
ORACLE®

What is new in Rdb?

V7.3.1 and later

Ian Smith
Oracle Rdb Product Architect
Oracle Rdb Engineering
October, 2014

ORACLE

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Remote Speaker Photo

Ian Smith

Rdb Product Architect, Oracle

Program Agenda

1. Development History
2. Optimizer Changes
3. New SQL language changes
4. New SQL precompiler changes
5. Q & A

Program Agenda

1. Development History
2. Optimizer Changes
3. New SQL language changes
4. New SQL precompiler changes
5. Q & A

Program Agenda

1. Development History
2. Optimizer Changes
3. New SQL language changes
4. New SQL precompiler changes
5. Q & A

Program Agenda

1. Development History
2. Optimizer Changes
3. New SQL language changes
4. New SQL precompiler changes
5. Q & A

Program Agenda

1. Development History
2. Optimizer Changes
3. New SQL language changes
4. New SQL precompiler changes
5. Q & A

Program Agenda

1. Development History
2. Optimizer Changes
3. New SQL language changes
4. New SQL precompiler changes
5. Q & A

Development History

State of the Release

History

- Developed in parallel with V7.2
- Some new features were delivered in various Rdb 7.2 releases
  - Built-in functions such as SYSTIMESTAMP
  - Sort improvements
  - Move data to 64 bits address space (P2)
  - Some query rewrite features
- V7.3.1 first feature release of V7.3

Requirements

- Require OpenVMS V8.3 or V8.4 systems
  - Please ensure all recommended OpenVMS patches are installed (see release notes)

Immediate dependencies on OpenVMS

- OpenVMS Patch kit names:
  - (Integrity) VMS84I_SYS-V0500
  - (Alpha) VMS84A_SYS-V0500
- Dependencies:
  - (Integrity) VMS84I_PCSI-V0400 and VMS84I_UPDATE-V0900
  - (Alpha) VMS84A_PCSI-V0400 and VMS84A_UPDATE-V0900
- Really applies to all OpenVMS systems running Rdb

OpenVMS Problem Description

- Process using JDBC hangs connecting to Oracle Rdb DB.
- Problem Description:
  - A multithreaded process using JDBC to connect to an Oracle Rdb database can hang when using V7.3 RDB$COSIP. RDB$COSIP is using SYS$ACMW for authentication.
- This problem has been fixed.
  - Images Affected:
  - [SYS$LDR]ACME.EXE

Note

- RDB$COSIP.EXE is not multi-version
- Will supersede older versions when V7.3 is installed
- Therefore, this problem may affect V7.2 production systems even if V7.3 is not in use but is installed

Current releases

- V7.3.1 in September, 2013
- V7.3.1.1 in January, 2014
- V7.3.1.2 in October, 2014
- Oracle Rdb strongly suggest using V7.3.1.1 or later when converting databases using **RMU Convert** or **RMU Restore** (from an older version)
- All customers are encouraged to upgrade their V7.3.1.* systems to V7.3.1.2 as soon as possible.
  - Known problem with zero cardinality

Optimizer Changes

System Table Changes

- **create database** now implicitly creates system tables with **sorted ranked** indices
- **RMU Convert** also changes system indices to **sorted ranked**
- Benefits:
  - Eliminate duplicate chains
  - Fixed sizes allows better page utilization
  - Improved performance for DDL operations
    - For example, DROP VIEW

Internal use of Bitmapped Scan

- Rdb now enables **bitmapped scan** for all system table references generated by internal tools
  - SQL and RDO queries

Optimize for

- Queries can now specify **optimize for bitmapped scan** on a **select**, **insert** … **select**, **update**, **delete** statements
- Compound statements can use **pragma (optimize for bitmapped scan)**
- RMU Unload supports **bitmapped_scan** option for the Optimize qualifier
  - The optimizer will attempt to use bitmapped scan if possible

Query Outlines

Inline query outline syntax

OPTIMIZE OUTLINE clause

- Outline can now be included with the query
- Allows tuning in cases where the outline can not be stored in the database

