



ORACLE®



BLR: the language inside a query

Ian Smith
Oracle Rdb Product Architect





Overview

- BLR – the language of Rdb
 - Terminology
 - Language Overview
 - Study a SQL cursor implementation
 - Compare simple query with stored procedure (CALL) and multi-statement (BEGIN...END) statements
 - Some final comments



Request

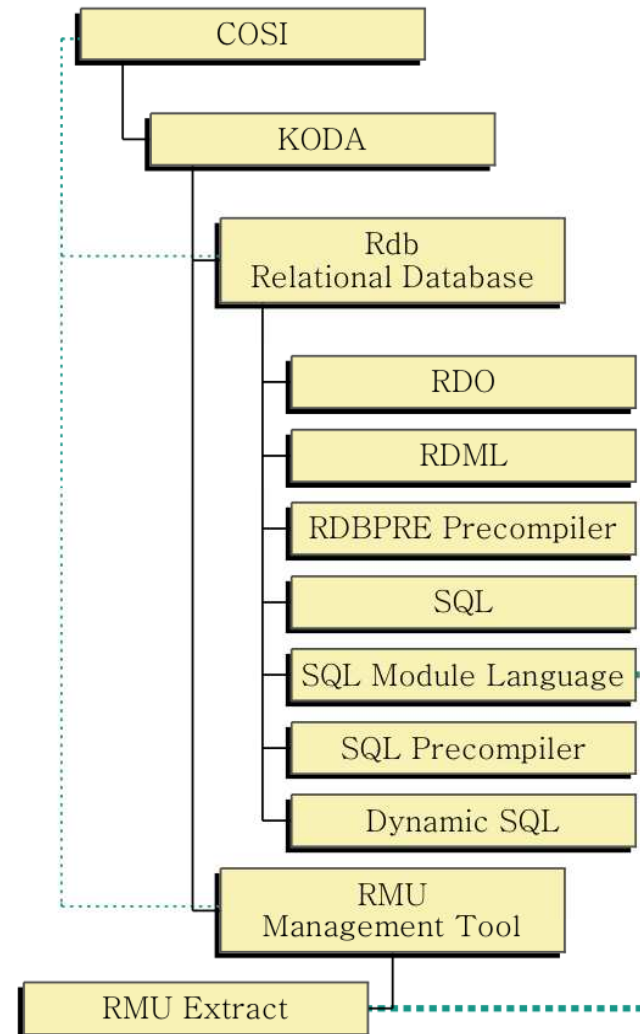
- **Request** - Rdb internal structure that is used to implement a SQL statement, or set of statements
- Each request is uniquely identified by a **request handle** (32 bit value)
- **A request** may include more than one query, or none at all



BLR

- **Binary Language Representation** - compact language that represents a procedural co-routine for the SQL statements
- Each BLR program has an unique signature
- **Co-routine** - two routines that exchange data to implement a single function

- **Rdb Product Structure**
- Not all clients use SQL as the query language
- Client language turned into BLR
- RMU Extract uses SQL plus some internal interfaces





BLR Clients

- Not all interfaces to Rdb use SQL
 - RDO
 - RDML, RDBPRE
 - RMU
 - Some third party tools
- SQL uses BLR to communicate with server
 - Queries compiled with RDBPRE/BLISS



BLR

- Brief overview
 - Language is operator prefix or function oriented
ADD arg1 arg2
SUBSTRING arg1 arg2 { arg3 | END }
 - Easy to parse
 - Compact representation using byte codes
 - Unambiguous expressions
 - Easy to enhance for new SQL support



History

- The language of Rdb is BLR not SQL
- Developed in the early 1980's as a general query language to be generated by various higher level languages
 - Datatrieve
 - RDO
 - RDML
 - RDBPRE
 - SQL
 - FOCUS
 - Rally



Why BLR?

