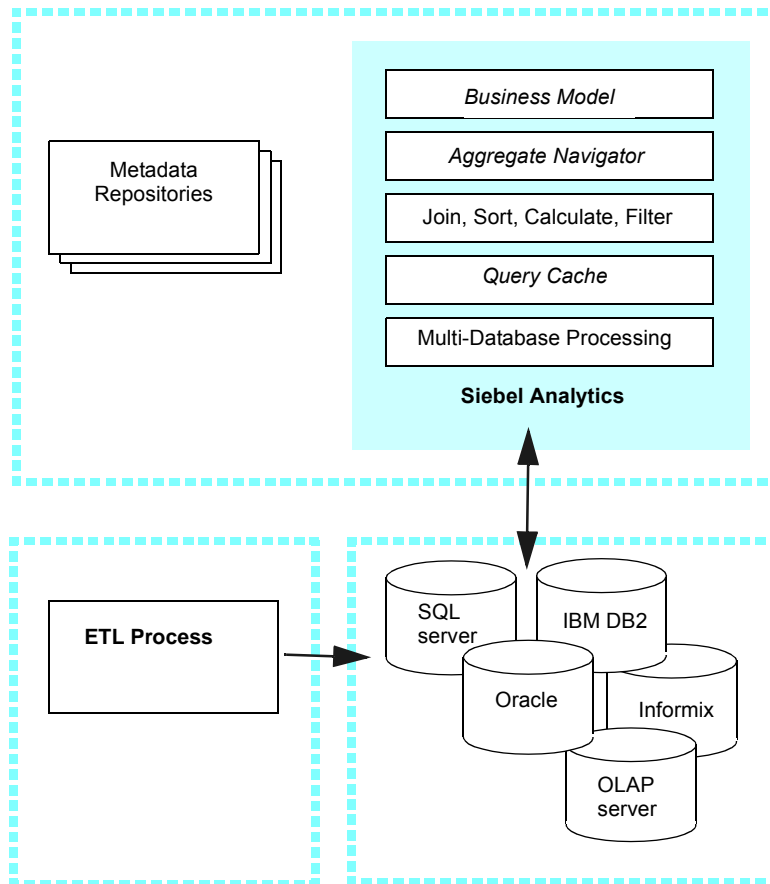


Figure 1. The Three Areas

Working Assumptions

The suggested performance improvement actions assume that sufficient computing and network resources are available to support your actual workload, and the workloads have been isolated to minimize interference:

- CPU, Memory, and I/O system
- Network Services
- Connection pooling (Siebel Analytics server relief)
- Isolation of the Data Warehouse Workload from Transaction Workloads

Without sufficient capacity, resource bottlenecks at constraining resources will result in poor query response time. A single bottleneck could be the source of all user complaints about performance. Any serious effort starts with a high-level survey of the computer host and its operating system to determine whether *obvious* bottlenecks exist.

Suggested Approach

When you address performance issues, follow a deliberate approach: Make a single change at a time, evaluates the impact of that change, and then continue on with the next change. Before you make a change, you should develop a plan to recover from the change should unintended consequences ensue.

NOTE: As you start taking actions to improve performance, the initial actions can be dramatic and come at a relatively small cost. As a consequence, later actions will be less dramatic and require much more effort. Performance tuning is a balancing act. A change that improves the performance of one area may adversely impact the performance of another area. You should evaluate the overall impact of a change based on your requirements. For example, adding new indexes or aggregate tables may help in improving the performance of a report but may also slow down ETL processing time.

Siebel Analytics

You can take several actions to improve the performance of the Siebel Analytics server. [Table 1](#) summarizes the principal actions.

Table 1. Performance Improvements for Siebel Analytics

Action	Description
Enable Query Caching	Verify that query caching is enabled.
Create Aggregates	Create aggregates that the Siebel Analytics server can use to rewrite queries.
Evaluate Design	Make your logical objects efficient. For example, divide large dimension into a set of smaller ones.
Evaluate Reports	Evaluate your reports and optimize. For example, look at frequently run Dashboard reports and make sure that only the columns required by the reports are selected. Remove all <i>superfluous</i> columns.
Review Configuration	Verify that the configuration of Siebel Analytics matches your actual workload. For example, configure a pool sufficient to support connections, case sensitive character comparisons, driving tables within queries, and query caching.

These are the most common actions you can take to improve the performance of the Siebel Analytics server.

Underlying Database Servers

Several actions can be taken to improve the performance of underlying database servers. Some of these can be taken immediately while others require some thought. [Table 2](#) summarizes the principal actions you can take to improve database server performance.

Table 2. Performance Improvements for Underlying Database Servers

Action	Description
Update Optimizer Statistics	All the database servers which Siebel Analytics supports store statistics used by their optimizers. If these statistics are stale, the optimizer can choose a suboptimal plan.
Reorganize Indexes	When tables in a database are modified by insert, delete, or load operations, the indexes on the tables become less efficient. The indexes on tables subject to updates should be periodically reorganized.
Drop Indexes	Indexes occupy storage space, provide alternate access paths which must be evaluated by the optimizers, must be maintained, and degrade update operations. Periodic surveys should be made to determine whether an index is being referenced by user queries. If not, remove the index.

Table 2. Performance Improvements for Underlying Database Servers

Action	Description
Create Indexes	<p>After query caching and aggregates, indexes are the most effective way to improve query performance. Create indexes to speed-up access times and improve join operations.</p> <p>Speedup access times by:</p> <ul style="list-style-type: none"> ■ Creating B-tree indexes on dimension table columns that have a large number of distinct values. ■ Creating Bitmap indexes on dimension columns that have few distinct values. These are most effective when queries constrain multiple columns which have bitmap indexes (Oracle only). <p>Speedup join operations by:</p> <ul style="list-style-type: none"> ■ Creating multicolumn B-tree indexes on foreign key reference columns of fact tables, and multicolumn B-tree indexes including the primary key and selected columns from the dimension tables. Indexes should be designed sensibly and ideally should not contain more than 5 columns. ■ Creating bitmap indexes on foreign key reference columns where the index value has a low degree of selectivity (more than 3 - 5% of the table values on average per value). (Oracle only).
Increase Parallel Query	<p>If you have sufficient computing resources, configure the database server for an optimal degree of parallel processing. Given sufficient resources, this can greatly improve query response time.</p>

Table 2. Performance Improvements for Underlying Database Servers

Action	Description
Manage I/O Traffic	Manage the input and output accesses to disk storage by striping the disk storage. The best and simplest action is to install disk storage arrays (RAID), the second best is to stripe volumes using a Logical Volume Manager.
Review Configuration	<p>Database vendor all have suggested configurations for specific kinds of workloads. Evaluate your workload, and configure the database server accordingly.</p> <ul style="list-style-type: none">■ Oracle <i>Oracle9i Data warehousing Guide and the Oracle9i Database Performance Guide and Reference</i>■ IBM <i>DB2 Administration Guide V7.1, Volume 3: Performance.</i>■ Microsoft SQL Server <i>RDBMS Performance Tuning Guide for Data Warehousing.</i>

These are the most common actions you can take to improve the performance of database servers.

The Extract, Load, and Transform Process

Several actions can be taken to improve the performance of the extract, transform, and load process. They are described in [Chapter 4, “The ETL Process.”](#) For additional guidelines on performance tuning of the ETL process, refer to the Informatica documentation.

This chapter describes the following actions you can take to directly improve the performance of Siebel Analytics:

- [Enable Query Caching on page 19](#)
- [Add Aggregate Tables on page 21](#)
- [Subset Large Physical Dimension Tables on page 27](#)
- [Review Configuration Parameters on page 35](#)

Analytics Considerations

Keep in mind that you must have sufficient hardware resources to support the query workload. While changes in the business model or queries can reduce the demand for computer resources, enabling query caching and creating aggregate tables can increase the demand for some resources (see [Figure 2](#)).

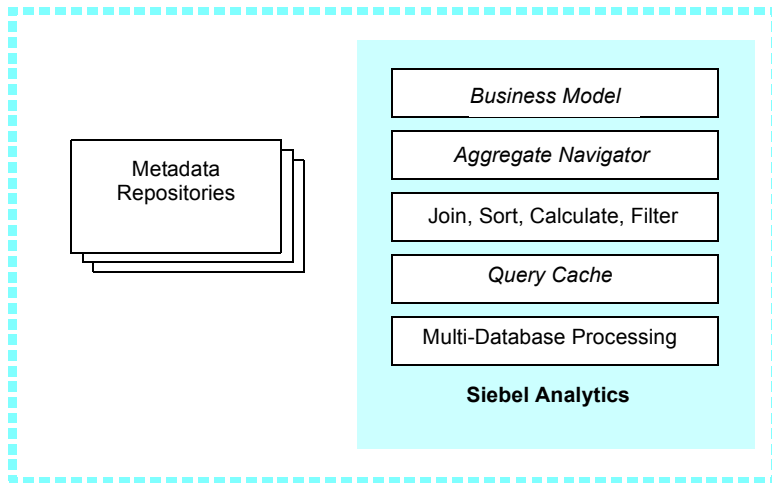


Figure 2. Siebel Analytics Server

Enable Query Caching

Decision support queries are usually complex and process a large number of rows. The fastest way to process such a query is to process the query once, save its result set in a cache, and then use the cached results to answer subsequent queries. The cached results can also be used to answer queries that request a logical subset of the cached results.

That is the basic idea of query caching, a technique Siebel Analytics uses which can dramatically improve query response time as well as reduce the overall demand for computing resources. Moreover, the user benefits without any knowledge of the underlying machinery.

Siebel Analytics is shipped with query caching enabled. You do need to configure the cache storage and decide on a strategy for flushing the outdated entries. The parameters that control query caching are located in the `NQSConfig.ini` file, which are described in the *Siebel Analytics Installation and Configuration Guide*.

Benefits

Query caching is similar to the technique used by database servers to rewrite queries to use precomputed results stored in aggregate tables. The advantage of query caching is that Siebel Analytics answers the query from its cache instead of forwarding it on to the database server for processing. Note that query caching still takes full advantage of query rewrites by the database on the first time the query is issued.

Query caching actually extends beyond a single query to the class of queries that can make use of the cached results. For example, Siebel Analytics can use the cached result set for queries that require a qualified subset of the cached results. Thus, a single cached result set can improve the average response time of a family of similar queries.

Query caching also conserves network resources and processing time. The demand on the network is reduced because intermediate results do not need to be transferred over the network to Siebel Analytics. The demand of the database server and computer host is reduced because the query workload has been reduced. Therefore the efficiency of query caching improves query performance and increases the capacity of computer and network resources.

Costs

Query caching incurs the following costs:

- Disk space for the cache
- Administrative costs of managing the cache
- Potential for cached results being stale
- Minor CPU and disk I/O on server machine

With proper cache management, the benefits will far outweigh the costs. Using cache, the cost of database processing need only be paid once for a query, not every time the query is run.

Reference Materials

Siebel Analytics Server Administration Guide describes how query caching operates in detail, and the *Siebel Analytics Installation and Configuration Guide* describes how to configure query caching for your Siebel applications.

Add Aggregate Tables

Aggregate tables are among the most important methods that can be used to improve query performance. Ralph Kimball asserts in his books on Data Warehousing that they are the single most effective method a designer or administrator has to improve query performance.

The reason why aggregate tables are so powerful is clear; once aggregate tables have been created and identified as data sources, the Siebel Analytics server can rewrite queries to reference the smaller, more compact aggregate tables instead of larger detail tables. Any server can read thousands of rows much faster than it can read, join, group, order, and process millions of rows.

Typically, aggregate tables greatly improve performance when the table compresses detail data by a factor of ten or more. By reducing the demand on resources, aggregate tables also effectively increase the capacity of a computer system.

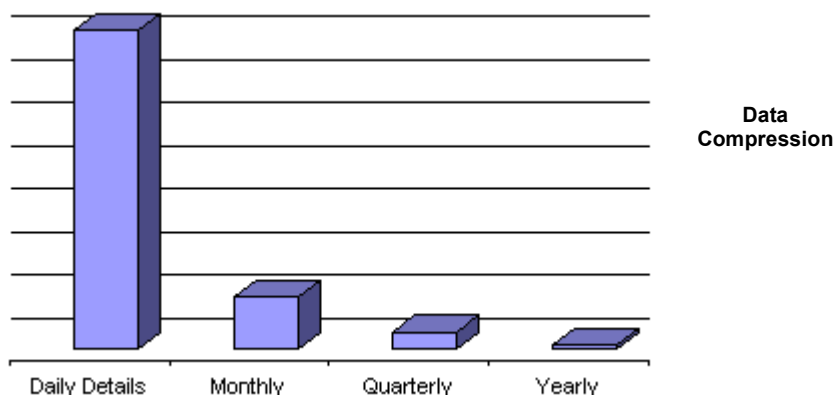


Figure 3. Detail Versus Aggregate Tables

Figure 3 illustrates a situation where a Monthly aggregate table would improve the performance of queries that summarize data to a monthly level. Siebel Analytics can also use the Monthly table to rewrite queries that summarize data at quarterly and yearly levels. Thus, there is no need to create, load, and manage the two additional higher-level aggregate tables.

Aggregates Shipped with Siebel Analytics

Siebel ships many aggregate tables with Siebel Analytics which are used to improve query performance.

For example, the aggregate table W_PLREVN_OP_A summarizes revenue measures across Product Lines. In many cases, this aggregate table can greatly compress the rows stored in the W_REVN_F base fact table, a fact table that stores revenue measures at the product level.

For a list of aggregate tables shipped with Siebel Analytics, see the *Siebel eBusiness Data Warehouse Data Model Reference* guide.

Identify Candidates

You will need to identify a set of candidates for aggregate tables as it is unrealistic to create and load every combination of possible aggregate tables. To see this, enumerate the number of possible aggregate tables for a simple fact table that has three dimensions, and each dimension has four possible aggregations.

To identify candidates for aggregate tables, you need to consider two factors:

- Query access patterns
- Distribution of data, or the “shape” of your data

Query access patterns directly point to data that is actually being summarized and how often this summarization occurs. If the summarization occurs infrequently and the query has little importance, then this would not point to a situation where a aggregate table would provide much value.

TIP: If a table has fewer than a half million rows, you might try indexing the table before creating one or more aggregates.

Once you have identified a set of candidates that promise value, you can evaluate each according to how much it *compresses* the data. For example, an aggregate table that allows a query to read one row that sums one hundred rows of detail data, would provide great value. On the other hand, a aggregate table that saves the query from reading three rows would provide little.

You can quickly calculate the compression factor using the Siebel Analytics Administrative Tool's Update Row Count function which is illustrated in [Figure 4](#).

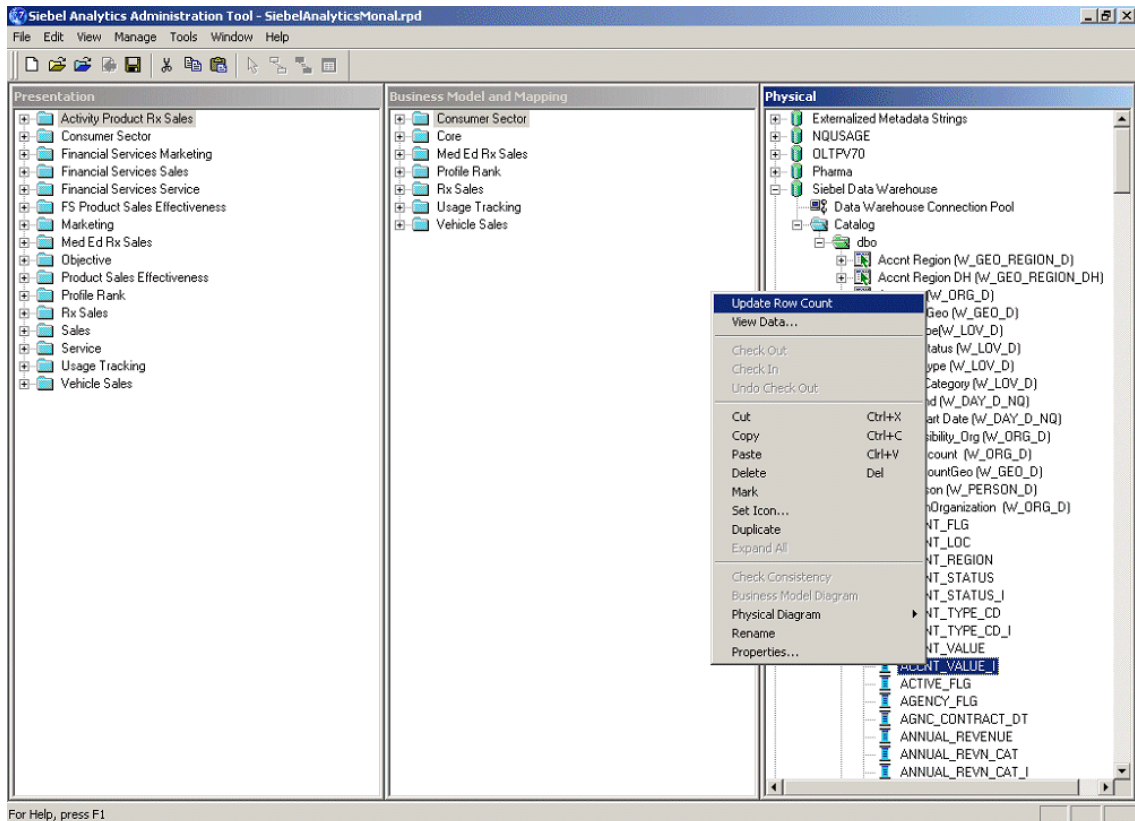


Figure 4. Update Row Count

The degree to which an aggregate table compresses detail data depends on the distribution of data. For more information on how to characterize the density and sparsity of these distributions and aggregate tables, see Chapter 14 of Ralph Kimball's book, *The Data Warehouse Lifecycle Toolkit* (Wiley, 1998). Identifying and qualifying candidates for aggregate tables is a matter of studying the query workload and determining the savings. In many cases, it is useful to simply experiment because the actual benefits are sometimes difficult to quantify with a pencil and paper.

