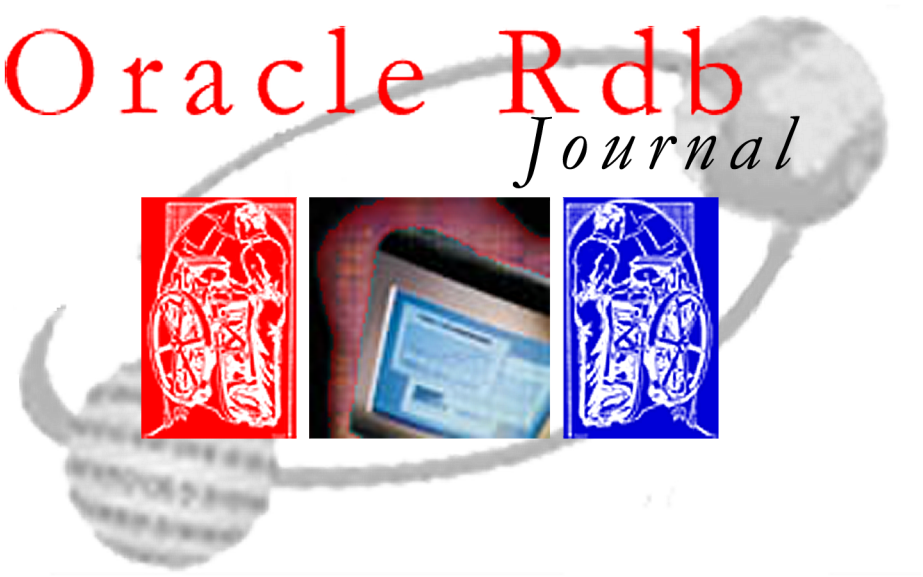

Rdb Optimizer Detailed Strategy

A tool for diagnosing poor query performance or wrong query results

An Article from the Rdb Journal

By Jean-Claude Proteau

March 15, 2002

Rdb Journal

feature

Rdb Optimizer Detailed Strategy

A tool for diagnosing poor query performance or wrong query results

Jean-Claude Proteau

Principal Member of Technical Staff, Oracle Rdb Engineering

March 15, 2002

The ability to get reports showing the execution strategy of Rdb queries has existed for more than a decade. Such reports can help you analyze and possibly improve query performance. This paper describes an option now available for seeing the Rdb optimizer strategies in greater detail than before. This additional detail can be useful not only to help you study query performance but also to help when you have queries that return wrong results or that return the data rows sorted incorrectly.

A database administrator can use this tool to better understand queries written by a customer's application developers. A study of query strategies might help to identify problems in the way queries are formulated. In such case, feedback should go to the application developers. On rare occasions, one might uncover some problem in the way a query is optimized by Rdb. In that case, use the information gathered and report it to the Rdb engineers.

Examples in this paper were generated using Rdb V7.0-62. The output from such examples might differ from that of other Rdb releases due to the ongoing nature of development in this area.

Existing Optimizer Strategy Documentation

It is assumed that you are already familiar with the traditional way of obtaining and interpreting Rdb strategy reports. This paper will not repeat that material other than to include the optimizer notation in a reference table farther on. If you need to refresh your memory, you can read the following:

Oracle Rdb7 Guide to Database Performance and Tuning, Copyright 1996, Oracle Corporation, section 5.8.7, Using RDMS$DEBUG_FLAGS and RDB_DEBUG_FLAGS, and *section C.1, Displaying Optimization Strategy with the S Flag*.