- There was no SQL standard
- Many relational languages
 - Digital's own Datatrieve was widely used
 - RDO was developed for Rdb/VMS
 - Epilogue (later called RDML) developed for Rdb/ELN
- SQL implementations did not include procedural languages
 - Oracle PL/SQL and Sybase Transact/SQL came later
- Needed powerful expression language
- Wanted it to be compact
 - for remote transmission and storage in the database



Procedural Language

- SQL Procedural language was released in 1992 and it defined:
 - Insert, Select, Update and Delete
 - Declare for variables
 - Looped cursors
 - Nested begin/end
 - Explicit leave for loops
 - If-then-else statements
- All these functions existed in BLR in 1984 when Rdb first shipped



Rdb worked to add extra syntax and semantics

- Call statement for user defined procedures
- User defined function calling
- Atomic blocks
- True CASE expressions
- New functions (TRIM, POSITION)
- New semantics (Character sets and multi-octet characters)
- INTERSECT and EXCEPT operators
- CASE statement
- And so on



Loop

- The basic BLR LOOP is the foundation for several SQL statements
 - LOOP ... END LOOP
 - WHILE ... DO ... END WHILE
 - REPEAT ... UNTIL ... END REPEAT
 - FOR ... IN END LOOP
 - Also called counted loop (Rdb extension)



How to see the BLR

- SET FLAGS command
 - BLR – displays the formatted BLR string
 - NOPREFIX – to suppress the offsets
 - CONTROL_BITS – display the semantic masks
 - ITEM_LIST – show setting for BLR compile
- RDMS\$SET_FLAGS logical
- RDMS\$DEBUG_FLAGS logical “Bnch” (deprecated)
- As part of a bugcheck dump
- RMU/EXTRACT/OPTION=DEBUG
- Oracle TRACE collects BLR for query source



Basic Structure

- Byte tags
 - BLR\$K_BEGIN, BLR\$K_END, BLR\$K_FOR,
BLR\$K_CASE, BLR\$K_IF, BLR\$K_LEAVE, ...
- Prefix notation
 - BLR\$K_ADD
BLR\$K_FIELD 1 “A”
BLR\$K_FIELD 1 “B”
 - BLR\$K_MISSING
BLR\$K_VARIABLE 22
- The display indents to show structure



Basic Structure - optional parameters

- SQL: trim (both from last_name)
- BLR\$K_TRIM RDB\$M_TRIM_BOTH
BLR\$K_FIELD 1 LAST_NAME
BLR\$K_END
- SQL: trim (trailing 'X' from last_name)
- BLR\$K_TRIM RDB\$M_TRIM_TRAILING
BLR\$K_FIELD 1 LAST_NAME
BLR\$K_LITERAL
DSC\$K_DTYPE_CHAR 1 (sub-type: 0) "X"



Basic Structure - dialect specific semantics

- SQL: substring (department_name from 1 for 2)
- Using different tags
 - BLR\$K_SUBSTR
 - Original DSRI semantics
 - BLR\$K_SUBSTR2
 - ANSI and ISO SQL (octet based)
 - BLR\$K_SUBSTR3
 - ANSI and ISO SQL (character based)



Basic Structure - dialect specific semantics

- Control bits for various operators
- BLR\$K_FOR
BLR\$K_CONTROL_BITS 8
(searched update)
- BLR\$K_FOR
BLR\$K_CONTROL_BITS 16
(searched delete)



Row processing loop

- BLR\$_FOR
 - Declare Cursor
 - FOR cursor loop
 - SELECT
 - INSERT ... SELECT
 - UPDATE
 - DELETE



Query structure

- SELECT ...
FROM ...rse...
WHERE ...boolean...
GROUP BY ...
HAVING ...boolean...
ORDER BY ...
OFFSET ...
FETCH FIRST ... ROWS
- | |
|------------------|
| BLR\$K_FOR |
| BLR\$K_RSE |
| BLR\$K_BOOLEAN |
| BLR\$K_AGGREGATE |
| BLR\$K_GROUP_BY |
| BLR\$K_SORT |
| BLR\$K_STARTS |
| BLR\$K_FIRST |



RSE (Record Select Expression)

- BLR\$K_JOIN – defines the join type and ordering
- BLR\$K_MERGE – defines UNION and derived tables
- BLR\$K_EXCEPT – define MINUS/EXCEPT
- BLR\$K_INTERSECT – defines INTERSECT
- BLR\$K_PROJECT – defines DISTINCT actions
- BLR\$K_AGGREGATE – processes grouped operations