Basic Guidelines

When you build aggregate tables, do not combine multiple-levels of aggregation. Instead, put each level of summarization in its own table. This simplifies writing queries and maintaining the aggregate tables.

When you create a aggregate table that spans dimensions and contains a large number of rows, create a star schema consisting of the aggregate table and corresponding dimensions. In this way, you can take advantage of the star schema structure and its usual indexing strategies.

For example, suppose you create a aggregate table that summarizes weekly product sales by city. The star schema would include a single aggregate table and three “shrunk” dimension tables:

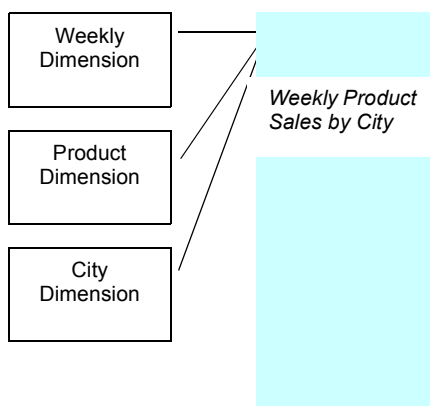


Figure 5. Aggregate Table with Dimensions

Each dimension represents a summarization of a detail dimension. For example, the Weekly Dimension table is a shrunk dimension of a more detailed Daily dimension table as [Figure 5](#) shows.

To further improve query performance, you can index the dimension and fact tables depending on the characteristics of their columns.

Too Many Aggregate Tables

Beware of too many aggregate tables. As the workload changes, aggregate tables like indexes fall into disuse. When this occurs, you should remove the aggregate tables. Similarly, should the data distribution shift so that the aggregate table fails to compress data adequately, you should remove the table.

Storage Space

Aggregate tables are effective and they do occupy physical storage space.

Adding and Managing Aggregate Tables

To add a aggregate table to the data warehouse, you must create the table itself and then develop a routine to load and maintain the table. Siebel Systems suggests that you develop a set of Informatica mappings to load and maintain aggregate tables.

To use this method, you must develop the mappings that load and maintain the aggregate tables and you must include entries in the Siebel metadata so that the Siebel Analytics server can rewrite queries to use the aggregate table rather than the detail tables whenever possible.

Names

You should use the standard naming conventions for aggregate tables so their names match those of other tables in the data warehouse.

Siebel Data Warehouse names aggregate tables as “W_XXX_A” where “XXX” references the star schema. This usage is consistent with the usage of names for fact and dimension tables.

Manage Aggregate Tables

You can manage loading and updating aggregate tables by developing Informatica mappings that extract data from fact tables, transform the data, and load it into the aggregate tables. You can develop mappings using the SQL Override feature or the Aggregator Transformation.

SQL Override

Create mappings to extract data from the fact tables, use SQL override to select the data, and use the Expression calculator to obtain summarized values as the server reads individual rows. The mapping then stores the summarized values in the aggregate table.

This approach can be implemented when you can not sort the data before it is routed to the Informatica server. This avoids memory cache problems with Aggregator transformations.

TIP: The SQL Override approach is the recommend method.

Aggregator Transformation

Create mappings to extract data from the fact tables and the Aggregator transformation to group and summarize the input data. This approach is more flexible because you can use conditional clauses to filter records. The output from the mapping is routed to the aggregate table.

When you use this method, you need to use presorted input data. Otherwise, the server demand for memory results in poor and usually unacceptable performance.

Packaged Aggregate Tables

Siebel eBusiness Data Warehouse includes several packaged aggregate tables to speed-up query performance. Aggregate tables are precomputed answer sets which are one of the most effective means to improve query performance. Siebel Analytics can substitute in place of detail tables to improve query performance. The server makes this substitution by rewriting the query, an operation that is transparent to the user. Aggregate tables are similar to indexes: the server, not the user, determines whether to use the aggregate table, and the decision rests on the expected performance improvement.

Subset Large Physical Dimension Tables

Often, a family of queries references only a well-defined subset of a large physical dimension. Suppose a family of queries references only competitor accounts in a large Accounts dimension table. To improve the performance of these queries, create a CompetitorAccounts dimension table and direct this family of queries to this smaller min-dimension table rather than to the larger dimension table.

You can accomplish this readily by identifying the subsets, creating the table, and including the subsets in the Business Model.

Subset Dimension Tables

The Siebel eBusiness Data Warehouse contains several large dimension tables that include well-defined subsets.

For example, the Person dimension (W_PERSON_D) includes contacts as well as employees. Similarly, the Organization dimension (W_ORG_D) includes accounts, partner accounts, competitor account, owner organization, and so forth.

These tables are shipped with the Data Warehouse and are mapped into the Siebel Analytics metadata as well as standard reports.

Creating a physical subset dimension makes sense only if you can identify a family of queries that reference only the rows in the subset. Otherwise, you incur the expense of additional physical storage, additional update management, overhead in the metadata and processing of your Business Model but receive no benefits.

Example

Suppose you have identified a set of frequently executed queries that reference only employees in a 2 million row Person dimension table. Of these 2 million rows, only 5000 represent employees, the remainder represent contacts.

To improve the performance of this family of queries, create a physical Employee table and direct queries that reference only Employees to the smaller table. This strategy makes sense because the query can scan the smaller table much faster than it can the larger one.

The W_EMPLOYEE_D table is already mapped into the Siebel metadata, and you can use it as an example for any custom subset dimension you might create. For a list of subset tables shipped with Siebel Analytics, see *Siebel eBusiness Data Warehouse Data Model Reference* guide.

To create and map a subset dimension

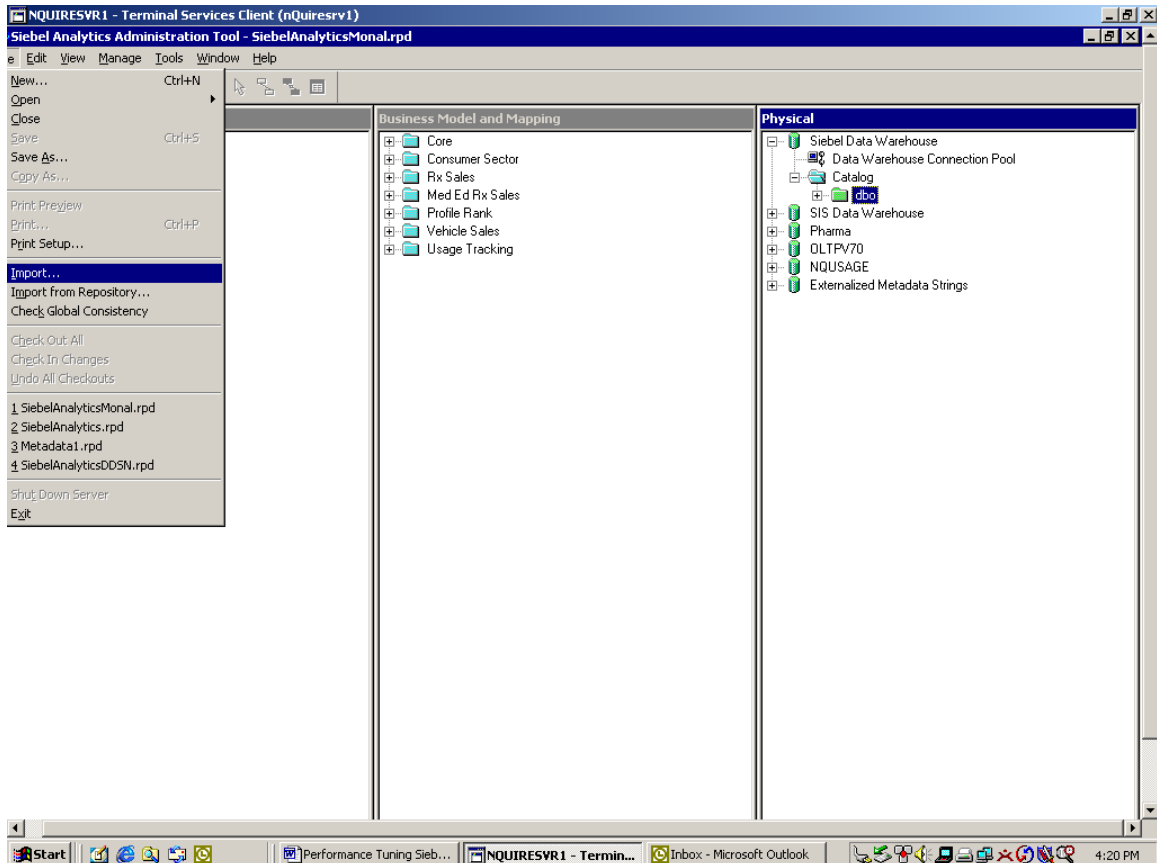
- 1** Create a W_EMPLOYEE_D table which is a subset of the W_PERSON_D table (rows which have 'Y' in their EMP_FLG column).

```
CREATE TABLE SIEBEL.W_EMPLOYEE_D
AS (select *
    from "W_PERSON_D"
    where EMP_FLG = 'Y');
```

- 2** Create indexes that can improve access performance on the W_EMPLOYEE table. See [Review Configuration Parameters on page 64](#).
- 3** Open the Analytics repository using the Siebel Analytics Administrator tool.

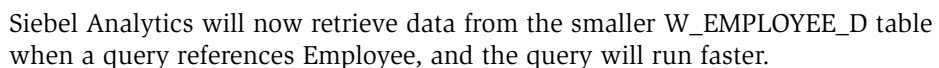
- 4 Import the new table from the Siebel Data Warehouse into the Physical layer.

This step modifies the Siebel Analytics metadata so that it points to the newly created W_EMPLOYEE_D table as illustrated below.



NOTE: You will need to refresh the contents of the W_EMPLOYEE table anytime the contents of the W_PERSON_D table are refreshed or loaded.

- See illustration below.



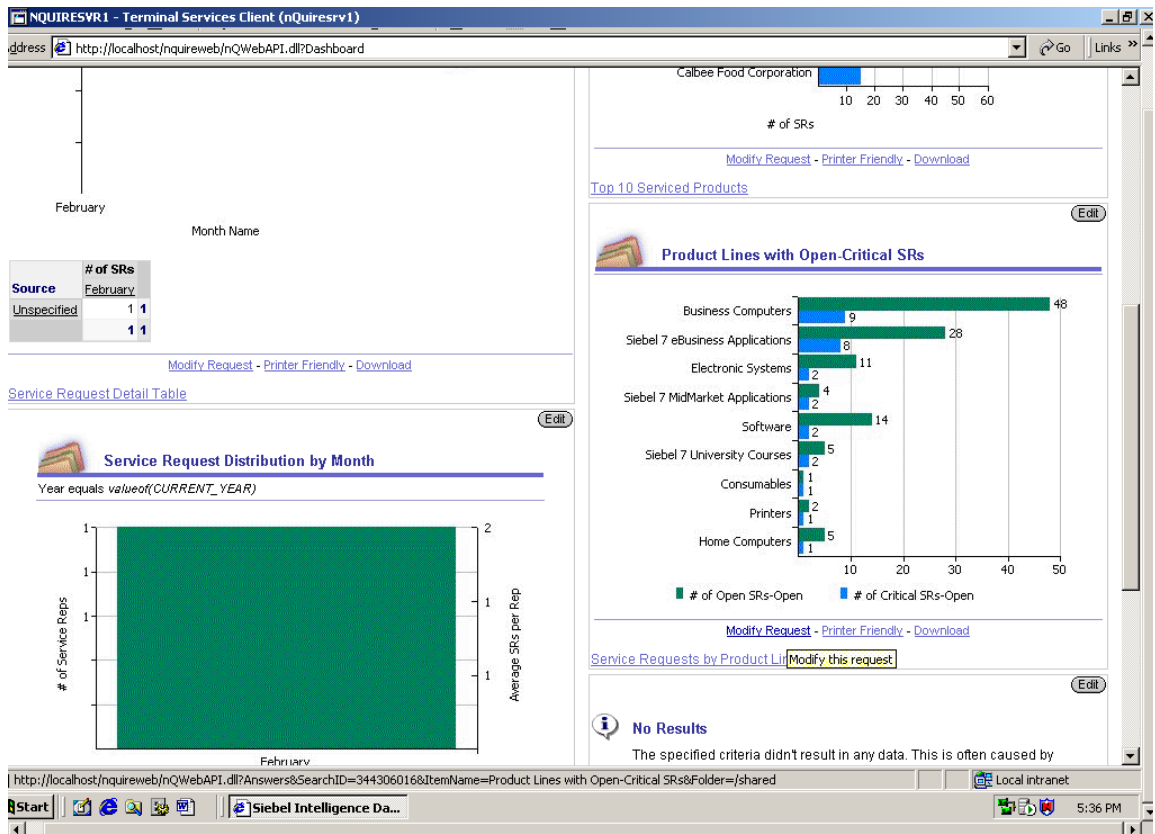
Remove Columns Not Required by a Report

Look at frequently run Dashboard reports and make sure that only the columns that are required by your reports occur are selected. *Remove any superfluous columns.* This simple action can improve query performance for those cases where a large number of columns not required by the report occur in the query select list.

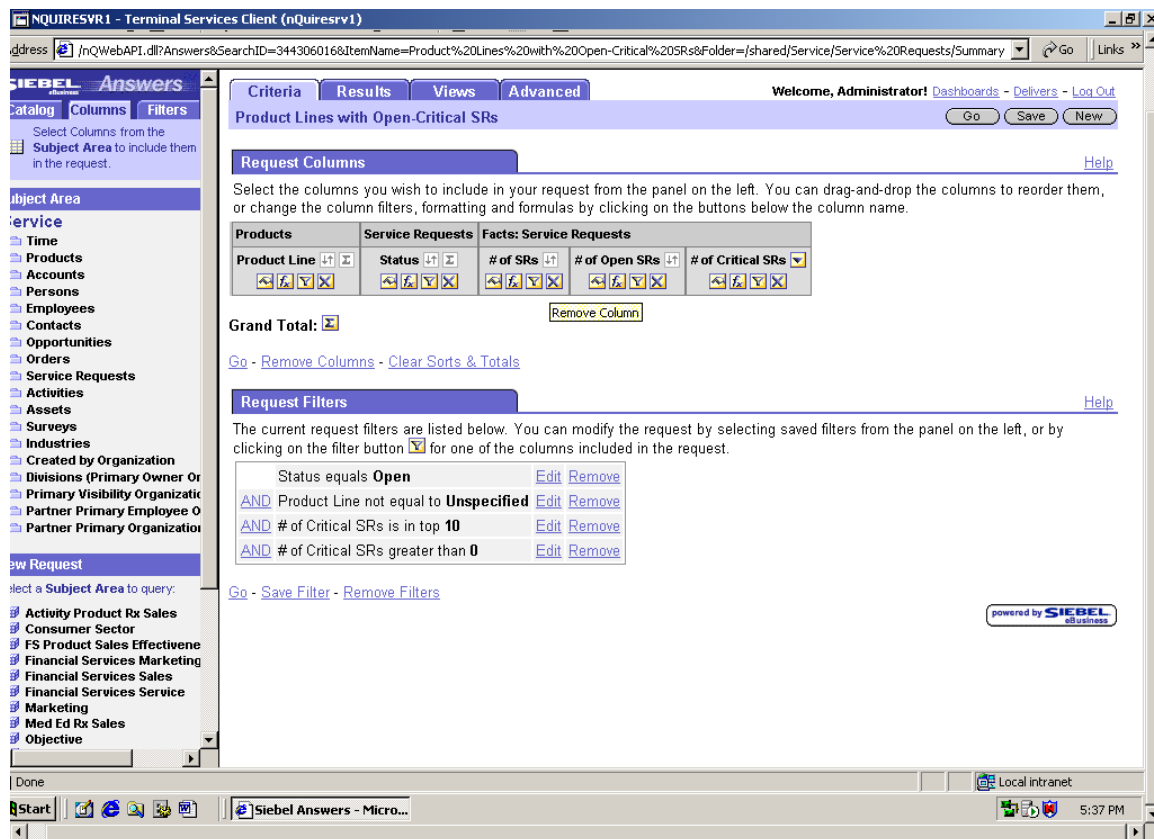
Example

This example shows you how to remove columns that are not required by a report.

- 1 Suppose you have a report named “Product Lines with Open-Critical SRs” similar to the one shown below.