OPTIMIZE OUTLINE clause

```
select last_name, middle_initial, first_name
from employees
where last_name = 'Toliver' and first_name = 'Alvin'
optimize
    as test1
    outline (mode 0
        as (
          query (subquery (
              EMPLOYEES 0 access path index E1_INDEX
              ) ) )
        compliance optional
        execution options (total time));
~Query Name: "TEST1"
~S: Outline "(unnamed)" used
…
LAST_NAME         MIDDLE_INITIAL   FIRST_NAME
 Toliver           A.               Alvin
1 row selected
```

Query Rewrite

Simplifying query during compile

Nullability

- Make use of nullability knowledge to simplify query

- For instance
  where employee_id is NULL
  can never be **true** if there is a PRIMARY KEY or NOT NULL constraint on the column

SORT simplification

- If value is a constant or parameter that does not vary during SORT then it can be eliminated
- For **order by** can reduce VM usage, or even allow the SORT to be eliminated
- Especially beneficial for **distinct** which often includes string literals

Constant folding

- Evaluate simple expressions in the compiler, rather than generating runtime code
- Detect simple transformations
  - **Value** + 0 transforms to **Value**
  - **Value** * 1 transforms to **Value**
  - **Value** * 0 transforms to 0
  - 0 – **Value** transforms to to –**Value**
- Applies to unscaled integer type and floating types
- **Value** must be not nullable

Comparison Simplification

- Some applications (usually generated) include queries like:
  where 1 = 1 and …
- Optimizer now eliminates TRUE predicates (1 = 1) during the compile phase

Comparison Simplification

- Some applications (usually generated) include queries like:
  where 1 <> 1 and …
- Optimizer now eliminates FALSE predicates during the compile phase

Comparison Simplification

- Some applications (usually coding error) include queries like:
  where project_id = NULL
- Optimizer now eliminates UNKOWN predicates during the compile phase
- (Note probably meant to be **project_id is NULL**)

AND, OR and NOT propagation of nullability

- As expressions get changed to **true**, **false** or **unknown** we apply logic to the query tree and remove any branch that isn't useful (always **false**)
- May lead to whole table access being eliminated

CASE expression pruning

- Applying logic rules to **case when** Boolean expression allows elimination of never possible branches
- Applies to various functions such as NULLIF, NVL,NVL2, COALESCE, ABS, DECODE, and SIGN
- In some cases remove condition completely

Data types for COUNT

And related operators

COUNT

- In prior versions COUNT was accumulated using an **integer**
- Now Rdb uses a **bigint**
- Automatically converted to smaller types in existing applications
- Dynamic applications must allow for **bigint** type

STDDEV, VARIANCE and AVG

- Each of these function uses the row count to compute their result. This internal count is also a **bigint**

COUNT (value-expr)

- Reimplemented for V7.3.1
- Previously the expression was translated to:
  - COUNT (*) FILTER (WHERE value-expr IS NOT NULL)
- This meant that SQL null-elimination semantics were never returned
- COUNT() couldn't report NULL values eliminated because those values were previously excluded
  - Require CDD V7.2.0.4 to support computed by and views with this syntax

COUNT…

- The side effect is that generated internal query (BLR) changed
- The query outline signature is different
- May need to redefine query outlines for these cases

DIVIDE

- DIVIDE always returns a **double precision** result
- In prior versions it might result in a real result (if using **tinyint**, or **smallint**)

SQL Language Changes

BINARY and BINARY VARYING

Subtitle

BINARY

- Fixed length binary string
- Padded with X'00' octets
  (during string compare or when a small string assigned to larger string)
- Usage for small images, encrypted values, compressed data, etc.

BINARY VARYING

- Varying length binary strings
- Length included (similar to VARCHAR)
- VARBINARY is a synonym

C/C++ Programmers

- Programmers can use $SQL_VARBINARY pseudo type to declare a typedef for binary data
- Then use .len and .body sub-fields to access length (int) and body (char) data

Supported in SQLDA using new types

- BINARY VARYING (VARBINARY)
  - Value: 909
  - Symbolic name: SQLDA_VARBINARY
- BINARY
  - Value: 913
  - Symbolic name: SQLDA_BINARY

List of Byte Varying

Storage Changes

Storage Model

- Supported *segmented strings* since first release in 1984
  - Designed as a list of segments, virtually unlimited in size
- With V4.1 introduced vector based storage for improved storage
  - goal was compact space filling storage