Other DML

- INSERT
 - maps BLR\$K_STORE
- INSERT ... RETURNING
 - Maps to BLR\$K_STORE2
- UPDATE
 - maps to BLR\$K_FOR with nested BLR\$K_MODIFY
- UPDATE ... WHERE CURRENT OF
 - Maps to BLR\$K_MODIFY
- DELETE
 - Maps to BLR\$K_FOR with nested BLR\$K_ERASE
- DELETE ... WHERE CURRENT OF
 - Maps to BLR\$K_ERASE



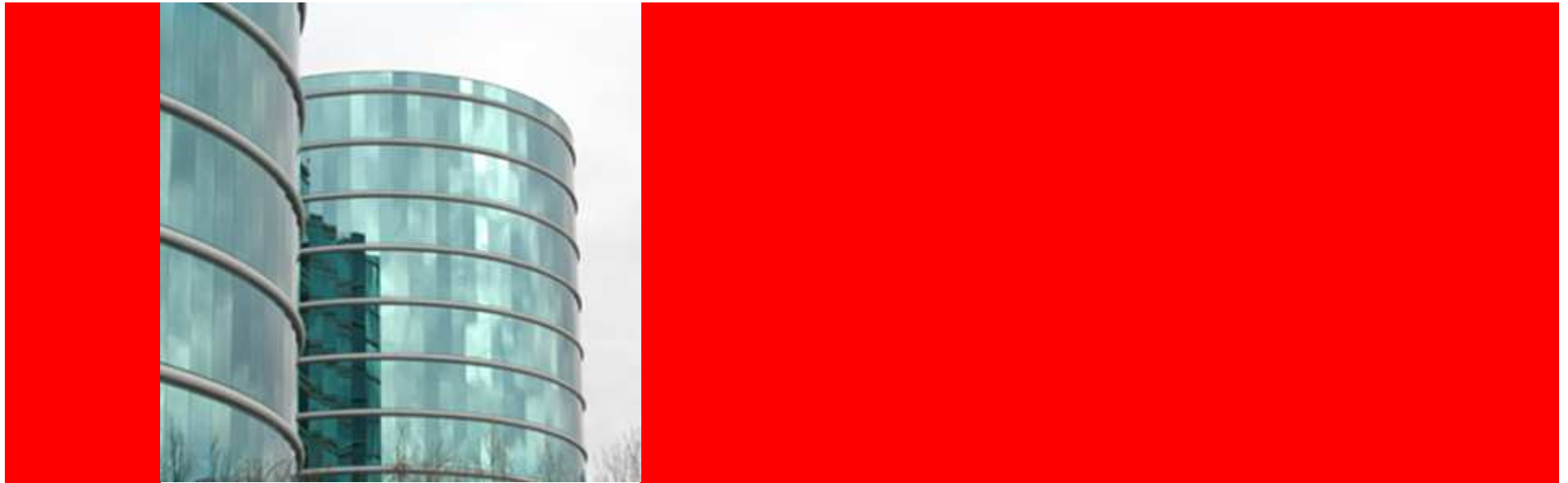
Program Layout

- Start with version and communications records
- (VERSION 4
BLR\$K_BEGIN
BLR\$K_MESSAGE 1 3
DSC\$K_DTYPE_L 0
DSC\$K_DTYPE_L 0
DSC\$K_DTYPE_L 0



Program Layout

- End with completed block
- BLR\$K_END
BLR\$K_EOC)



ORACLE®

Part 1: Simple Query



Outline of Application

- Simple query uses DECLARE CURSOR
- Rows are read using FETCH ... INTO ...
- Rows updated by UPDATE ... WHERE CURRENT OF ...
- Rows deleted using DELETE ... WHERE CURRENT OF ...
- Finish using CLOSE statement

Cursor with Open

```
declare TEST_CURSOR cursor for
  select last_name
  from employees
  where employee_id = :emp_id

procedure OPEN_TEST_CURSOR
  (sqlcode
  ,:emp_id  ID_NUMBER)
;
-- open the cursor, passing data for query
open TEST_CURSOR;
```