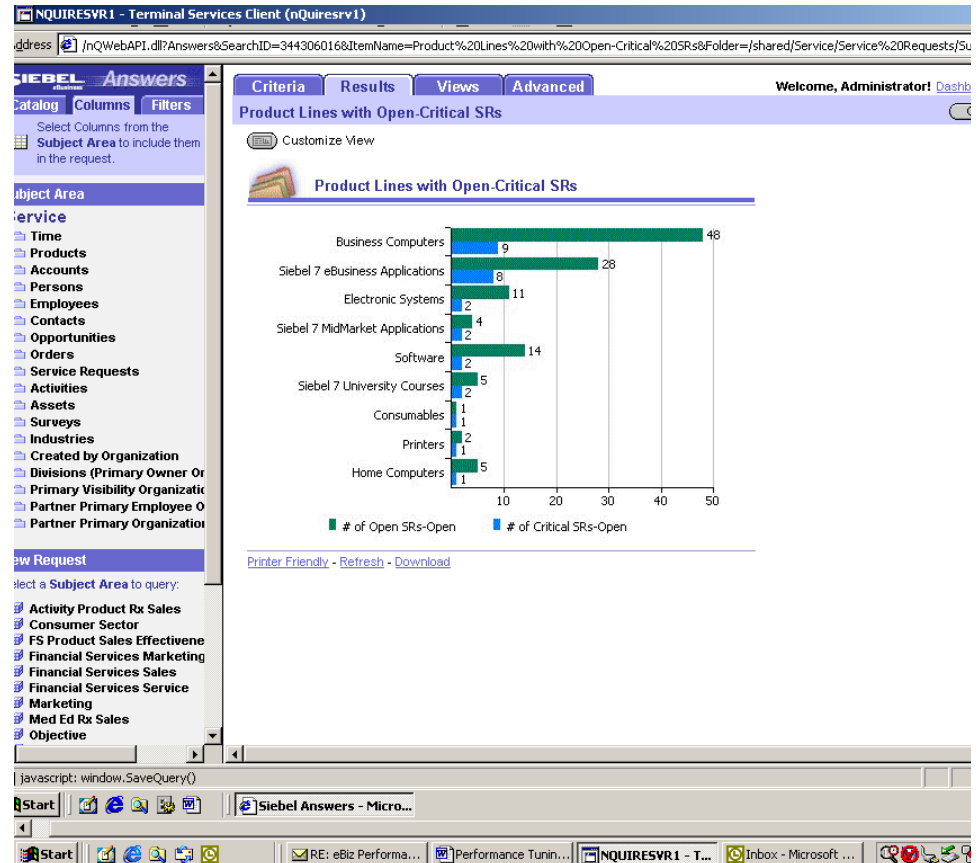


2 To see the criteria for selecting columns, click “Modify Request.”



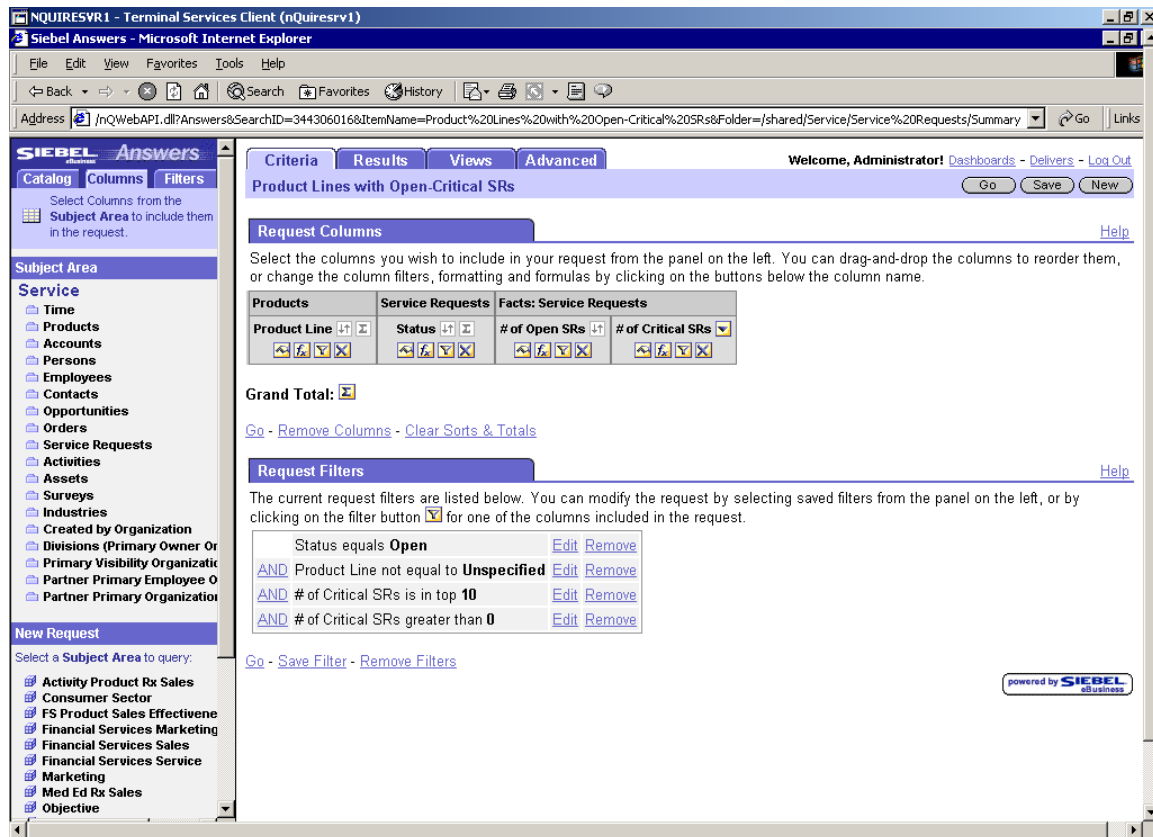
3 The “# of SRs” fact is selected although it is not required in the “Product Lines with Open-Critical SRs” report. See the figure above.

- 4 Remove this column from the selection criteria.



- 5 Click the Results tab, as shown in the figure above, to verify whether the report remained the same.

6 Save your changes.



7 Continue to search for other columns that force the database server to retrieve data that is not required by the report. See the figure above.

Review Configuration Parameters

This technique applies to both the application software, Siebel Analytics software, and database server software.

Siebel Analytics Configuration Parameters

Consider changing the following parameters in the `NQSConfig.ini` configuration file for the Siebel Analytics server:

- Query Caching
- Case-Sensitive Character Comparison
- Driving Table per Siebel Analytics example

Query Caching

This was the first technique described in this guide. For the details, see [Enable Query Caching on page 19](#).

Case-Sensitive Character Comparison

The parameter that determines whether the Siebel Analytics Server differentiates between lower and upper case characters can impact query performance. When the `CASE_SENSITIVE_CHARACTER_COMPARISON` parameter in `NQSConfig.ini` file is set to ON, the server does not retrieve values for the result set that fail to exactly by case. When set to ON, the server can process the query more efficiently.

For instructions that describe how to set this parameter to ON, see *Siebel Analytics Installation and Configuration Guide*.

Driving Table

- Table Joins (driving table, see *Siebel Analytics Server Administration Guide*)

Repository Configuration

Make sure to select the correct database type in your repository configuration to use the optimal database specific feature for underlying queries.

This chapter shows you how to improve the performance of the database servers accessed by Siebel Analytics. The chapter emphasizes how to select, evaluate, and manage indexes for database servers, although database configuration and the capacity of the host system are also important factors to consider.

The chapter covers these topics:

- [Overview of Database Indexing on page 38](#)
- [Index Selection on page 39](#)
 - [More Than One Kind of Index on page 38](#)
 - [B-Tree Indexes on page 39](#)
 - [Bitmap Indexes on page 48](#)
 - [Vendor Index Evaluation Tools on page 51](#)
 - [Determinants of Index Efficiency on page 52](#)
- [Periodic Tuning Tasks on page 55](#)
 - [Periodic Index Evaluation on page 55](#)
 - [Refresh Optimizer Statistics on page 57](#)
 - [Reorganize Indexes on page 60](#)

NOTE: The examples in this chapter are based on an Oracle database and illustrate performance tuning techniques. In many cases, you can adapt the technique to DB2 and SQL Server databases.

Overview of Database Indexing

Indexes are database objects *designed* to improve query performance by shortening access paths to data and accelerating join operations. Indexes improve performance and are transparent to users; the server, not the user, determines which index to use, although some optimizers allow users to include hints.

Indexes can improve query performance by reducing access time and accelerating join operations. Nevertheless, they occupy storage space, they must be maintained, and they can interfere with database update and load operations. Consequently, indexes should be selected with care, and their benefits balanced against their costs.

More Than One Kind of Index

All database vendors support B-Tree indexes, and a few support bitmap indexes.

A B-Tree index is an *ordered* set of entries. Each entry contains a search-key value and a pointer to a row in the table that contains the value. The server can attack the ordered structure of B-Trees predictably and efficiently, and a B-Tree index is smaller than the underlying table. B-Tree indexes improve performance most when a column contains mostly distinct values (index cardinality is low). They are also used to enforce uniqueness.

A bitmap index is also a set of entries. Each entry contains a search-key value and a bitmap (or pointer to a bitmap). Each bit corresponds with a row in the table, and the presence of a bit indicates the row contains the search-key. Bitmap indexes are often useful in many situations where B-Tree indexes are not. Namely, for columns which have a large number of duplicate values (index cardinality is high). For example, a Size column that contains only five distinct values (tiny, small, medium, large, grand) is a good candidate. Oracle Corporation supports bitmap indexes.

Database Optimizers

All the database servers supported by Siebel Analytics have optimizers that generate a set of execution plans and then select the most optimal plan. These optimizers can generate plans based on rules and on processing costs. The cost-based optimizers rely on data distribution statistics stored in the database catalog which must be generated and periodically refreshed by the administrator, but they generally outperform rules-based optimizers in data warehousing environments.

Index Selection

Indexes should be selected on the basis of how much they improve query performance versus their cost in terms of physical storage, maintenance, and interference with the ETL load process. This section describes a basic method for index selection, guidelines for individual indexes, and a few cases where indexes would not be appropriate.

B-Tree Indexes

Generally speaking, in an online transaction processing environment, a B-tree is most effective when it is highly selective. When this is the case, the index is said to have “high selectivity” because a low percentage of rows in the table have the same index key value.

General Guidelines

With high selectivity in mind, evaluate creating B-tree indexes on columns that:

- Occur frequently in WHERE clauses
- Often used to join tables (include aggregate tables)
- Occur in ORDER BY clauses (the index can facilitate ordering)
- Occur in a foreign key reference constraint
- Used to enforce PRIMARY KEY and UNIQUENESS constraints

You can also look at your query workload and identify families of queries that include tight table constraints on tables (point, multi-point, and range queries).

When you have star schemas, both DB2 and Oracle database servers can exploit multi-column B-Tree indexes to accelerate join processing when they are created over the foreign key reference columns of “fact” tables. DB2 and Oracle can also accelerate star join operations when *single* column indexes are created on the fact table’s foreign key reference columns. In the Oracle case, these indexes are specialized Bitmap indexes which are used by the server’s star transformation algorithm. See [“Bitmap Indexes” on page 48](#).

Where B-Trees Should Not Be Created

Several situations are worth noting where you *should not* create B-Tree indexes on columns. These cases include columns which:

- Have only a few distinct values in their domains. For example, a Type column that has only four distinct values (A, B, C, and D). The index would be said to have “low selectivity.” If you have an Oracle database, then these columns of low selectivity are ideal candidates for Bitmap indexes.
- Occur in WHERE clauses but within functions other than MIN or MAX.

Indexes in these cases waste space and slow down the load process.

Siebel Recommended Methodology

You can follow the methodology described in the following example procedure to create new indexes to speed up a slow running report. The slow running report used in this example is “Abandoned Carts Detail.”

To analyze a slow running report

- 1** Log into Siebel Analytics and run the Abandoned Carts Detail report.
- 2** Click the Results tab.
- 3** After waiting for unacceptably long period of time, cancel the query.
- 4** Trace the SQL using the Siebel Analytics Admin Mode.
 - a** Return to the dashboard.
 - b** Click the Admin hyperlink.

Analytics returns the Siebel Analytics Administration page.

- c Click the Manage Sessions hyperlink.

Analytics returns the Sessions page shown in the figure below.

View session information below. Finished

User ID	Host Address	Session ID	Browser Info	Logged On	Last Access
Administrator	172.20.108.126	172.20.108.126.b48.122706ea.6	Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)	7/2/2002 2:41:02 AM	7/2/2002 2:47:34 AM
Administrator	172.20.108.126	172.20.108.126.1175.122e692c.7	Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)	7/2/2002 2:49:04 AM	7/2/2002 2:49:26 AM
	BZHAO	172.20.10.86.7436.faf9f93.2	Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 4.0)	7/1/2002 3:01:46 PM	7/1/2002 5:42:34 PM
	BZHAO	172.20.10.86.75ef.faf908dd.3	Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 4.0)	7/1/2002 3:04:05 PM	7/1/2002 5:41:47 PM
	smt6000i092	172.20.238.32.740f.faf6ca67.1	Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0; T312461; .NET CLR 1.0.3705)	7/1/2002 3:01:34 PM	7/1/2002 3:01:33 PM
Administrator	FMANTFEL	172.20.40.46.428a.1136e6a0.5	Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)	7/1/2002 10:18:52 PM	7/2/2002 12:38:16 AM
	BZHAO	172.20.10.86.6f7d.103a7264.4	Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 4.0)	7/1/2002 6:07:53 PM	7/1/2002 6:58:31 PM

Cursor Cache

[Cancel Running Requests](#) - [Close All Cursors](#)

ID	User	Refs	Status	Time	Action	Last Accessed	Statement	Records
648	Administrator	1	Finished	2m 52s	Close View Log	7/2/2002 2:47:29 AM	SELECT "- Quote Header".Type, "- Profile"."Account Name", "- Quote Header"."Full Name", "- Quote Header"."Quote #", "- Created Date"."Created Date", "- Quote Header Facts"."Total Quote Revenue" FROM Quotes WHERE (1=1) ORDER BY 1, 2	4

Finished

Done Local intranet

- 5 Look in the Cursor Cache window for your session.

In this case, there is only one session in the window and it ran 2 minutes and 52 seconds.

6 Click the View Log hyperlink.

Analytics returns another window that contains the SQL generated for the session, the logical SQL inside the Analytics server and the SQL sent to the database, which is shown in the following sections.

```
+++Administrator:30000:30002:----2002/07/01 22:20:42

#####
----- SQL Request:
SELECT
  "Marketing Campaign"."Program Name",
  "- Profile"."Account Name",
  "- Order Facts"."Total Order Revenue",
  "- Opportunity Facts"."Closed Margin"
FROM Customers
WHERE
  (TOPN{"- Order Facts"."Total Order Revenue",10) <= 10)

+++Administrator:30000:30002:----2002/07/01 22:20:42

----- General Query Info:
Repository: Star, Subject Area: Core, Presentation: Customers

+++Administrator:30000:30002:----2002/07/01 22:20:42

----- Logical Request (before navigation):

RqList
  Program.Program Name as c1 GB,
  Account.Account Name as c2 GB,
  Total Order Revenue:[DAggr(FACTS.Total Order Revenue by [Account.Account Name,
  Closed Margin:[DAggr(FACTS.Closed Margin by [Account.Account Name, Program.Prog
```

Note the time between each section marked by lines such as:

```
+++Administrator:30000:30002:----2002/07/01 22:20:42
```

In the screen shot below, note that the mean time between these lines is just short of 3 minutes.

```
7 05 01
OrderBy: c1 asc, c2 asc [for database 0:0,0]

+++Administrator:40000:40001:----2002/07/02 02:56:38

----- Sending query to database named Siebel Data Warehouse (id: <3
select T1035949.TYPE as c1,
       T1034992.NAME as c2,
       concat(T1042934.FST_NAME, concat(' ', T1042934.LAST_NAME)) as c3,
       T1035949.QUOTE_NUM as c4,
       T1038921.DAY_DT as c5,
       sum(T1035847.QUOTED_NET_PRI * 1 * T1035847.QTY_REQUESTED * T1035847.INCL_CALC_
from
  W_PERSON_D T1042934,

  W_DAY_D T1038921,

  W_QUOTE_D T1035949,

  W_ORG_D T1034992,

  W_QUOTEITEM_F T1035847
where T1035847.CREATED_DT_WID = T1038921.ROW_WID and T1035847.CONTACT_WID = T104293
group by T1034992.NAME, T1035949.QUOTE_NUM, T1035949.TYPE, T1038921.DAY_DT, T103892
order by c1, c2, c3, c4, c5

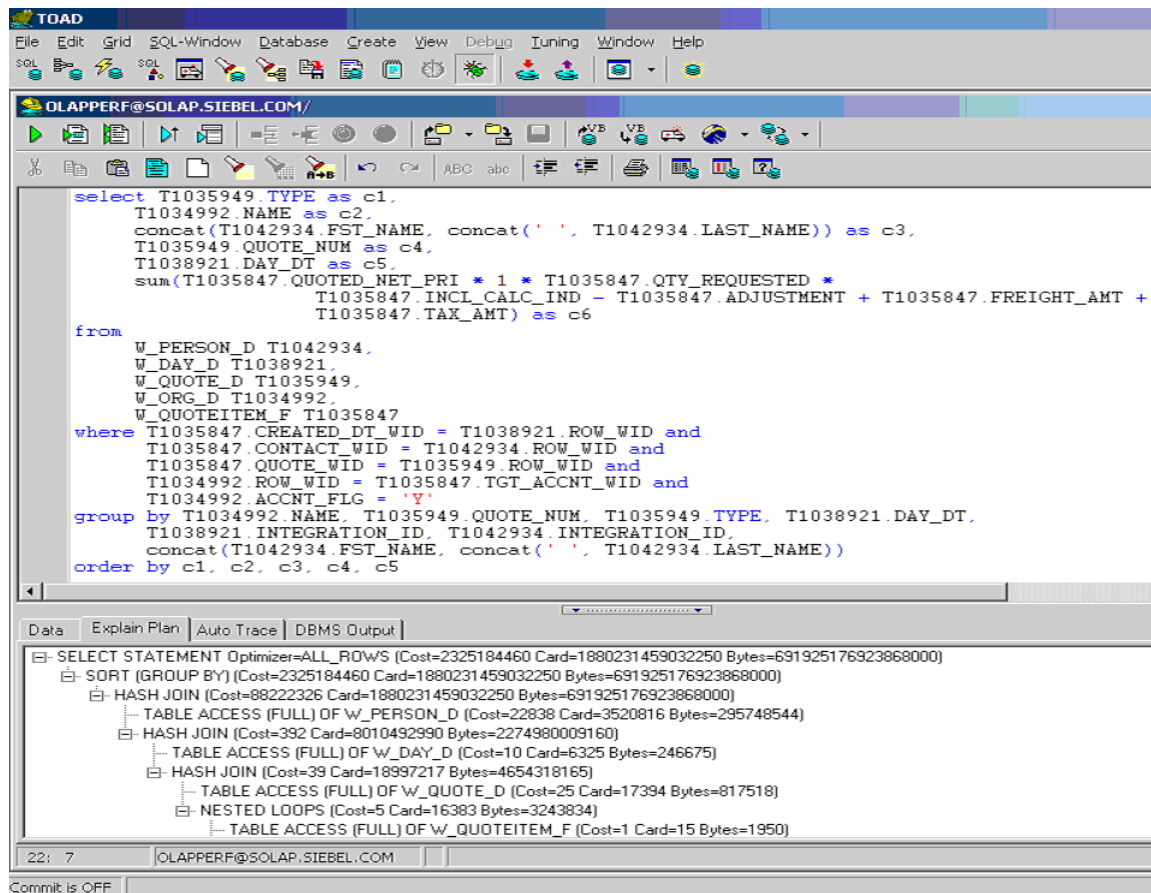
+++Administrator:40000:40001:----2002/07/02 02:59:19

----- Query Status: Successful Completion
```

- 7 Run the SQL sent to the database using the EXPLAIN PLAN command.

