Rdb Internals: Mapping SQL Cursors To DSRI

A feature of Oracle Rdb

By Ian Smith
Oracle Rdb Relational Technology Group
Oracle Corporation
Rdb Internals: Mapping SQL Cursors to DSRI

This report describes how the SQL cursor model is mapped to a single BLR request. It describes each component of the request to assist in analysis of applications at the DSRI (Database Standard Relational Interface) level.

**SQL to DSRI Mapping**

To understand the mapping of SQL to BLR, let us first look at a SQL module that includes a complete set of procedures to process a cursor. It is possible that some applications contain a subset of possible operations and so the generated BLR will differ accordingly.

```sql
module SAMPLE
language GENERAL
authorization RDB$DBHANDLE
parameter colons

declare alias for filename 'DB$:PERSONNEL'

-- declare the RSE for the cursor
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;

procedure CLOSE_TEST_CURSOR
    sqlcode
    ;
    -- close the cursor
    close TEST_CURSOR;
```

1 This article is a revised version of Rdb Technical Notes #18, Mapping SQL Cursors to DSRI
procedure FETCH_TEST_CURSOR
   sqlcode
   :last_name LAST_NAME
   ;
   -- fetch data from the cursor
   fetch TEST_CURSOR into :last_name;

procedure UPDATE_TEST_CURSOR
   sqlcode
   :last_name LAST_NAME
   ;
   -- update the current record
   update EMPLOYEES
   set LAST_NAME = :last_name
   where current of TEST_CURSOR;

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

This set of procedures represent the entire range of functions for a SQL cursor and all specified procedures will be converted to a single BLR request by SQL. The STREAM in RDO or RDML language is processed in a similar way. The BLR below is dumped using SET FLAGS ‘BLR,NOPREFIX’ which generates a complete and reasonably readable version of the BLR program. The annotations on the left were added later for this report.
BLR$K_BEGIN
BLR$K_FOR -- (1) for loop
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$K_END
BLR$K_BEGIN
BLR$K_SEND 1 -- (b) prefetch data
BLR$K_HANDLER
BLR$K_BEGIN
BLR$K_ASSIGNMENT
BLR$K_FIELD 1 LAST_NAME
BLR$K_PARAMETER3 1 1 1 2
BLR$K_ASSIGNMENT
BLR$K_LITERAL
DSC$K_DTYPE_L 0 "0"
BLR$K_PARAMETER 1 0
BLR$K_END
BLR$K_LABEL 0
BLR$K_LOOP
BLR$K_SELECT -- (c) select action
BLR$K_RECEIVE 3
BLR$K_STATEMENT
BLR$K_BEGIN
BLR$K_MODIFY 1 2 UPDATE
BLR$K_CONTROL_BITS 1 WHERE CURRENT OF...
(one record)
BLR$K_BEGIN
BLR$K_ASSIGNMENT
BLR$K_PARAMETER2 3 1 0
BLR$K_FIELD 2 LAST_NAME
BLR$K_END
BLR$K_END
BLR$K_RECEIVE 4
BLR$K_STATEMENT
BLR$K_BEGIN
BLR$K_HANDLER DELETE
BLR$K_ERASE 1 WHERE CURRENT OF...
(one record)
BLR$K_END
BLR$K_RECEIVE 5
BLR$K_STATEMENT
BLR$K_BEGIN
BLR$K_LEAVE 0 FETCH NEXT
The user first performs an OPEN of the cursor. This will generate a call to the routine \texttt{rdb\_start\_and\_send} to start the request, and send data to initialize the RSE of the BLR$\_K\_FOR. The ‘send’ is matched in the BLR with BLR$\_K\_RECEIVE (a). The BLR proceeds to execute the BLR$\_K\_SEND (b) which fetches the first row fetched by the BLR$\_K\_FOR loop (known as prefetching) and stalls the request.

The users only possible actions are CLOSE -- which may call \texttt{rdb\_unwind\_request} to discard the prefetched row, or FETCH NEXT -- which will use \texttt{rdb\_receive} to accept the data already fetched.

Assuming FETCH NEXT is used then the \texttt{rdb\_receive} call will accept the sent data (copy it from the Rdb buffers to the users buffer), and advance the BLR to the BLR$\_K\_SELECT (c) which stalls the request.

At this point the possible actions are FETCH_NEXT (\texttt{rdb\_send} of message 5, then \texttt{rdb\_receive} of message 1), DELETE WHERE CURRENT OF (\texttt{rdb\_send} of message 4), or UPDATE WHERE CURRENT OF (\texttt{rdb\_send} of message 3). The incoming message number from \texttt{rdb\_send} is used to select the action in the BLR, and the BLR program advances:

- when it is 3 the BLR$\_K\_MODIFY (2) is performed and the BLR advances and loops back to the BLR$\_K\_SELECT where it again stalls
- when it is 4 the BLR$\_K\_ERASE (3) is performed and the BLR advances and loops back to the BLR$\_K\_SELECT where it again stalls
- when it is 5 the BLR$\_K\_LEAVE (4) is performed and the BLR leaves the loop returning to the outer BLR$\_K\_FOR which fetches a new row which is returned via the subsequent \texttt{rdb\_receive} call.
Any of these actions can continue until all rows have been fetched. When that happens the BLR$K_FOR terminates and the stall is left on the latter BLR$K_SEND (d) which does not return data but instead sets an end-of-stream indication in the message buffer.

This is a complete cursor example. However, some SQL cursor definitions and usage inform SQL that only a subset of the actions is required. Therefore, the user might only perform FETCH NEXT, or FETCH NEXT and UPDATE, or FETCH NEXT and DELETE. In such cases the BLR$K_RECEIVE blocks for BLR$K_MODIFY and/or the BLR$K_ERASE need not be present and are omitted from the BLR by the SQL compiler.