Open must
supply
EMP_ID as
referenced in
the DECLARE



Fetch and Delete

```
procedure FETCH_TEST_CURSOR  
  (sqlcode  
   ,:last_name LAST_NAME)  
  ;  
  -- fetch data from the cursor  
  fetch TEST_CURSOR into :last_name;
```

```
procedure DELETE_TEST_CURSOR  
  (sqlcode)  
  ;  
  -- delete the current record  
  delete from EMPLOYEES  
  where current of TEST_CURSOR;
```



Module language and SQL Pre-compiler

- All statements that reference the cursor are seen before code generation
- Allows single request for all actions
- Contains knowledge of each required action
- We will focus on this model for the request



Contrast: Interactive and Dynamic

- These interfaces are statement at a time delivery
- Each statement seen separately
 - i.e. When DECLARE is processed SQL doesn't know if an UPDATE or DELETE will be used
- Must use DBKEY retrieval for positional DELETE and UPDATE
- Separate request for each action
 - In our example 5 requests versus 1
- You may see different semantics when testing queries in Interactive SQL



Displaying BLR

- SQL OPEN statement will compile the request, this is when the BLR can be viewed
- Enable display using **set flags 'blr,control,noprefix'**

Sample BLR [1]

```
(VERSION 4
BLR$K_BEGIN
  BLR$K_MESSAGE 1 4
    DSC$K_DTYPE_L 0
    DSC$K_DTYPE_L 0
    DSC$K_DTYPE_L 0
    DSC$K_DTYPE_CHAR 14 (sub-type: 0)
  BLR$K_MESSAGE 2 1
    DSC$K_DTYPE_CHAR 5 (sub-type: 0)
  BLR$K_MESSAGE 3 2
    DSC$K_DTYPE_L 0
    DSC$K_DTYPE_CHAR 14 (sub-type: 0)
  BLR$K_MESSAGE 4 0
  BLR$K_MESSAGE 5 0
```

Message is both a
'record' and a
synchronization
handle

Sample BLR [2]

```
BLR$K_RECEIVE 2
BLR$K_BEGIN
  BLR$K_FOR
    BLR$K_RSE 1
      BLR$K_RELATION EMPLOYEES 1
      BLR$K_BOOLEAN
      BLR$K_EQL
      BLR$K_FIELD 1 EMPLOYEE_ID
      BLR$K_PARAMETER 2 0
    BLR$K_END
```

...

For ... from
EMPLOYEES where
employee_id = :emp_id

Sample BLR [3]

```
BLR$K_LABEL 0
BLR$K_LOOP
BLR$K_SELECT
  BLR$K_RECEIVE 3
  BLR$K_STATEMENT
  ...update...
BLR$K_RECEIVE 4
  BLR$K_STATEMENT
  ...delete...
BLR$K_RECEIVE 5
  BLR$K_LEAVE 0
  ...fetch next...
BLR$K_END
```

...

Labeled loop
SELECT acts as a
conditional based on
incoming message



Interface to Rdb

- Call interface includes many routines to manage BLR
 - rdb_compile_request (aka DECLARE CURSOR)
 - rdb_release_request (aka RELEASE in dynamic)
 - rdb_start_request (aka OPEN)
 - rdb_unwind_request (aka CLOSE)
 - rdb_start_and_send (aka OPEN + parameters)
 - rdb_receive (aka FETCH)



Call interface

- rdb_start_send_receive (aka CALL and compound statements)
- More advanced interface:
 - rdb_send (used by IMPORT to load a table)
 - rdb_send_buffer (used by RMU Load)
 - rdb_receive_buffer (used by RMU Unload)



Communication

- Client communicates with the server across:
 - Protected mode (user mode to exec mode)
 - Network
- Message (data) must be shipped across an interface to the server (BLR co-routine)