You can generate the execution plan for a query in a variety of ways such as SQL*Plus, Oracle Enterprise Manager, or TOAD (Tools for Oracle Application Development). This example copies and pastes the SQL into TOAD.

The figure below shows the generated SQL in a TOAD window.



The SQL text including the explain plan output was copied and pasted on the next page.

```

select          T1035949.TYPE as c1,
                T1034992.NAME as c2,
                concat(T1042934.FST_NAME,
                concat(' ', T1042934.LAST_NAME)) as c3,
                T1035949.QUOTE_NUM as c4
                T1038921.DAY_DT as c5,
                sum(T1035847.QUOTED_NET_PRI * 1 * T1035847.QTY_REQUESTED
                * T1035847.INCL_CALC_IND - T1035847.ADJUSTMENT
                + T1035847.FREIGHT_AMT + T1035847.TAX_AMT) as c6
from            W_PERSON_D T1042934, W_DAY_D T1038921, W_QUOTE_D T1035949, W_ORG_D
                T1034992, W_QUOTEITEM_F T1035847
where           T1035847.CREATED_DT_WID = T1038921.ROW_WID
and             T1035847.CONTACT_WID = T1042934.ROW_WID
and             T1035847.QUOTE_WID = T1035949.ROW_WID
and             T1034992.ROW_WID = T1035847.TGT_ACCNT_WID
and             T1034992.ACCNT_FLG = 'Y'
group by        T1034992.NAME, T1035949.QUOTE_NUM, T1035949.TYPE,
                T1038921.DAY_DT, T1038921.INTEGRATION_ID, T1042934.INTEGRATION_ID,
                concat(T1042934.FST_NAME, concat(' ',T1042934.LAST_NAME))
order by        c1, c2, c3, c4, c5

SELECT          STATEMENT Optimizer=ALL_ROWS (Cost=2325184460
Card=1880231459032250
                Bytes=691925176923868000)
SORT (GROUP BY) (Cost=2325184460 Card=1880231459032250
                Bytes=691925176923868000)
HASH JOIN (Cost=88222326 Card=1880231459032250 Bytes=691925176923868000)
TABLE ACCESS (FULL) OF W_PERSON_D (Cost=22838 Card=3520816
                Bytes=295748544)
HASH JOIN (Cost=392 Card=8010492990 Bytes=2274980009160)
TABLE ACCESS (FULL) OF W_DAY_D (Cost=10 Card=6325 Bytes=246675)
HASH JOIN (Cost=39 Card=18997217 Bytes=4654318165)
TABLE ACCESS (FULL) OF W_QUOTE_D (Cost=25 Card=17394 Bytes=817518)
NESTED LOOPS (Cost=5 Card=16383 Bytes=3243834)
TABLE ACCESS (FULL) OF W_QUOTEITEM_F (Cost=1
Card=15 Bytes=1950)
TABLE ACCESS (BY INDEX ROWID) OF W_ORG_D (Cost=1 Card=16383
                Bytes=1114044)
INDEX (UNIQUE SCAN) OF W_ORG_D_P1 (UNIQUE)

```

The processing costs for each operation within the query is a relative estimate which highlights the costliest operations within the query. These are the operations that you need to analyze closely.

The most expensive table to access within this plan is to W_PERSON_D. This also happens to be the largest table in the test database (4 GB). If the hash area of the columns from this table can be reduced, this will reduce access time.

One method to reduce this access time is to index the columns in the table so that the query scans the indexes instead of the table.

- 8 Locate all occurrences of the table's columns within the query.

The columns FST_NAME and LAST_NAME occur in the query's select list and the column ROW_WID occurs in its where clause.

- 9 Create and analyze an index that covers these columns.

```
Create index w_person_d_x7
      on w_person_d (row_wid, fst_name, last_name)
      tablespace idx nologging pctfree 0 ;
```

```
Analyze index w_person_d_x7 compute statistics ;
```

TIP: The columns referenced in the WHERE clause should be the leading columns for a particular index. If the columns were created in another order, say (LAST_NAME, FST_NAME, ROW_WID), the optimizer would not generate a query that uses the index.

- 10 Generate the explain plan for the query and compare query plans.

```
SELECT STATEMENT Optimizer=ALL_ROWS (Cost=2325173878 Card=1880231459032250
      Bytes=691925176923868000)
  SORT* (ORDER BY) (Cost=2325173878 Card=1880231459032250
      Bytes=691925176923868000)
    SORT* (GROUP BY) (Cost=2325173878 Card=1880231459032250
      Bytes=691925176923868000)
      SORT* (GROUP BY) (Cost=2325173878 Card=1880231459032250
      Bytes=691925176923868000)
        HASH JOIN* (Cost=88211744 Card=1880231459032250
      Bytes=691925176923868000)
          VIEW* OF index$_join$_001 (Cost=12256 Card=3520816
      Bytes=295748544)
            HASH JOIN* (Cost=88211744 Card=1880231459032250
      Bytes=691925176923868000)
              INDEX* (FAST FULL SCAN) OF W_PERSON_D_X7 (NON-UNIQUE)
      (Cost=1 Card=3520816 Bytes=295748544)
                INDEX* (FAST FULL SCAN) OF W_PERSON_D_U1 (UNIQUE)
      (Cost=1 Card=3520816 Bytes=295748544)
                  HASH JOIN* (Cost=392 Card=8010492990 Bytes=2274980009160)
      TABLE ACCESS (FULL) OF W_DAY_D (Cost=10 Card=6325 Bytes=246675)
        HASH JOIN (Cost=39 Card=18997217 Bytes=4654318165)
```

```

TABLE ACCESS (FULL) OF W_QUOTE_D (Cost=25 Card=17394
      Bytes=817518)
NESTED LOOPS (Cost=5 Card=16383 Bytes=3243834)
  TABLE ACCESS (FULL) OF W_QUOTEITEM_F (Cost=1 Card=15
        Bytes=1950)
  TABLE ACCESS (BY INDEX ROWID) OF W_ORG_D (Cost=1
Card=16383
        Bytes=1114044)
    INDEX (UNIQUE SCAN) OF W_ORG_D_P1 (UNIQUE)

```

The plan shows that the full table scan of the W_PERSON_D was replaced with a hash join of two indexed lookups.

11 Verify the performance improvement.

Copy the query from the explain plan, paste it into TOAD, and then execute the query with TOAD. The elapsed time reported by TOAD is now 48 seconds. This is roughly one-fourth the original time.

12 Flush the query cache from the Analytics server.

Navigate to the Admin View - Manage Session, and click the Close All Cursors hyperlink.

13 Run the Analytics report again.

14 Return to Admin mode and verify that the report ran faster (47 seconds).

Thus, a multicolumn B-Tree index on the W_PERSON_D dimension table improves query performance roughly 300 percent.

Bitmap Indexes

Oracle supports bitmap indexes which are often useful in many situations where B-tree indexes are not optimal. Namely, for columns which have a large number of duplicate values. For example, a Size column that contains only five distinct values (tiny, small, medium, large, grand).

Oracle's "star transformation algorithm" uses bitmap indexes to join a fact table to its dimensions when they exist on the foreign key constraint columns in a fact table. Actually, this algorithm is robust enough to use a combination of bitmap and B-tree indexes. IBM DB2 takes advantage of this technology in selected situations by dynamically building bitmaps from a single-column B-tree to join tables. Unlike Oracle, however, these are highly specialized cases.

Entries in a bitmap index consist of a search key value and a bitmap which describes rows that contain the search key value. Each bit in the map corresponds with a row in the table, and a bit on signals a row that contains the value in the search key. key value.

Candidates for Bitmap Indexes

Bitmap indexes are most advantageous whenever the cardinality of the index is less than one percent, or lowly-selective. This criterion is nearly the opposite of the guideline for B-Tree indexes.

Look for cases where:

- A query constrains multiple columns which have few distinct values in their domains (large number of duplicate values).
- A large number of rows satisfy the constraints on these columns.
- Bitmap indexes have been created on some or all of these columns.
- The referenced table contains a large number of rows.

Given this kind of scenario, the server can evaluate the constraints by ANDing the bitmaps and potentially eliminate a large number of rows without ever accessing a row in the table.

The Oracle database server can also generate query execution plans to join a fact table to its dimensions using its star transformation algorithm when bitmap indexes exist on the fact table foreign key reference columns. When this execution plan is the least costly, the server joins the tables using the bitmap indexes.

CAUTION: Bitmap indexes should be used only for static tables and are not suited for highly volatile tables in online transaction processing systems.

Also, the Oracle optimizer generates query plans only when the cost-based optimizer has been enabled.

To create a bitmap index, use the “bitmap” keyword:

```
create bitmap index w_srvreq_d_m2 on w_srvreq(area_I)
      nologging tablespace idx pctfree 0 ;
```

You can analyze a bitmap index just like you do any other index:

```
analyze index w_srvreq_d_m2 compute statistics ;
```

Example Bitmap Index

Bitmap indexes are useful for low-selectivity cases. For example, consider the AREA_I column on the W_SRVREQ_D table of a sample database. A simple query against this query shows that this column has a very small domain (few distinct values).

NOTE: Be sure that you understand your data shape when evaluating the need for indexes. Also be aware that this example is heavily dependent on a particular data shape.

```
SQL> select area_i, count (*) cnt from w_srvreq_d group by area_i;
```

AREA_I	CNT
-----	-----
3rd Party Software	171,456
Abnormal Usage	77,988
CD-ROM	171,717
Disk Drive	115,379
Ethernet Card	59,076

Hard Drive	59,328
Installation	1,850,376
Internet Registration Request	199,272
Memory	258,352
Network	199,850
Operating System	168,396
Performance	140,895
Unspecified	1,368,797
Upgrade	278,136
Usage	199,178
User Interface	276,738

This table contains only 16 distinct values that occur in 5,594,934 rows. Clearly, this is not a candidate for a B-Tree index. Such an index would more likely degrade rather than improve query performance. This column is, nevertheless, an excellent candidate for a bitmap index. With a cardinality of 1 / 350,000 (0.00028%), it is nearly perfect for a bitmap index.

TIP: Bitmap indexes are most effective when a query constrains a set of columns that have bitmap indexes or a combination of single-column bitmap and B-Tree indexes. When these conditions are ANDed, then the server can AND the bitmaps rather than data fetched from individual rows.

When Bitmap Indexes Should Not Be Created

Bitmap indexes should not be created in the following cases:

- A column with a UNIQUE constraint
- A column that mostly contains distinct values
- Where the table is frequently updated or loaded with new data

Indexes incur several costs and you need to balance their cost against their actual benefits. Use a performance monitor to evaluate their actual benefits.

Vendor Index Evaluation Tools

The database vendors provide tools that can assist you in choosing an optimal set of indexes.

Database	Tool	Description
DB2	Index Advisor	For additional information see the <i>DB2 Administration Guide V7.1, Volume 3: Performance</i> .
Oracle 8i and 9i	ALTER INDEX ... MONITORING USAGE Oracle Tuning Pack	For additional information, see the <i>Oracle9i Database Performance Guide and Reference</i> .
SQL Server 2000		For additional information, see the Microsoft white paper <i>RDBMS Performance Tuning Guide for Data Warehousing</i> .

Determinants of Index Efficiency

The distribution of data within a table's column largely determines the performance of indexes. Earlier sections have already pointed out that if the column contains mostly distinct values, then B-Tree indexes can be quite effective; otherwise, a bitmap index may perform better.

This general notion of selectivity needs to be moderated with a little perspective on how selectivity can be skewed. When this occurs, the index seems magical; sometimes it is very effective and other times it seems to get in the way.

Consider a million row table of contacts used by a medical insurance provider for a state dominated by one large city. Suppose that one borough dominates the large city and the distribution of zip codes throughout the state is not evenly distributed as illustrated in the table below.

Region	Contacts	Number of Zip Codes
LC Borough 1	400,000	1
LC Borough 2	125,000	1
LC Borough 3	125,000	1
LC Borough 4	125,000	1
LC Borough 5	125,000	1
Outside LC	100,000	495

If you create a B-Tree index in this situation, then the performance of queries generated by rule-based optimizers will vary by borough. The efficiency of a search for all contacts within a certain zip code would depend largely on the zip code, and the data itself is the determinant. This explains why a query that performs well in some situations is absolutely sluggish in others.

Specifically, a query that searches for 200 contacts in a suburb of a city outside of the Large City, will have an almost instantaneous response time. Oracle would invoke the index on an index range scan, and for each value, the server would go and fetch the corresponding row from the table. Once the dataset is built, Oracle would process the rest of the query.

What about a query against Borough 1 of the Large City? The Rules Based Optimizer in this case performs an index range scan, and iteratively fetches 400,000 records, *one at a time*. This is going to take a long time.

A solution in this case is to turn to cost-based optimizers which generate query execution plans based on statistics stored in the system catalog and match the granularity of the histograms they generate to your particular workload. These statistics describe the distribution of index key values, and the server can generate plans accordingly.

You can control the granularity of the histograms using options on the analyze statements. For example, this statement creates histograms with 75 buckets:

```
Analyze table w_person_d
      compute statistics for all indexed columns ;
```

To create a histogram on the LAST_NAME column with 200 buckets, you can use the following statement:

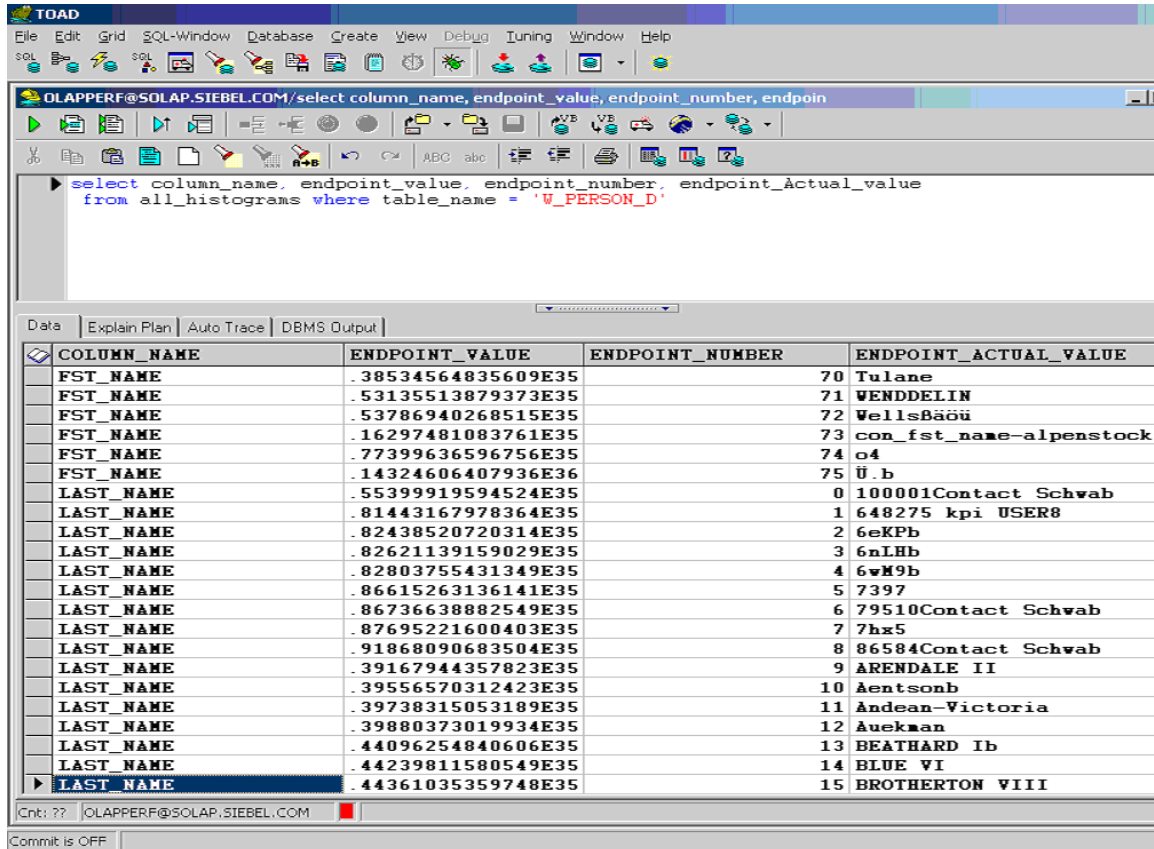
```
Analyze table w_person_d
      compute statistics for column LAST_NAME size 200 ;
```

To create a histogram on each of the LAST_NAME and FST_NAME columns with 250 buckets each, use the following statement:

```
Analyze table w_person_d
      compute statistics for columns
      LAST_NAME, FST_NAME size 250 ;
```

Remember to drop and recreate the histograms after each ETL load, because the histogram is just a representation of values that can become stale if not recomputed when the table is updated. Stale statistics can lead to suboptimal query plans.