Storage issue

- I/O pattern is many segments stored plus a chain of pointer segments
- Pointer segments could grow to fill whole page if many segments stored
- Search for free space becomes complicated because now you have many fixed sized segments and some variable sized segments
- Some customers resort to defining RDMS$USE_OLD_SEGMENTED_STRING so that the old style chain format is used

New support

- Allow some control with a tuning logical (ported to 7.2.5)
  - RDMS$BIND_SEGMENTED_STRING_PSEG_SIZING
- Directs Rdb to use one of three algorithms
  - 0, use the current sizing algorithm
  - 1, use the maximum segment length to limit the pointer segment size
  - 2, use the average segment length
- Some round up occurs to ensure basic operation

Compact Storage

- Pointer segment - list of data references
  - Integer length
  - DBK pointer to the data segments

Example V7.2 – 9 data segments

• Example 1	• Len=9
• <empty>	• Len=0
• The	• Len=3
• Rain	• Len=4
• In	• Len=2
• Spain	• Len=5
• Falls	• Len=5
• Mainly	• Len=6
• On the Plain	• Len=12

Store segment even for zero length data

Compact Storage

- We observed that if actual data is tiny then
  - wasted overhead
    - Overhead: record header + TDX/LDX entries on page
  - and I/O to fetch
- Examined usage where there are variable length segments
  - Often see zero length segment for paragraph breaks
  - Have seen one application store telemetry readings as list of FLOAT values

Changes in V7.3

- Now store the data in the pointer segment if less than or equal to 8 bytes
  - (Note: RMU Convert /NOCOMMIT databases do not use new algorithm)
- Zero length segments often appear in source and comments stored in the metadata
- The example list of FLOAT values now 100% stored in the pointer segment
  - (Note: LIST OF BYTE VARYING columns not rewritten by RMU Convert)

Example V7.3 – just 2 data segments

- Example 1
- The
- Rain
- In
- Spain
- Falls
- Mainly
- On the Plain

- Len=9
- (in pointer)
- (in pointer)
- (in pointer)
- (in pointer)
- (in pointer)
- (in pointer)
- (in pointer)
- Len=12

Declare LOCAL Temporary View

New syntax

LOCAL TEMPORARY VIEW

- Allows a module to have a global view definition without a view being defined in the database
- Based on **declare local temporary table** support
- View definition is loaded from the module when the first routine is called

Local Temporary View

```
SQL> declare local temporary view module.a
cont>  (eid edit string 'XXBXXX'
cont>       comment is 'Employee id'
cont>  ,num_jobs query name 'NUMBER_JOBS'
cont>  ,started query header 'When'/'Started'
cont>  ,current_start
cont>     default value for dtr '1-Jan-1900 00:00:00.00')
cont>  as select employee_id, count(*),
cont>             min (job_start), max (job_start)
cont>        from job_history
cont>        group by employee_id;
SQL>
SQL> select * from module.a where eid <= '00164';
                          When
 EID         NUM_JOBS    Started      CURRENT_START
 00 164             2     5-JUL-1980    21-SEP-1981
1 row selected
SQL>
```

REPLACE Statement

Alternate to INSERT statement

New statement

- Acts like INSERT but will preserve uniqueness for PRIMARY KEY using a pre-DELETE action
- If there is no PRIMARY KEY it is a simple INSERT
- Triggers are affected
  - Activates BEFORE and AFTER INSERT trigger
  - Activates BEFORE and AFTER DELETE trigger if a primary key value exists
- Valid statement in CREATE TRIGGER

REPLACE example with PRIMARY KEY

```
SQL> alter table WORK_STATUS
cont>     add constraint PK_WORK_STATUS
cont>         primary key (STATUS_CODE)
cont>         not deferrable;
SQL>
SQL> replace into WORK_STATUS
cont>  values ('0', 'INACTIVE', 'RECORD EXPIRED');
1 row replaced
SQL> select * from work_status;
 STATUS_CODE   STATUS_NAME   STATUS_TYPE
 1             ACTIVE        FULL TIME
 2             ACTIVE        PART TIME
 0             INACTIVE      RECORD EXPIRED
3 rows selected
```