Co-routine Communication

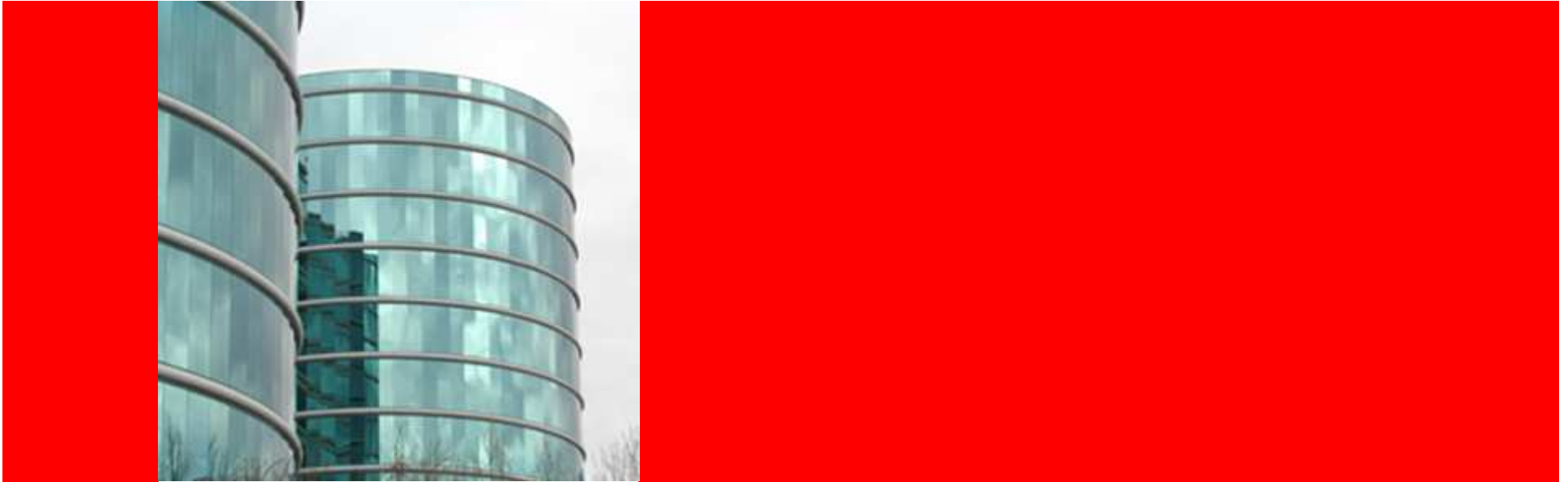
- rdb_start_and_send
(open with emp_id)
- rdb_receive (1)
(fetch into last_name)
- rdb_send (4)
(delete where current)
- rdb_send (3)
(update where current)

SQL Client

- BLR\$K_RECEIVE 2
(enter BLR\$K_FOR)
- BLR\$K_SEND 1
- BLR\$K_SELECT
sees message 4
BLR\$K_RECEIVE 4
- Would get synch error
since we are expecting
message #1

Rdb Server

ORACLE



ORACLE®

Part 2: Call Statement



CALL

- Two variants:
 - Standalone (or single statement CALL)
 - Compound use (or multi-statement CALL)
- Same end result but differ in implementation
- Which is more efficient?
 - Compound use is probably more efficient if the routine is used more than once in the application



MSP and SP

- Use `rdb_start_send_receive` to send parameters (IN and INOUT) and receive output (INOUT, OUT, and SQLCODE/SQLSTATE)
- Single API call to activate many statements
- Might be a CALL or a compound statement



MSP and SP

- **CALL:**
 - passes BLR header (just the BLR\$K_MESSAGE portion)
 - name of the procedure
 - Loads BLR from metadata (row in Rdb\$ROUTINES)
- **Multi-statement procedure:**
 - passes whole BLR program
 - Similar to simple query example