After you create statistics, you can retrieve histograms generated for specific columns using an SQL statement.



The screenshot shows the TOAD application window with a SQL query executed. The query is: `select column_name, endpoint_value, endpoint_number, endpoint_Actual_value from all_histograms where table_name = 'W_PERSON_D'`. The results are displayed in a table with four columns: COLUMN_NAME, ENDPOINT_VALUE, ENDPOINT_NUMBER, and ENDPOINT_ACTUAL_VALUE. The results show histograms for various columns, including FST_NAME and LAST_NAME, with their respective endpoint values, numbers, and actual values.

COLUMN_NAME	ENDPOINT_VALUE	ENDPOINT_NUMBER	ENDPOINT_ACTUAL_VALUE
FST_NAME	.38534564835609E35	70	Tulane
FST_NAME	.53135513879373E35	71	WENDELIN
FST_NAME	.53786940268515E35	72	WellsBaou
FST_NAME	.16297481083761E35	73	con fst_name-alpenstock
FST_NAME	.77399636596756E35	74	o4
FST_NAME	.14324606407936E36	75	Ü.b
LAST_NAME	.55399919594524E35	0	100001Contact Schwab
LAST_NAME	.81443167978364E35	1	648275 kpi USER8
LAST_NAME	.82438520720314E35	2	6eKPb
LAST_NAME	.82621139159029E35	3	6nLHb
LAST_NAME	.82803755431349E35	4	6wM9b
LAST_NAME	.86615263136141E35	5	7397
LAST_NAME	.86736638882549E35	6	79510Contact Schwab
LAST_NAME	.87695221600403E35	7	7hx5
LAST_NAME	.91868090683504E35	8	86584Contact Schwab
LAST_NAME	.39167944357823E35	9	ARENDALE II
LAST_NAME	.39556570312423E35	10	Aentsonb
LAST_NAME	.39738315053189E35	11	Andean-Victoria
LAST_NAME	.39880373019934E35	12	Auekman
LAST_NAME	.44096254840606E35	13	BEATHARD Ib
LAST_NAME	.44239811580549E35	14	BLUE VI
LAST_NAME	.44361035359748E35	15	BROTHERTON VIII

You can see the various buckets in the figure above that cost-based optimizers use for a BETWEEN comparison to get the right histogram bucket and the distribution of values. This gives you some insight into the shape of your data.

Periodic Tuning Tasks

Tuning a database for query performance is an ongoing, periodic activity. As the tables in the underlying databases change during load and other update operations, the indexes themselves change. The changes to tables can be sufficient to render some indexes useless and provide opportunities to add new indexes. These changes also necessitate reorganizing indexes and refreshing their statistics.

This section describes these tasks:

- Periodic index evaluation
- Refresh optimizer statistics
- Reorganize indexes

These tasks should always be performed immediately after you complete an ETL load on one of the underlying databases.

Periodic Index Evaluation

After you complete an ETL load of an underlying database, you should always analyze and evaluate *all* database indexes to determine whether individual indexes remain effective. A load operation, or successive load operations, can change the distribution of the data enough to alter the effectiveness of an index.

To illustrate how this can happen, consider an example based on an index supplied with Siebel Analytics. Siebel Systems recommendation to use this index is based on expected usage patterns against laboratory data. In practice, the index improves performance for a large number of customers but fails to be effective for a few. The failure occurs when the laboratory data is not representative of a customer's data.

The following example uses the index W_REVN_F_F23, a B-Tree index supplied with Siebel Analytics. After loading your data into the database, you may find that this particular index is ineffective, depending on your data shape.

Preliminary Analysis

To evaluate the index, analyze the index and retrieve a few statistics from the system catalog table `all_indexes` using the following query.

```
SQL> analyze index w_revn_f_f23 compute statistics ;
Index analyzed.
Elapsed: 00:00:12.48
SQL> select num_rows, distinct_keys, avg_leaf_blocks_per_key
from all_indexes
where index_name = 'w_revn_f_f23'
num_rows      distinct_keys      avg_leaf_blocks_per_key
2789291        1                  4539
```

For this particular data, the B-Tree index is useless because there is exactly one index key. The same would be true of a bitmap index. You can learn a little more by looking at the contents of the column:

```
SQL> select pr_offer_wid, count (*) from w_revn_f
group by pr_offer_wid ;

PR_OFFER_WID  COUNT(*)
-----
0             2789291
```

For this customer's data, the index is useless. This does not imply that the index is not useful at a large number of other sites; just not in this situation at this particular time. To take advantage of indexes, you must thoroughly understand your data.

You need to analyze and evaluate every index within the database each time you complete an ETL load. This evaluation is for *all* indexes: those initially supplied with Siebel Analytics and others which you have added to improve performance.

Suggested Action

In the previous case, the best solution is to simply drop the index.

You could take a further step and disable the index in the `ddl sme.ctl` file, the file that controls creation of the index. Of course, if the data makes another shift in the future, the index could become useful at that time.

Refresh Optimizer Statistics

Given a query, a database server generates a set of potential execution plans, and then executes the optimal plan. If cost-based optimization is enabled, the optimizer chooses the least costly plan. The server estimates the cost of each plan based on statistics stored in the database catalog, statistics that include characteristics of tables, indexes, aggregate tables, and other structures that determine access paths. If the statistics are not fresh, the optimizer can generate a suboptimal plan.

Cost-Based Optimization

To take advantage of cost-based optimizers, you must:

- 1 Select and evaluate indexes for the database.
- 2 Enable the cost-based optimizer.
- 3 Generate statistics for database objects.
- 4 Periodically update the statistics so they reflect the current situation.

Cost-based optimizers are well-suited for data warehousing environments.

CAUTION: Do not assume that a technique that works well for one database will work similarly for others. Oracle, DB2, and SQL Server each have their own optimizers which generate query execution plans peculiar to their individual background assumptions. Query execution plans can differ markedly.

The Necessity of Current Statistics

Databases change over time, and as they change the statistics become stale and must be updated. Otherwise, the generated query plan may not be optimal.

To insure optimal query performance, the administrator must periodically calculate statistics for the database using tools provided by the database vendor. These tools vary by vendor. Statistics should be calculated for all tables and indexes which include those used by the server's query rewriter (aggregate tables in DB2 and materialized views in Oracle).

When You Refresh Statistics

To refresh statistics for a table, you submit a single Oracle DDL statement. This statement refreshes statistics for the table and all the indexes defined on the table. Oracle recursively splits up the analyze statement to analyze the table and its indexes.

When you execute the default statement, such as:

```
Analyze table w_person_d compute statistics ;
```

Oracle processes the entire analyze statement as an autonomous unit contained within a single undo transaction. This will also be held in a single instance of the SORT_AREA_SIZE in the PGA.

To speed up an analyze statement, use this sequence of steps:

```
Alter session set sort_area_size = 50000000  
Analyze table w_person_d compute statistics for table;  
Analyze index w_person_d_idx1 compute statistics ;
```

Fresh statistics for an index are critical to the cost-based optimizer.

Vendor Tools to Refresh Statistics

All three database vendors provide tools that calculate and update statistics for the cost-based optimizers. The table below lists these tools by vendor.

Database	Tool	Description
DB2	RUNSTATS	<p>SQL command that calculates statistics for the physical characteristics of a table and its indexes, which include number of records, number of pages, and average record length.</p> <p>For additional information see <i>DB2 Administration Guide V7.1, Volume 3: Performance</i>.</p>
Oracle 8i and 9i	DBMS_STATS	<p>DBMS_STATS is a PL/SQL package that generates and manages statistics for cost-based optimization. Use this package to gather, modify, view, statistics on indexes, tables, columns, and partitions.</p> <p>The SQL ANALYZE command updates statistics for pre-8i database tables.</p> <p>For additional information, see <i>Oracle9i Database Performance Guide and Reference</i>.</p>
Pre-Oracle 8i	ANALYZE	
SQL Server 2000	sp_createstats	<p>A system stored procedure that creates indexes on all eligible columns in all user tables in the current database.</p> <p>Microsoft recommends that the database option AUTO_CREATE_STATISTICS is set to ON, which is the default and which automatically updates statistics.</p> <p>For additional information, see the Microsoft white paper <i>RDBMS Performance Tuning Guide for Data Warehousing</i>.</p>

When To Update Statistics

The following events signal that statistics for the data warehouse should be updated.

- 1** During the initial load of data into the data warehouse.
 - a** Update statistics for the general tables (_G) immediately after they have been loaded and before the staging and dimension table loads.
 - b** Update statistics for the dimension tables (_D) immediately after they have been loaded but before loading the fact tables.
 - c** Update statistics for the fact tables (_F) immediately after they have been loaded but before running the remaining ETL.
 - d** After the data warehouse has been loaded with the initial ETL and the query indexes have been created, update statistics for the indexes.
- 2** After an incremental update or load.
- 3** Any time when an index within the data warehouse is created or dropped.
- 4** After an index has been reorganized.

NOTE: For information on the Query indexes in the Siebel Data Warehouse refer to *Siebel Analytics Server Administration Guide* and *Siebel Analytics Installation and Configuration Guide*.

Reorganize Indexes

When you initially create an index, the database server attempts to store the index entries in contiguous physical pages linked with pointers. This structure reflects the sequential nature of B-tree indexes and optimizes the sequential read operations the database server makes when it scans an index.

At some point, the additional read operations required to scan an index degrades query performance. In some cases, the increase in scan time can be dramatic. The remedy for this kind of fragmentation is index reorganization, an operation that compacts the index pages to minimize fragmentation and restores to a greater degree the sequential order of the physical pages.

Administrators always have the option to drop and recreate indexes. This two-statement operation allocates a new set of index pages which are roughly contiguous and sequential. All the database vendors also supply tools which the administrator can use to reorganize indexes.

Vendor Tools

The easiest way to reorganize an index is to simply drop and recreate the index using the DROP and CREATE INDEX commands. This manual operation two-step operation usually results in a clean index but can be inefficient, consume too many computer resources, and in some cases be impractical. Each of the database vendors has additional tools that reorganize indexes more efficiently.

Database	Tool	Description
DB2	REORG	Utility that rebuilds and compresses the physical structure of indexes. For additional information see <i>DB2 Administration Guide V7.1, Volume 3: Performance</i> .
Oracle 8i and 9i	ALTER INDEX ... REBUILD	SQL command that reorganizes an existing index to make it more compact. You can also use this command to change the index storage characteristics. This statement uses the existing index as the basis for the new one and is usually much faster than drop and create statements. For additional information, see <i>Oracle9i Database Performance Guide and Reference</i> .
SQL Server 2000	DBCC DBREINDEX	Statement that can rebuild just a single specified index for a table or all of its indexes and takes advantage of optimizations not available with individual drop and create statements.
	DROP_EXISTING	Clause for the CREATE INDEX command that makes the reorganization more efficient than a simple drop and create statement. For additional information, see the Microsoft white paper <i>RDBMS Performance Tuning Guide for Data Warehousing</i> .

When to Reorganize

Index reorganization may help when the index's data is not be arranged efficiently. The index's data becomes scattered when the table is modified by insert or delete statements. You can gain a measure of index fragmentation by looking at trends in statistics generated for the indexes. Look for overflow pages, rapidly increasing page counts, increasing number of leaf pages, and an increase in the number of levels for B-tree indexes (NLEVELS in DB2 statistics).

Periodic Reorganizations

Periodically reorganizing indexes is important for query performance and is a task that should not be overlooked by the data warehouse administrator. You can set up and schedule jobs to automatically reorganize indexes using the tools provided by your database vendor.

Drop Indexes

Indexes can greatly improve query performance and are among the most effective means to improve query performance. They should always be considered in a data warehousing environment which is a “read-mostly” environment. Nevertheless, indexes do incur significant costs, and you should drop any indexes which are not being used by queries.

Indexes incur the following costs:

- Occupy physical database storage.

The amount of space an index occupies depends on the size of the table, the size and number of columns in the index, and the kind of index.

- Require more processing time for update operations.
- Require periodic maintenance.
- Can require more query compilation time.

Indexes provide alternative access paths. The cost-based optimizers generate execution plans for potential indexes, calculate the cost of executing the alternative, and then compares the cost of each plan. If the database server never references an index, then these indexes are adding time to the compilation of some queries, and this can bear on the query's performance.

Which Indexes Are Used

You can determine whether queries are using an index with tools provided by database vendors. You can use the EXPLAIN PLAN command to determine whether a specific query uses an index. You can reduce the time require when you follow this approach by looking at a representative sample of you queries.

Oracle 9i Databases

Oracle includes an ALTER INDEX MONITORING USAGE command that collects statistics over a period of time. The most effective approach is to monitor your query workload during a representative interval.

Review Configuration Parameters

Several options can improve query performance such as parallel query, partitioning, local indexes, and so on.

Evaluate Parallel Query Operations

Databases support Parallel Query operations. Give sufficient resources, decision support queries can be improved when several server processes work in behalf of a single query.

Computer Host

At minimum, a high-level survey should be made of the computer host and its operating system to determine whether *obvious* bottlenecks exist. A single bottleneck could be the source of user complaints about poor query response time.

- CPU, Memory, I/O system
- Network Services
- Connection pooling (Siebel Analytics server relief)
- Workload Isolation of Data Warehouse and Transaction Systems

Miscellaneous Tips

Another thing you can do to increase the performance of queries is measure how many users you will have, or more correctly, how many sessions in the database you will have at any given time. Then see how much memory is on the available server, and set the values for SORT_AREA_SIZE and SORT_AREA_RETAINED_SIZE rather high. 20, 30 or 40 MB per session is not uncommon in a data warehouse situation. Also set HASH_AREA_SIZE rather large.

Never use Oracle MTS (multi-threaded server) for an OLAP DSS type database, it does not lend itself well, as the one long-running process will consume shared server resources and slow down all others. In this case, dedicated connections is without any doubt the way to go.

The Database Servers

Computer Host

This chapter shows you how to improve the performance of the extract, transform, and load process. The actions you can take depend on the underlying database.

This chapter describes three steps you can take to improve the ETL process:

- [Remove Unused Batches on page 68](#)
- [Rearrange Batches for Balance on page 77](#)
- [Drop and Recreate Indexes on page 79](#)

Remove Unused Batches

Siebel Analytics includes many star schemas, but you may need to load and maintain only those you actually use. To improve the ETL process, you should evaluate which stars you actually use, and then customize the batch jobs so they only maintain these stars.

This action requires that you carefully define batches that load only the dimensions and facts you actually use. This requires that you determine the sessions you need, define them as batches, set the characteristics of each batch, and disable all other batches shipped with Siebel Analytics.

Example Company

For example, a company named “Data Warehouse Inc.” only intends to use the Activity and Service Request fact tables. Then, the administrator needs to maintain only these two fact tables and the dimensions they reference.

NOTE: The examples in the following sections show you how to set up the initial load batches; you must also set up the incremental batches in a similar fashion.

Determine the Dimension Tables

To determine the dimension tables referenced by the two fact tables, look in *Siebel eBusiness Data Warehouse Data Model Reference* guide. According to this guide, the Activity and Service Request fact tables reference these dimension tables:

- | | |
|----------------|-------------------|
| ■ Agreement | ■ Opportunity |
| ■ Asset | ■ Person |
| ■ Day | ■ Position |
| ■ Entitlement | ■ Product |
| ■ Geography | ■ Program |
| ■ LOV | ■ Region |
| ■ Organization | ■ Service Request |

Define the Batches

Now that you have defined all the required dimensions and fact tables, you need to set up batches to load these tables. The following procedure rearranges the batches already defined in the Informatica repository. These batches were shipped with Siebel Analytics.