Sequences and Identity Changes

ISO/ANSI Standard

Sequence

- Create sequences was based upon Oracle Database syntax
- Now part of SQL standard
- Rdb accepts both Oracle and ISO/ANSI SQL syntax for **create** and **alter sequence** syntax
- Biggest change is the NO is a separate keyword

Identity

- Based on commonly used syntax from other database systems
  - e.g. IDENTITY (1, 2)
- Based wholly on Sequence feature
- Only one identity per table, enforced by naming the special sequence after the table
- Now supported by ISO/ANSI SQL Standard
- Clauses are the same as create sequence
  - e.g. IDENTITY (start with 1 increment by 2 cycle **cache**)

Reverse attribute

V7.3.1.2

- New **reverse** keyword for **create sequence** statement and also **identity** clause
- Changes the value returned by **.currval** and **.nextval** pseudo columns
- Values are bit reversed
- Allows index keys to be scattered around the index even those base sequence is systematically increasing

SAVEPOINT Support

Sub-transaction units

Savepoint support

- **savepoint** allows for a small inner part of a transaction to be named and managed.
- **release savepoint** is used to discard the controlled changed
- **rollback to savepoint** is used to undo part of the current transaction

SAVEPOINT example

```
set flags 'TRANSACTION';
begin
set transaction read write;

insert into SV_X values (10);

savepoint D;
insert into SV_X values (20);
insert into SV_X values (30);
insert into SV_X values (40);

rollback to savepoint D;

insert into SV_X values (50);
insert into SV_X values (60);

commit;
end;
```

SAVEPOINT example

```
~T Start_transaction (8) on db: 1, db count=1
~T Savepoint "D" (8.2) on db: 1
~T Rollback to Savepoint "D" (8.2) on db: 1
~T Commit_transaction (8) on db: 1
~T Prepare_transaction (8) on db: 1

select * from SV_X order by V;
          V
          10
          50
          60
3 rows selected
```

Savepoint support

- Each **savepoint** is given a unique name which is specific to a database attach
- If multiple database ALIAS are in use then use the ALIAS name to provide context.
- `SAVEPOINT db1.mysavepoint`

Savepoint support

- Currently Rdb limits a session to just one active **savepoint**
- Extending this to multiple nested savepoints is planned for a future release
- Current limit imposed because of complexity of in memory metadata

WITH Clause

Prefix to SELECT clause

WITH clause

Simplifying queries

WITH clause of SELECT statement

- Define partial queries for use in a query
  - known as subquery factoring
- Currently used by some Oracle clients to collect data. Enables use of those tools with Rdb
  - Rdb does not support recursive queries