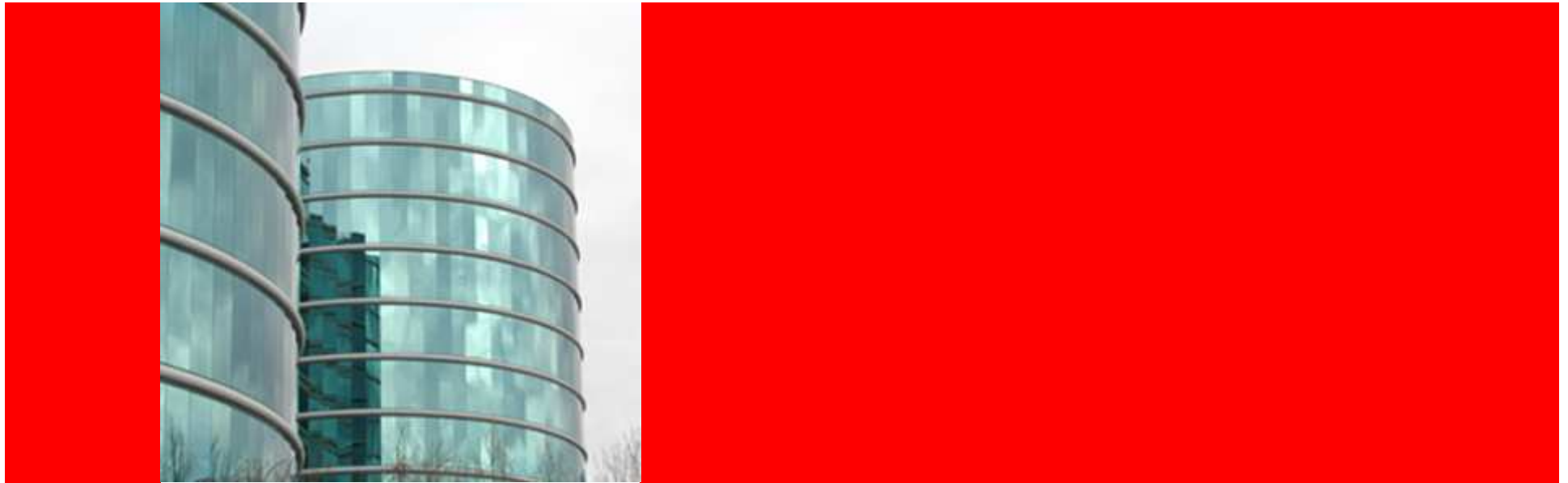
Item list used to pass name of the routine

```
SQL> set flags 'item_list,noprefix';
SQL> call trace_it();
~H Request Information Item List: (len=11)
RDB$K_SET_REQ_ROUTINE_NAME "TRACE_IT"
RDB$K_INFO_END
~Routine Name: "TRACE_IT"
~H Request Information Item List: (len=15)
RDB$K_SET_REQ_OPT_PREF "0"
RDB$K_SET_REQ_NULL_PARAMETERS "1"
RDB$K_SET_REQ_IS_COMPOUND "1"
SQL>
```



Efficiency

- Simple CALL will compile based on default settings of the session (SET OPTIMIZATION LEVEL, etc)
- Routine might be compiled many times
- Compound-use causes just one copy to be compiled and referenced by many calls



ORACLE®

Final comments
Just touching the surface



Watching requests

- SET FLAGS can reveal something about execution
- WATCH_OPEN: see when the 'start' request executes
- Displays the BLR signature (see query outline) to help locate request usage
- WATCH_CALL: see what procedures get called



Watch Open

```
SQL> begin
cont>   optimize as "CURSOR_EXAMPLE"
cont> declare :sc integer;
cont> declare :ln LAST_NAME_DOM;
cont> call OPEN_TEST_CURSOR (:sc, '00164');
cont> trace :sc;
cont> call FETCH_TEST_CURSOR (:sc, :ln);
cont> trace :sc, :ln;
cont> call CLOSE_TEST_CURSOR (:sc);
cont> trace :sc;
cont> end;
~Xo: Start Request 4B3C0F6229B6D9027F7C6C317B4E332F
"CURSOR_EXAMPLE" (query)
```



Dynamic Applications

- Use **begin call P(...); end** instead of direct call **P(...);**
 - P is cached
 - Small request executed to activate the call
 - Allows complex expression arguments
 - Allows optional parameters
 - No signature mismatch concerns



Final words

- I have lived and breathed BLR for over 27 years
- I currently design and update the BLR architecture
- For example, I added polymorphic support for generic function:
 - BLR\$K_VALUE_LIST “Concatenate”
 - BLR\$K_VALUE_LIST “Trunc”
 - BLR\$K_VALUE_LIST “Round”
 - BLR\$K_VALUE_LIST “Least”
 - BLR\$K_VALUE_LIST “Greatest”



For More Information

- Rdb V7.2 Documentation
 - For the SET FLAGS keywords
- Rdb Internals Course
- www.oracle.com/rdb
 - For Rdb Technical Journal articles
- ian.e.smith@oracle.com

QUESTIONS & ANSWERS

ORACLE®