This example rearranges the sessions into two batches so they load only the dimension and fact tables listed above:

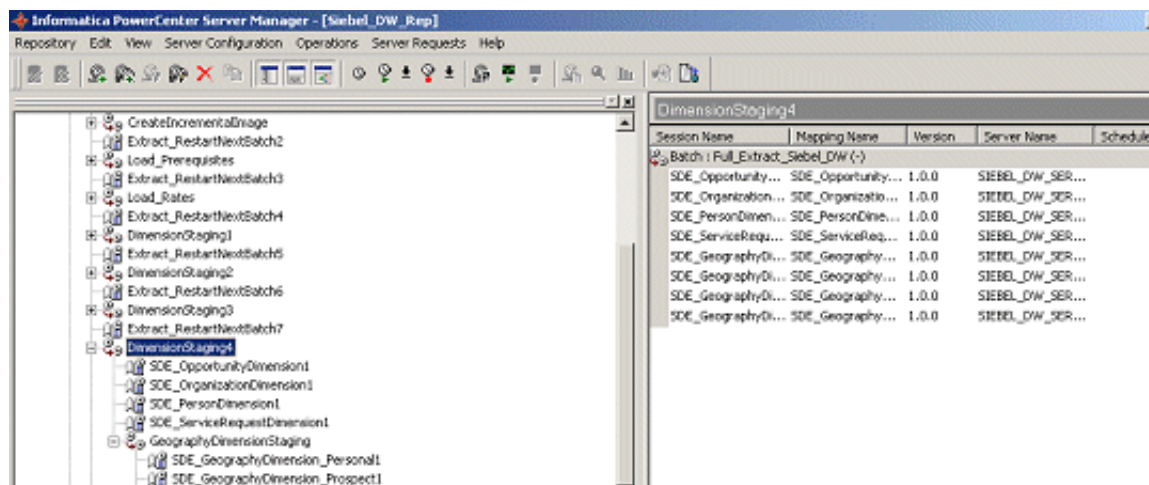
- DimensionStaging3
- DimensionStaging4

As shipped, these batches contain some of the required sessions as well as several sessions which are not required.

To set up these batches, you need to locate the sessions that load the required dimensions tables and move them into either batch three or four. You also need to remove the sessions which are not required.

To rearrange batches

- 1 Login to the Informatica server manager and open Siebel_DW_Rep.
- 2 Open Full_Extract_Siebel_DW batch as shown in the figure below.



- 3 Locate the sessions required to load all the required fact and dimension tables.

Some of these sessions already reside in the DimensionStaging3 and DimensionStaging4, but several do not.

- 4 Drag the names of the required sessions to the DimensionStaging3 and DimensionStaging4 batches as shown in the following table.

Session Name	Current Batch	Destination (batch)
SDE_ProductDimension1	DimensionStaging1	DimensionStaging3
SDE_RegionDimension1	DimensionStaging2	DimensionStaging3
SDE_EntitlementDimension1	DimensionStaging2	DimensionStaging3
ActivityTemp	FactStaging1	DimensionStaging3
SDE_ActivityCost Fact 1	FactStaging2	DimensionStaging4
SDE_ServiceRequestFact1	FactStaging3	DimensionStaging4
SDE_ActivityFact1	FactStaging4	DimensionStaging4

CAUTION: ActivityTemp batch must run before SDE_ActivityCost Fact 1. For a successful load, you must always keep dependencies among the sessions and batches in mind.

- 5 Remove unnecessary sessions from the DimensionStaging3 and DimensionStaging4 batches as illustrated in the following table.

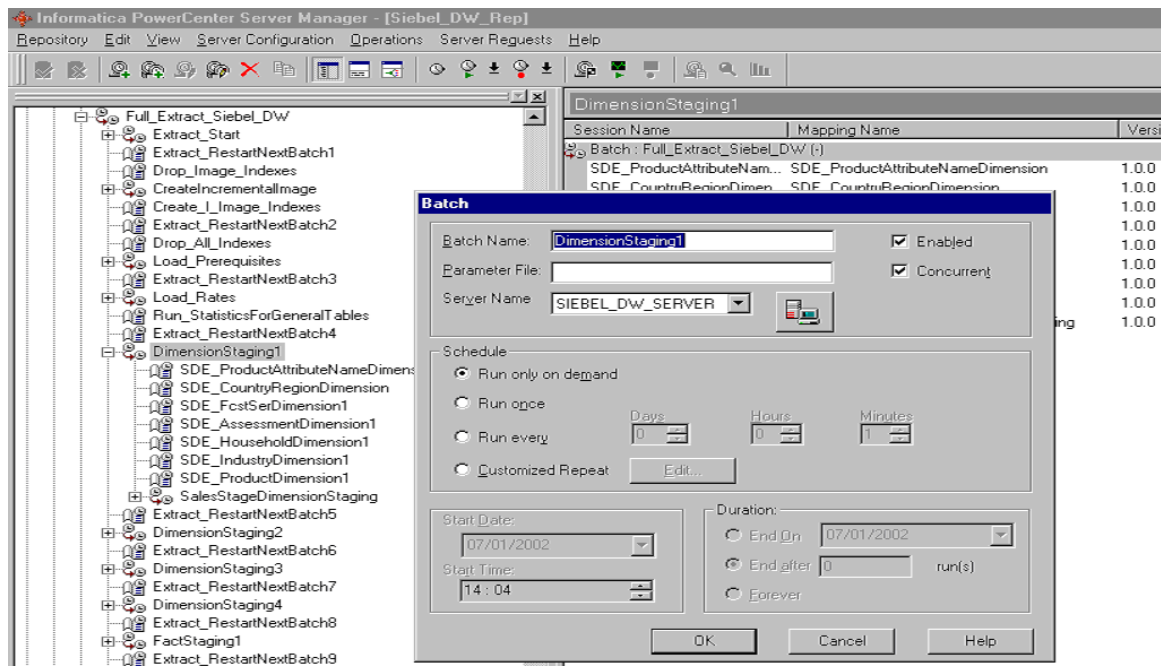
Rather than removing and destroying the sessions that are not required, they are moved to a batch that will be disabled. Then, if you need the session in the future, you can retrieve it from the disabled batch.

Session Name	Current Batch	Destination Batch
SDE_OrderDimension1	DimensionStaging3	DimensionStaging2
SDE_QuoteDimension1	DimensionStaging3	DimensionStaging2
SDE_ResponseDimension1	DimensionStaging3	DimensionStaging2
SDE_ProductAttributeDimension_LoadQuoteItem1	DimensionStaging3	DimensionStaging2

Session Name	Current Batch	Destination Batch
SDE_ProgramOffer_Helper1	DimensionStaging4	DimensionStaging1
SDE_ProductAttributeDimension_LoadOrderItem1	DimensionStaging4	DimensionStaging2
SDE_ActivityFact1	FactStaging4	DimensionStaging4

6 Rename and configure each batch.

- a Select the batch.
- b From the Menu, select Operations > modify batch name as shown in the figure below.

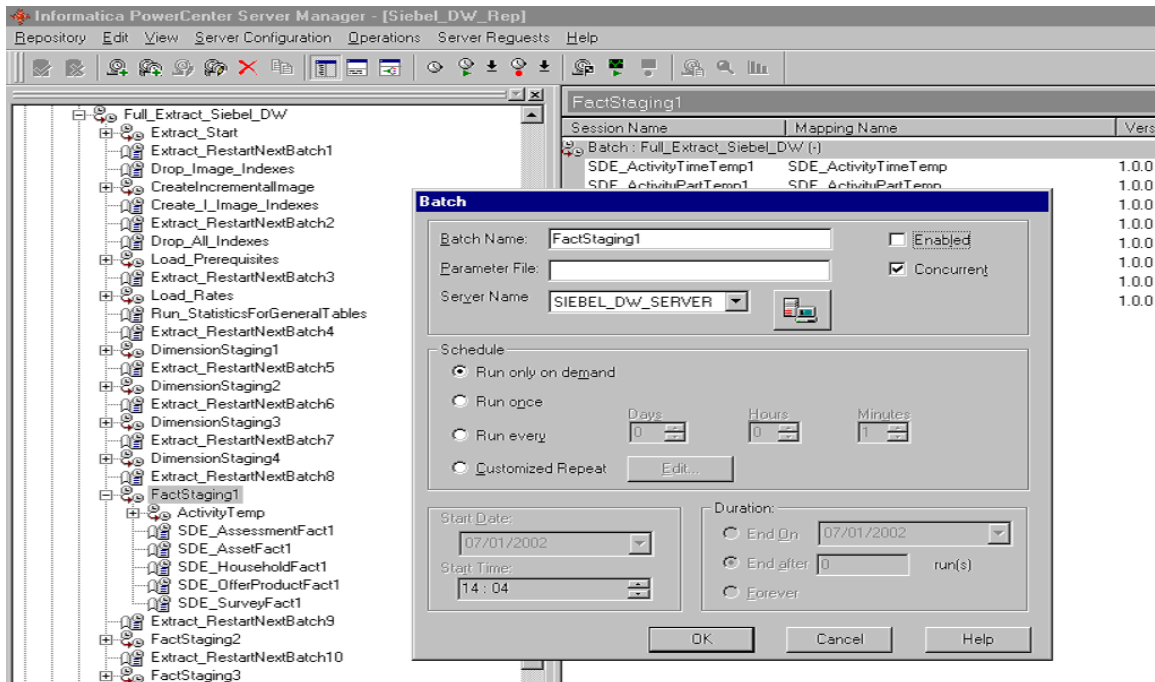


- c Change the names to StagingDWINC_1 and StagingDWINC_2.
- d Check Concurrent.

This enables the mappings within the batch to run in parallel.

7 Disable the following batches, which are not required.

- DimensionStaging1
- DimensionsStaging2
- FactStaging1
- FactStaging2
- FactStaging3
- FactStaging4



To disable a batch, clear the Enable box.

8 Open Full_load_Siebel_DW-Dimensions batch.

- 9 Rearrange the existing Dimension3 batch as shown in the following table.

Batch Name	Current Batch	Destination (batch)
Region	Dimension1	Dimension3
Position	Dimension1	Dimension3
Agreement	Dimension2	Dimension3
Entitlement	Dimension2	Dimension3
Program	Dimension2	Dimension3
Response	Dimension3	Dimension2
ProductAttributes	Dimension3	Dimension2

CAUTION: Person dimensions have to be loaded after Household dimension. In the present case, the interest is not in Household dimension; otherwise when moving you must make sure that you run one after the other.

- 10 Open Dimension3, MapEnabledDimensions, and Map_dimensions batches.

- 11 Disable the following batches.

- Vendor
- Territory
- Industry

To disable the batch, clear the Enable box.

- 12** Edit MapEnabledDimensions as shown in the figure below.

Batch

Batch Name: ☒ Enabled

Parameter File:

Server Name:

Schedule

☒ Run only on demand

☐ Run once

☐ Run every Days Hours Minutes

☐ Customized Repeat

Start Date:

Start Time:

Duration:

☐ End On

☒ End after run(s)

☐ Forever

- Check Concurrent if OLAP is Oracle; clear Concurrent if OLAP is DB2 or SQL Server 2000.

- 13** Open Full_Load_Siebel_DW_Facts and rearrange the Fact2 and Fact3 batches as shown in the following table.

Batch Name	Current Batch	Destination (batch)
SIL_PersonFact1	Fact3	Fact2
SIL_ResponseFact1	Fact3	Fact2
SIL_CampaginHistoryFact1	Fact3	Fact2
SIL_CampaignOpportunityFact1	Fact3	Fact2
SIL_ActivityFact1	Fact4	Fact3

- 14** Rename Fact3 to FactForDWINC.
- 15** Disable Load_Hierarchy batch if you do not want Account Hierarchy.
- 16** Disable the following batches:
 - Dimensions1
 - Dimensions2
 - Fact1
 - Fact2
 - Facts
 - Dimensions
 - Load_KPI

CAUTION: The dependencies among sessions which must be run must be established. The dependencies of the DW ETL batch as shipped are listed below.

ETL Batch Dependencies for a Full Load

Batch names occur in italics.

- 1** Session SDE_ActivityCost Fact1 has to run after sessions in *ActivityTemp* batch.
- 2** Session SIL_HouseholdDimension1 has to run after session.
SIL_PersonDimension1.
- 3** Within the *ProductAttributeName* batch:

Session SIL_ProductAttributeNameDimension_Unspecified & session
SIL_ProductAttributeNameDimension1 should run after the session
SIL_ProductDimension1.
- 4** Session SIL_SurveyFact1 should run after session SIL_ServiceRequestFact1.
- 5** Session SIL_AgreeFact1 should run after session SIL_AgreeItemFact1.
- 6** Do not alter any sessions before the session Extract_RestartNextBatch4 and in
UpdateRowImage batch.

- 7** Any session that is loading a Dimension Staging table or a Fact Staging table can run in parallel (concurrently with each other).
- 8** Dimension tables must be loaded after their counterpart Staging tables are loaded.
- 9** Fact tables and Helper tables have to be loaded after Dimension tables.
- 10** Hierarchy tables have to be loaded after their counterpart Dimension Tables have been loaded.
- 11** *Load_Aggregates* and *Load_Pipeline* batches have to be loaded after *Full_Load_Siebel_DW_Facts* batch is completed.
- 12** Sessions in batches *Facts* and *Dimensions* (these batches are inside the *Full_Load_Siebel_DW_Facts* batch) have to be loaded after the ETL process has populated all the dimensions and facts tables. Sessions in the *Dimensions* batch have to be loaded after sessions in the *Facts* batch.
- 13** *KPI* batch has to be run after all dimension tables have been loaded.
- 14** Slowly Changing Dimension (SCD) sessions should run after their counterpart Dimension table has been loaded e.g. *SIL_PriceListItemDimension_SCD1* should be run after *SIL_PriceListItemDimension1*.

Rearrange Batches for Balance

You can sometimes reduce the amount of time required for a load by balancing the batches in such a way that increases parallel processing.

Example

Suppose the company Data Warehouse Inc. uses the FactStaging2 and FactStaging3 batches to load their data warehouse. These batches are shipped with several sessions within each batch as shown in [Figure 6](#).

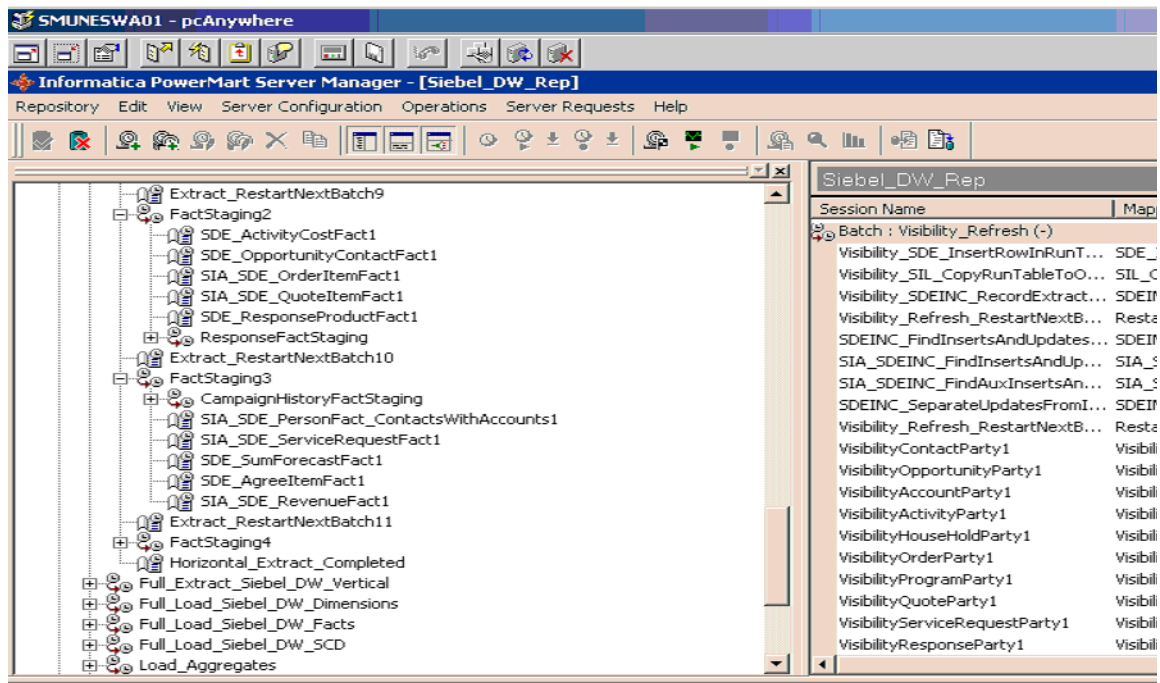


Figure 6. Batches Shipped with Several Sessions

Their Activity Fact, Order Item Fact, Service Request, and Revenue Fact tables have several million rows, and the rest of the fact tables have a few hundred rows.

The loads for the multimillion row fact tables each required nearly four hours (the Service Request required four and a half hours), whereas the other loads required less than thirty minutes per load.

When you run the batches as shown above, the FactStaging2 batch requires four hours, the FactStaging3 batch requires about four and a half hours: eight and a half hours for two batches. The Batch FactStaging2 is running, Activity Fact and Order Item is running for long time while the other sessions took less than 30 minutes which means the load on the Informatica server is very minimal.

In a similar way, if you see the FactStaging3 Batch is running, the Service Request and Revenue Fact ran for long time, and the other session took less than 30 minutes.

You can rearrange the sessions in these two batches so that the Informatica Server can be used more efficiently and therefore save time. You can place the sessions that took a long time in one batch and move all the other sessions into another batch.

For example, you could move the Service Request and Revenue Fact into the Fact Staging2 and the rest of the sessions into FactStaging3 as shown in [Figure 7](#).