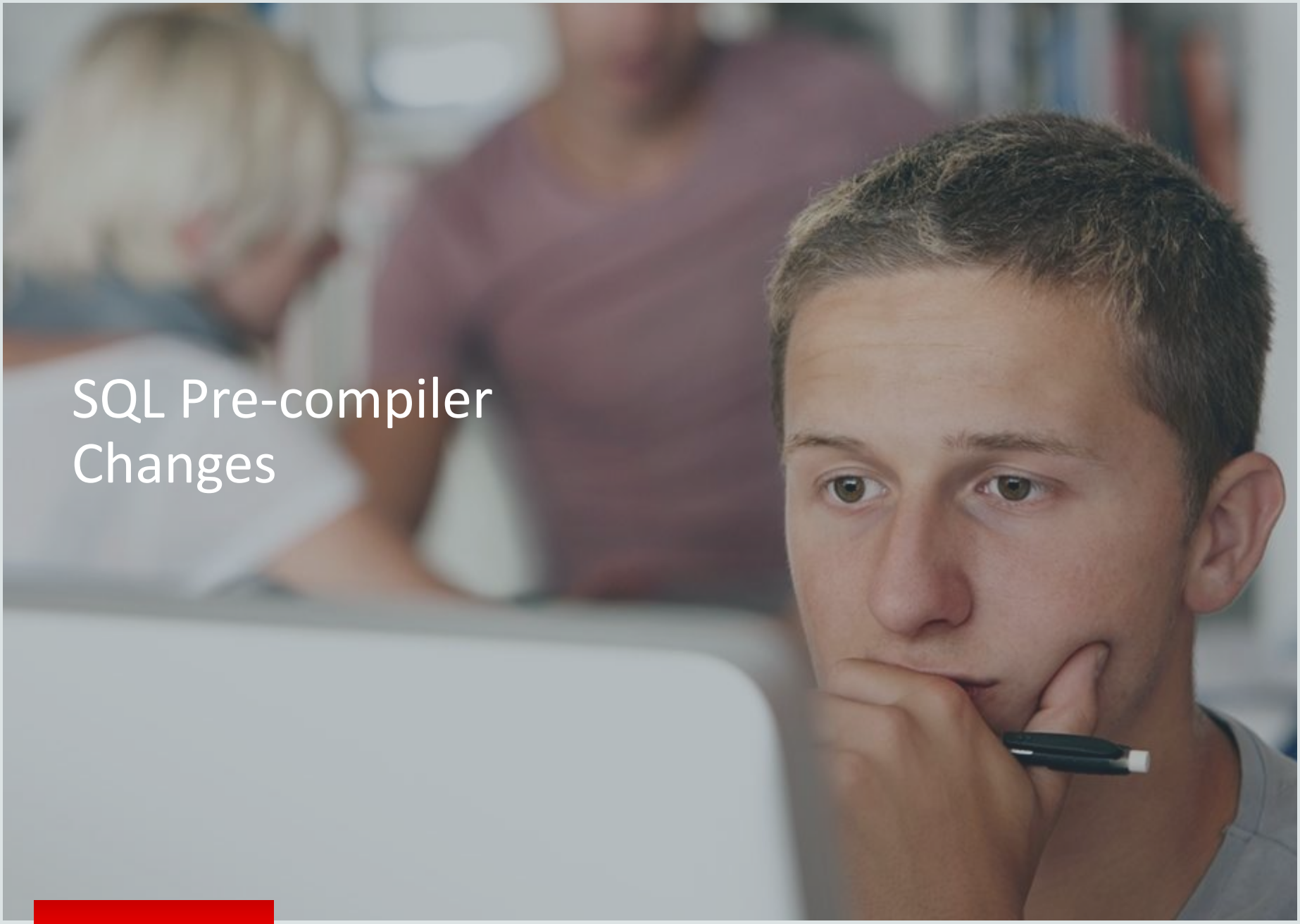
- Note: `begin with hold … end;`
  is deprecated due to this syntax.
  Use `begin pragma (with hold) … end;`
  instead.

WITH example

Definition of factor

```
SQL> with emp as (select *
cont>             from employees
cont>             inner join job_history using (employee_id)
cont>             where job_end is null),
cont>      dpt as (select * from departments)
cont> select e.last_name, d.department_name,
cont>        m.last_name as Manager
cont> from emp e
cont>      left outer join dpt d using (department_code)
cont>      inner join emp m on (d.manager_id = m.employee_id)
cont> order by d.manager_id
cont> ;
 E.LAST_NAME      D.DEPARTMENT_NAME                MANAGER
 Siciliano        Board Manufacturing
…
 Herbener         Engineering
100 rows selected
SQL>
```

Reference to factor name



SQL Pre-compiler Changes

C++ Compiler Support

CXX compiler

Improved support for C++

- SQL pre-compiler generates .C source files from .SC embedded sources
- Calls routines in generated auxiliary module
- The routine prototypes defined for these routines were not C++ complaint