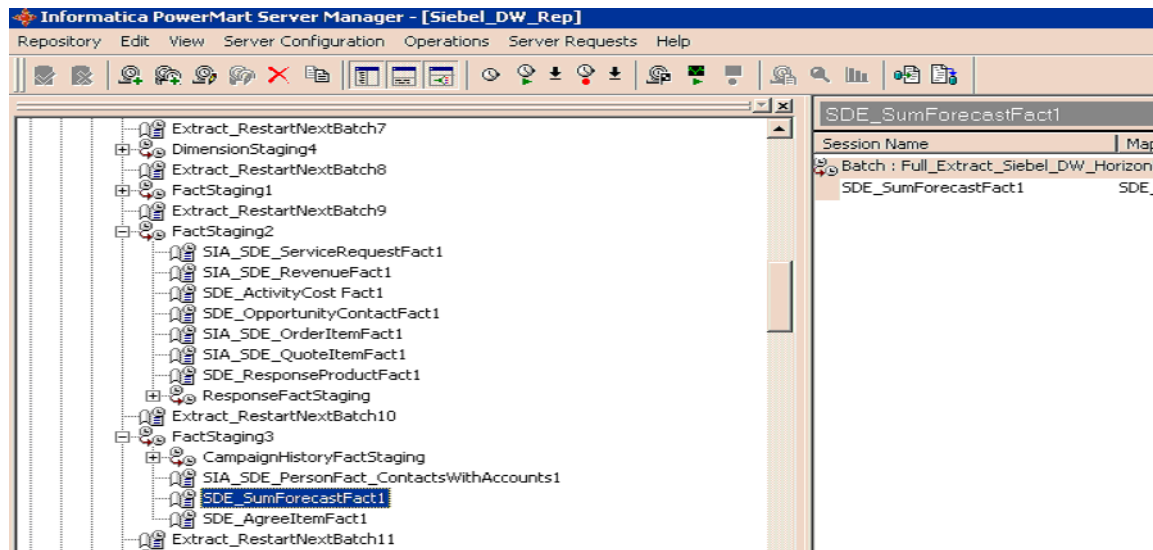


Figure 7. Separate FactStaging Sessions

After rearranging the sessions, the FactStaging2 batch required five hours and the FactStaging3 required about a half hour.

Drop and Recreate Indexes

Siebel Systems recommends that you drop the indexes on the W_ tables before you load these tables, and then recreate the indexes after the load. This can reduce the time required to load the tables.

You can automate this process with batch files that can be attached to an Informatica session as a post-session-script. These scripts automatically drop the specified indexes before the session, and then recreates the indexes after the session loads the data. Each batch file executes an .sql file to drop or create the indexes.

Create and Drop Batch Scripts

The following examples illustrate typical batch files.

Oracle

This example accepts the SQL file as a parameter and includes a hard-coded Oracle database connection.

```
@echo off
echo ..... >> index.log
echo %1 >> c:\indexes\index.log
echo start date >> c:\indexes\index.log
date /T >> c:\indexes\index.log
time /T >> c:\indexes\index.log
echo ..... >> index.log
set cmd=SQLPLUS ORAPERF/ORAPERF@PERFSUN5 @c:\indexes\%1.sql
%cmd% >> c:\indexes\index.log
echo ..... >> index.log
echo End date >> c:\indexes\index.log
date /T >> c:\indexes\index.log
time /T >> c:\indexes\index.log
echo ..... >> index.log
@echo on
```

CAUTION: You need to modify the hard-coded connection username and password so that they meet the security policies at your installation.

You need to include this script as a post-session script to drop or re-create indexes. The index drop and create scripts can go against the OLTP or the data warehouse so you need to have the appropriate connect information as required. The batch executes the script `c:\indexes\%1.sql`.

IBM DB2

This example illustrates a script for IBM DB2.

```
@echo off
set DB2CLP=-1617278812
set DB_CONNECT_STRING=OLAP703
set DB_USER=siebel
set DB_PWD=siebtest
db2 connect to %DB_CONNECT_STRING% user %DB_USER% using %DB_PWD%
db2 -tvf %1 > %1.log
db2 disconnect %DB_CONNECT_STRING%
exit
```

To get the `db2clp` parameter, enter “`echo %DB2CLP%`” in the IBM Db2 -> db2 command window. You do not need to connect.

NOTE: Scripts that create and drop indexes shipped with Siebel Analytics are available on the Siebel SupportWeb.

Even though these scripts can be executed at the end of any session, it is suggested that you create dummy sessions and attach the `postsession` script to those sessions.

Dummy Sessions and Initialization

Dummy sessions are already defined for sessions shipped with Siebel Analytics. For these cases, you can simply open the session, and reference the batch script and the location of the create or drop script.

To initialize a session, you need to include the command in the Post Session Commands line as shown in [Figure 8](#).

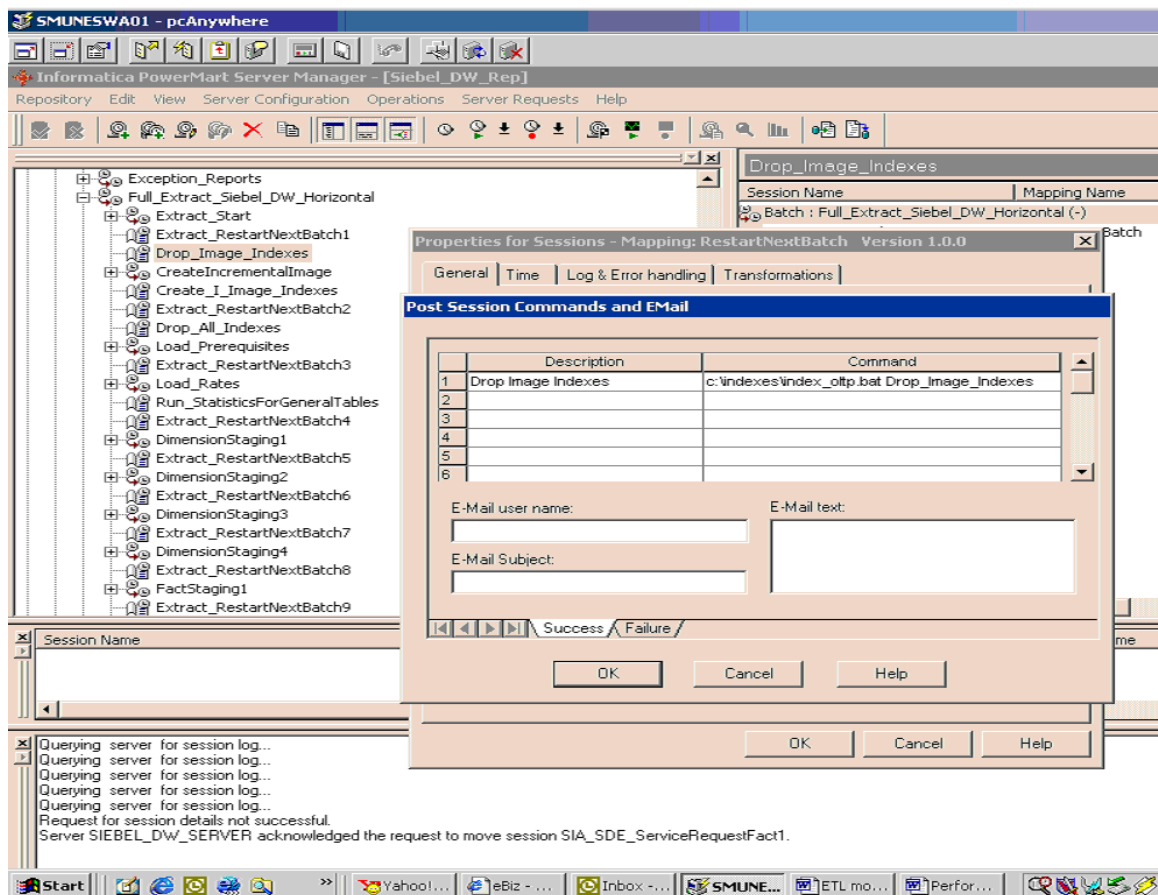


Figure 8. Post-Session Dummy Batches

Batches for the Initial Extract

You can use the following approach for the Full_load_Siebel_DW and Full_Extract_Siebel_DW batches which perform the initial extract and load of Siebel Data Warehouse.

The step Drop_Image_Indexes (drop_image_indexes.sql) drops the Image table indexes and should be executed after the session Extract_RestartNextBatch1 (Should be executed against the OLTP).

The step Create_I_Image Indexes (create_i_image_indexes.sql) creates the Image table Index after the Image table is built and should be executed after the batch CreateIncrementalImage (Should be executed against the OLTP).

The step Drop Dimension and Fact Indexes (drop_all_indexes.sql) drops all the dimension and fact indexes after the staging tables are populated and prior to executing Full_Load_Siebel_DW batch.

The screenshot displays the Informatica PowerMart Server Manager interface. The main window shows the repository structure for 'Siebel_DW_Rep'. The left pane lists the following objects:

- Siebel_DW_Rep
 - Exception_Reports
 - Full_Extract_Siebel_DW
 - Extract_Start
 - Extract_RestartNextBatch1
 - Drop_Image_Indexes
 - CreateIncrementalImage
 - Create_I_Image_Indexes
 - Extract_RestartNextBatch2
 - Drop_All_Indexes
 - Load_Prerequisites
 - Extract_RestartNextBatch3
 - Load_Rates
 - Run_StatisticsForGeneralTables
 - Extract_RestartNextBatch4
 - DimensionStaging1
 - Extract_RestartNextBatch5
 - DimensionStaging2
 - Extract_RestartNextBatch6
 - DimensionStaging3
 - Extract_RestartNextBatch7
 - DimensionStaging4
 - Extract_RestartNextBatch8
 - FactStaging1
 - Extract_RestartNextBatch9
 - FactStaging2
 - Extract_RestartNextBatch10
 - FactStaging3
 - Extract_RestartNextBatch11
 - FactStaging4
 - Extract_Completed
 - Full_Load_Siebel_DW_Dimensions
 - Full_Load_Siebel_DW_Facts
 - Full_Load_Siebel_DW_SCD
 - Load_Aggregates
 - Load_Pipeline
 - Refresh_Extract_Siebel_DW
 - Refresh_Load_Siebel_DW_Dimensions
 - Refresh_Load_Siebel_DW_Facts

The right pane shows a list of session names for the 'Siebel_DW_Rep' repository. The sessions are listed in a table with columns 'Session Name' and 'Map'.

Session Name	Map
Batch : Visibility_Refresh (-)	
Visibility_SDE_InsertRowInRunT...	SDE_...
Visibility_SIL_CopyRunTableTo...	SIL_...
Visibility_SDEINC_RecordExtract...	SDEINC_...
Visibility_Refresh_RestartNextB...	Resta...
SDEINC_FindInsertsAndUpdates...	SDEINC_...
SDEINC_SeparateUpdatesFromI...	SDEINC_...
Visibility_Refresh_RestartNextB...	Resta...
VisibilityContactParty1	Visibil...
VisibilityOpportunityParty1	Visibil...
VisibilityAccountParty1	Visibil...
VisibilityActivityParty1	Visibil...
VisibilityHouseHoldParty1	Visibil...
VisibilityOrderParty1	Visibil...
VisibilityProgramParty1	Visibil...
VisibilityQuoteParty1	Visibil...
VisibilityServiceRequestParty1	Visibil...
VisibilityResponseParty1	Visibil...
VisibilitySegmentParty1	Visibil...
VisibilityPartyLogin1	Visibil...
Visibility_Refresh_RestartNextB...	Resta...
SDEINC_UpdateImageWithInser...	SDEINC_...
SDEINC_UpdateImageForUpdat...	SDEINC_...
SDEINC_IdentifyDeletes_Visibility	SDEINC_...
SDE_OpportunityParty_LoadDel...	SDE_...
SDE_SegmentParty_LoadDelete...	SDE_...
SDE_ServiceRequestParty_Load...	SDE_...
SDE_ResponseParty_LoadDelete...	SDE_...
SDE_QuoteParty_LoadDeletedR...	SDE_...
SDE_HouseHoldParty_LoadDelete...	SDE_...
SDE_ProgramParty_LoadDelete...	SDE_...
SDE_OrderParty_LoadDeletedR...	SDE_...
SDE_ContactParty_LoadDeleted...	SDE_...
SDE_ActivityParty_LoadDeleted...	SDE_...
SDE_AccountParty_LoadDeleted...	SDE_...

Version 7.5

The ETL Process

Drop and Recreate Indexes

The step CreateHouseHoldIndexes (create_household_indexes.sql) should be executed after the batch Dimension2 under the folder Full_Load_Siebel_DW_Dimensions.

The step Create Dimension Indexes (create_Indexesfor_fact .sql) should be executed after the batch Dimension3 under the folder Full_Load_Siebel_DW_Dimensions as shown in [Figure 10](#).

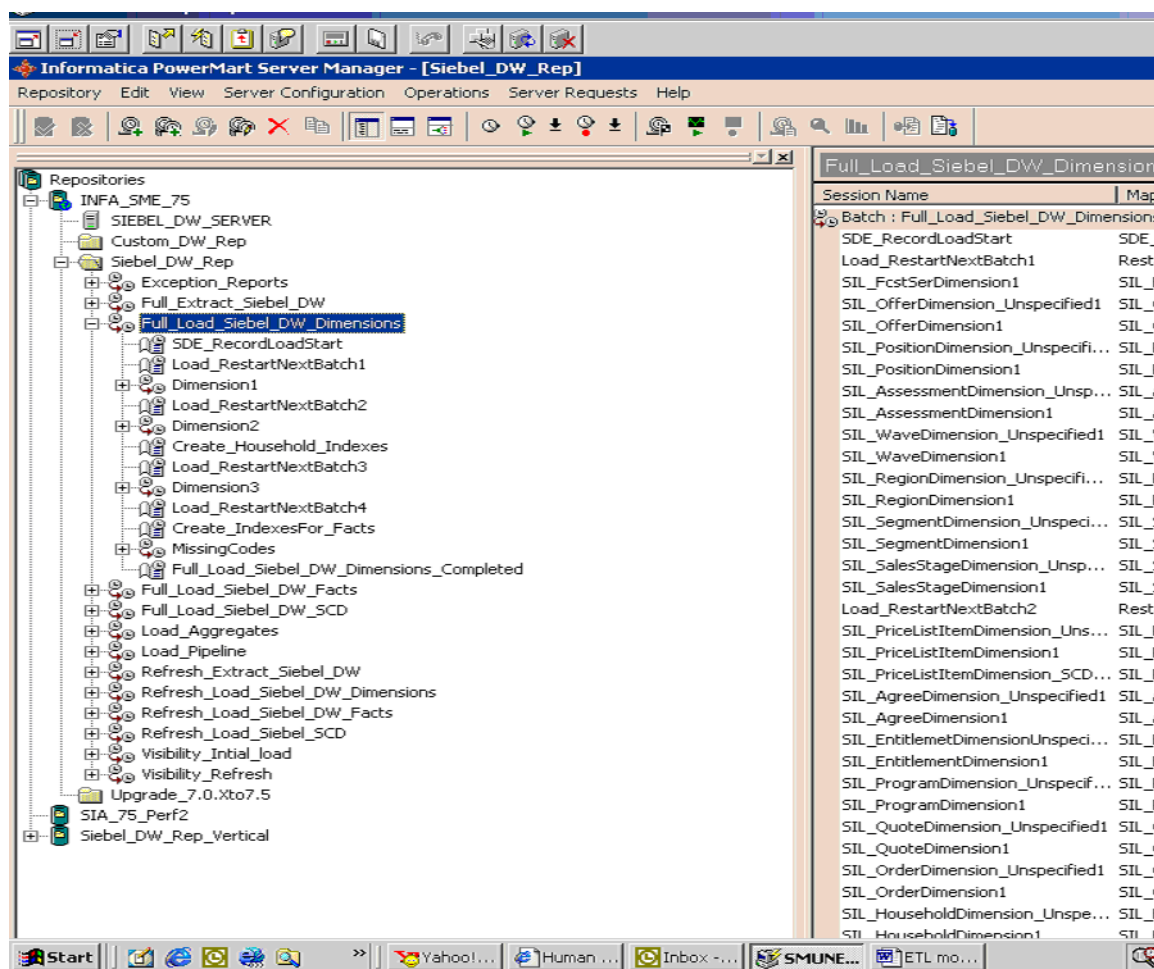


Figure 10. Initial Extract Post Session Dummy Batches

The step Create Service Request Indexes (create_servicerequest_indexes.sql) should be executed after the batch Fact3 under the folder Full_Load_Siebel_DW_Facts.

The step Create Indexes For Aggregates (create_indexes_for_aggreagates.sql) should be executed right after the batch Load_Hierarchy batch under the folder Full_Load_Siebel_DW_Facts.

The step Create Rest Indexes (create_rest_indexes.sql) should be executed after the batch UpdateRowImange under the folder Full_Load_Siebel_DW_Facts.

The ETL Process

Drop and Recreate Indexes

The step Create R Image Index (create_r_image_indexes.sql) should be executed after the batch UpdateRowImage. (It should be executed against the OLTP). See Figure 11.

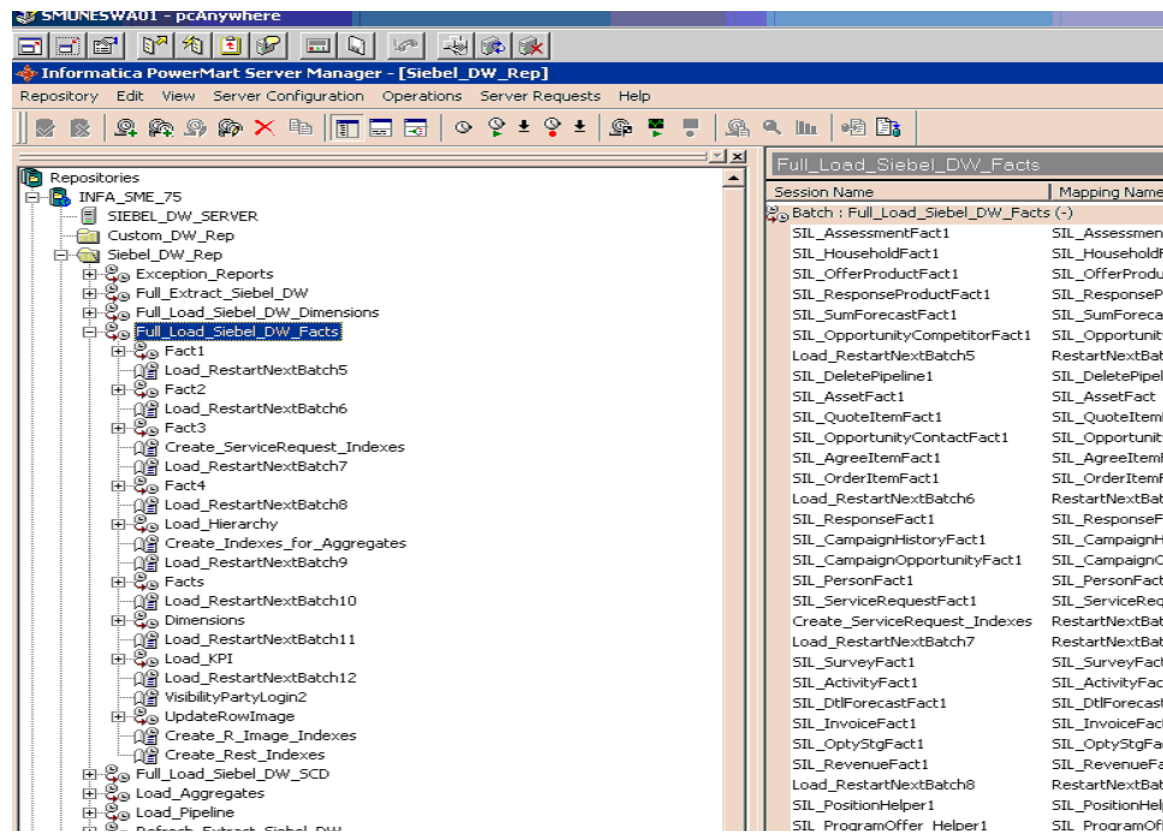


Figure 11. Create R Image Index

The step Create SCD Indexes (create_scd_indexes.sql) should be executed after the batch SCD under the folder Full_Load_Siebel_DW_SCD as show in [Figure 12](#).

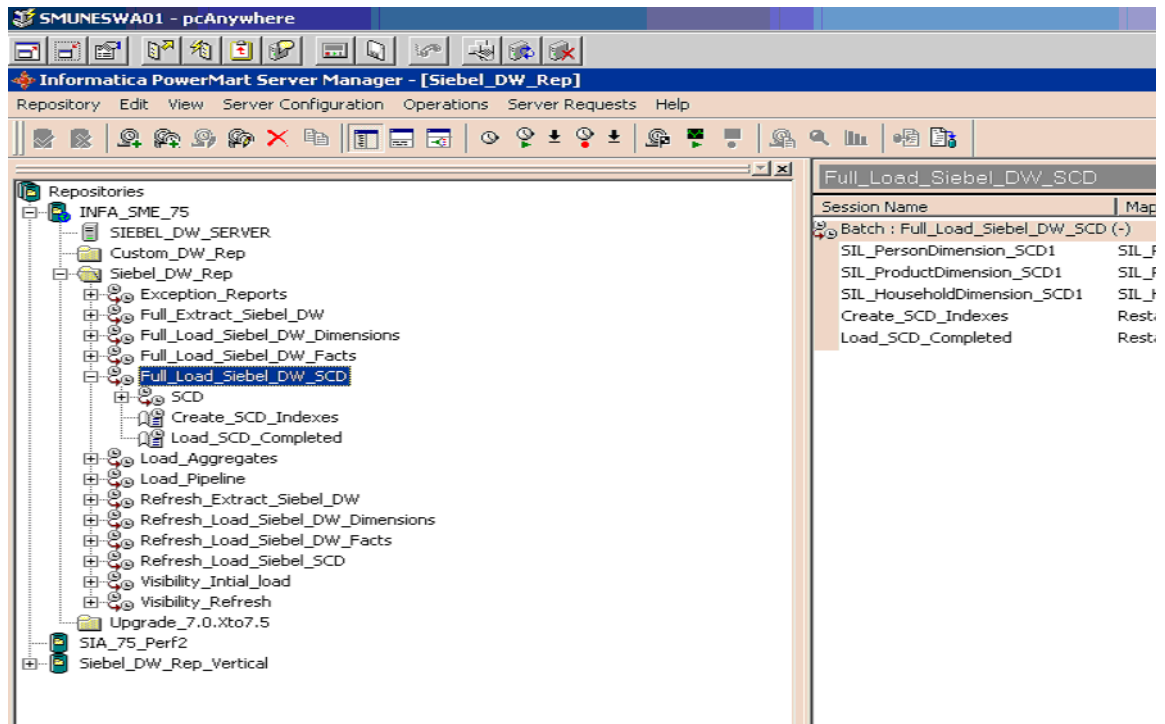


Figure 12. Create SCD Indexes

The step Drop Aggregate Indexes (drop_aggregate_indexes.sql) should be as the first session under the folder Load_Aggregates.

The ETL Process

Drop and Recreate Indexes

The step Create Aggregate Indexes (create_aggregate_indexes.sql) should be executed after the batch LoadAgg2 under the folder Load_Aggregates. See [Figure 13](#).

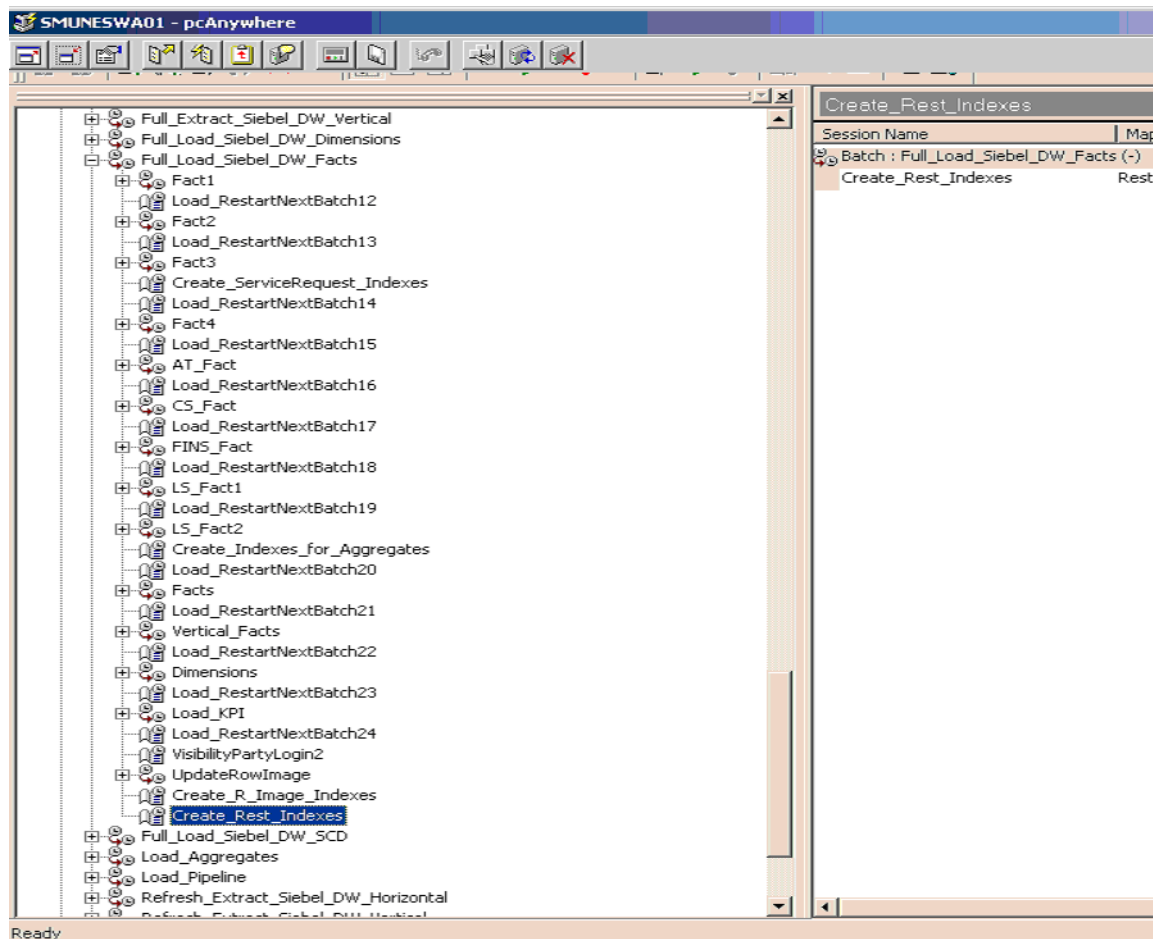


Figure 13. Create Aggregate Indexes

Split Index Script Files for Parallelism

Suppose Data Warehouse Inc. used the script `Create_Rest_indexes.sql` to create indexes using the session `create_rest_indexes` in [Figure 14](#).

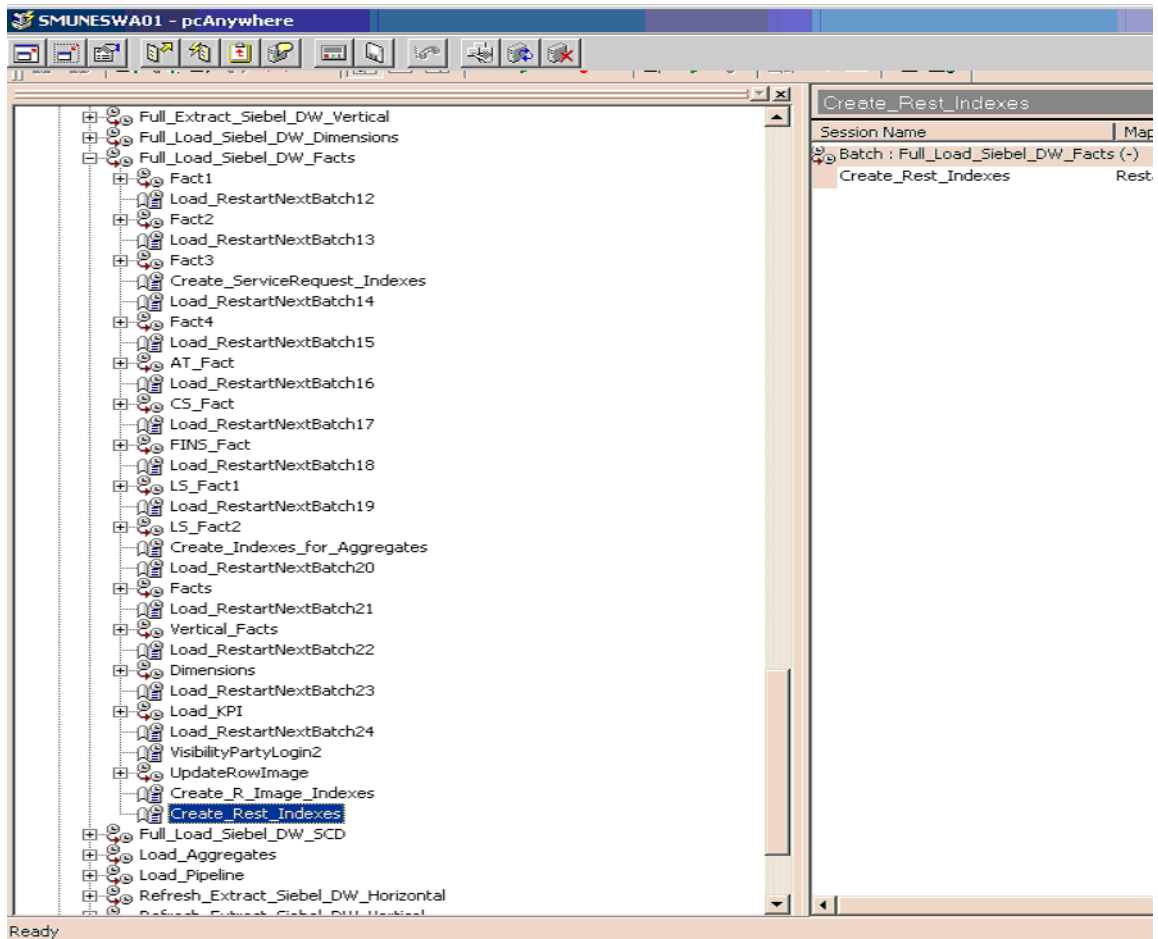


Figure 14. `Create_Rest_indexes.sql`

An administrator can potentially improve performance by splitting the .sql script in to three scripts:

- Create_rest_indexes1.sql
- Create_rest_indexes2.sql
- Create_rest_indexes3.sql

To create the indexes in parallel

- 1** Create a new batch called *CreateRestIndex*.
- 2** Check the concurrent option.
- 3** Move the existing session *CreateRestIndex* into the newly crated batch.
- 4** Add two new sessions after the existing session (*createRestindex*).
- 5** Modify the post session properties to point to the new .sql script files.

Splitting the index scripts reduced the load time by roughly an hour and forty-five minutes. See [Figure 15](#).

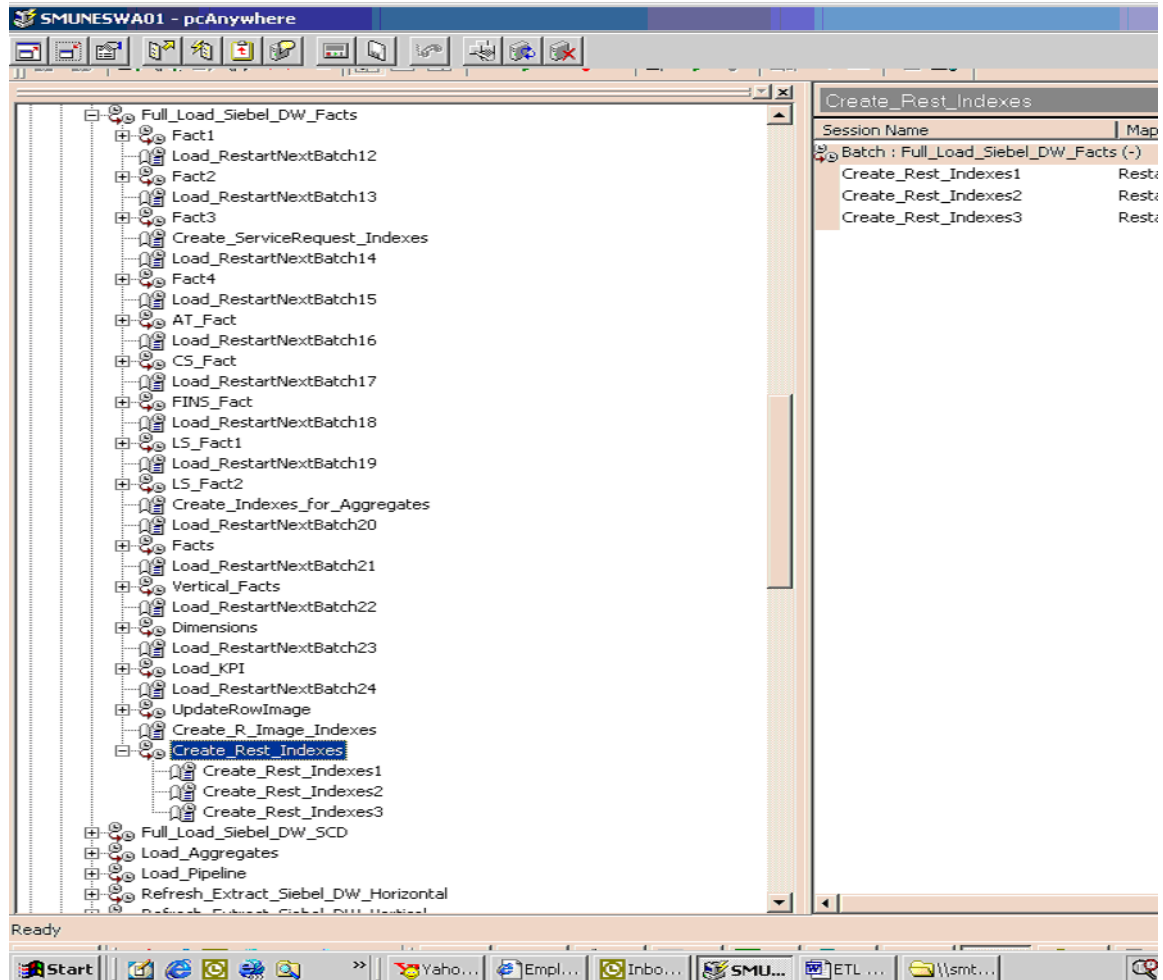


Figure 15. Split Index Scripts

Look for other opportunities to split these scripts into multiple parts as shown in the figure above so you can take advantage of processing the scripts in parallel.

The ETL Process

Drop and Recreate Indexes