New CC option

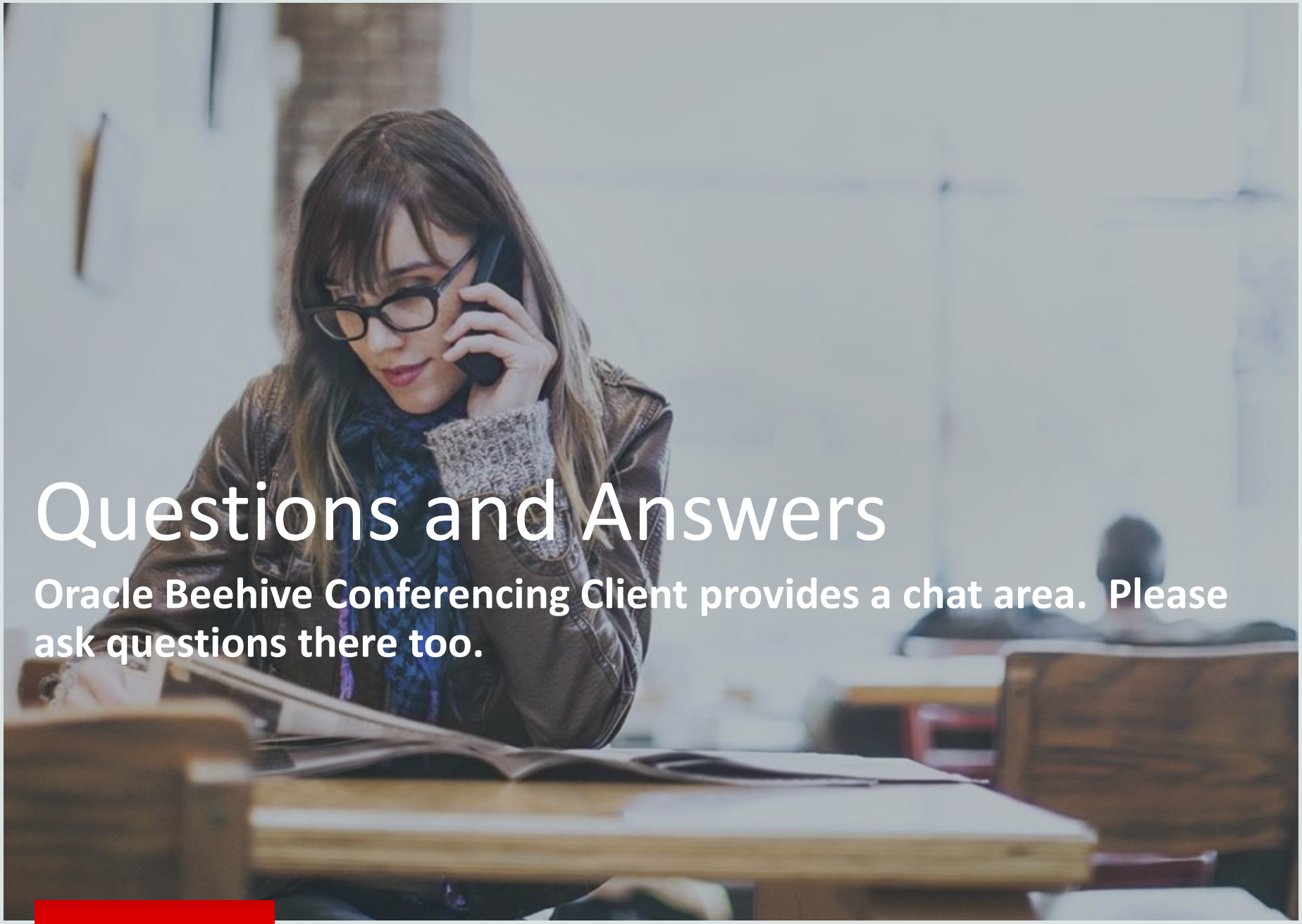
- Can now use /CC=CXX option to change the generated C code
- SQL$PRE will now invoke the CXX compiler to process the augmented C source
- Changes to various structures and call interface
- Any qualifiers (apart from /SQLOPTIONS) must be acceptable to CXX compiler

Changes in generated code

- Prototypes will include parameter definitions
- Prototypes are enclosed by **extern "C" {…}** to prevent the names being interpreted as C++ routines
- **sql_rdb_headers.h** now provides alternate prototypes
- SQLCODE must be defined as **int** type
- Rdb$Message_Vector now a **typedef**, not a **struct** type

Notes

- SQL$PRE is still basically processing C code
- These changes do not include language features from C++
- Sources should use standard C syntax



Questions and Answers

Oracle Beehive Conferencing Client provides a chat area. Please ask questions there too.

Safe Harbor Statement

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Hardware and Software
Engineered to Work Together


ORACLE®