Example 1

Consider this simple query and the different levels of information that can be provided about the query strategy. More interesting examples will be given later along with an explanation of the new, detailed output.

```
select last_name from employees
  where last_name = 'Nobody';
```

First, here is the traditional (undetailed) strategy:

```
Conjunct Get Retrieval
  sequentially of relation EMPLOYEES
```

Contrast that with the detailed strategy:

```
Tables: 0 = EMPLOYEES
Conjunct: 0.LAST_NAME = 'Nobody'
Get Retrieval sequentially of
  relation 0:EMPLOYEES
```

In an un-detailed strategy report, you can see where tables are joined using the match versus cross method, if indexes are used for data retrieval, when the data are sorted, and some types of filtering operations (Conjunct or Bool). When you request a detailed strategy report, you'll be able to see the actual index keys used, the filter expressions (Conjuncts and Bools), the sort keys, and more. The example above shows 0.LAST_NAME = 'Nobody', which is the Conjunct (filter expression) performed on the rows retrieved from the EMPLOYEES table.

Enabling and Disabling Detailed Strategy Output

Section C.1, Displaying Optimization Strategy with the S Flag, in the *Oracle Rdb7 Guide to Database Performance and Tuning*, says that to enable the display of optimizer query strategies in OpenVMS you define the logical name RDMS$DEBUG_FLAGS with the value "S", meaning Strategy. Nowadays, the preferred method of enabling and disabling such features is by defining, instead, the RDMS$SET_FLAGS logical name. Although one can still use RDMS$DEBUG_FLAGS as in the past, it cannot be used for some of the more recent Rdb features, one of which is the flag to enable detailed strategy output. Therefore, from this point forward only the "set flags" method will be discussed.

To enable output of query strategies, you can use the SET FLAGS statement in interactive SQL or you can define the RDMS$SET_FLAGS logical name:

```
SQL> SET FLAGS 'STRATEGY';
  or
$ DEFINE RDMS$SET_FLAGS
  "STRATEGY"
```

After doing so, queries you execute will show the traditional (undetailed) optimizer query strategies.

To enable detailed strategy output, include the DETAIL keywords as shown here:

```
SQL> SET FLAGS 'DETAIL,STRATEGY';
  or
$ DEFINE RDMS$SET_FLAGS
  "DETAIL,STRATEGY"
```

The keywords DETAIL and DETAIL(1) are synonymous. DETAIL(2) is meant to invoke a greater level of detail. At present, there is nothing in Rdb that relies on more than one level of detail, so DETAIL(2) is treated the same as DETAIL(1).

```
SQL> SHOW FLAGS;

 Alias RDB$DBHANDLE:
 Flags currently set for Oracle Rdb:
   STRATEGY,PREFIX,
   MAX_RECURSION(100),
   DETAIL_LEVEL(1)
```

To disable detailed strategy output:

```
SQL> SET FLAGS 'NODETAIL,NOSTRATEGY';
  or
$ DEASSIGN RDMS$SET_FLAGS
```

In interactive SQL, DETAIL(0) is equivalent to NODETAIL. If you use the RDMS$SET_FLAGS logical instead, deassigning the logical name disables all flags that had been previously set using that logical name.

Practical Uses for Detailed Strategy Output

Study of the optimizer detailed strategy may help you to better understand query performance, wrong results, or incorrect sort order of the results. Here are a few tips that might get you started in the right direction.

Poor Performance:

If you have a query which is performing poorly, consider doing indexed retrieval in place of sequential retrieval. Indexed retrieval is not always better, so make certain you understand the conditions under which the query is being executed.

If a full scan of a large index is being done, perhaps a rewrite of the query or change in index definition will result in a limited index scan instead. Continuing with this idea, if all the columns referenced for a given table in the query appear as segments of an index, then in many cases all the information can be retrieved directly from the index and you can avoid having to fetch the data rows themselves.

If the strategy shows that Sorting is being performed, that's a costly step you might be able to avoid. One way is to define an index in which the data are already sorted in the desired order.

Wrong Order:

If a query is returning results in the wrong sort order, you can look at the detailed strategy to see what sorting if any is done and the sort keys used. Rdb attempts to

eliminate unnecessary sort operations in queries. For example, if you select data through a view and then specify an ORDER BY of the results, and if the view also contains an ORDER BY or GROUP BY clause, Rdb might be able to eliminate one of the sorts. A sort can also be eliminated if the data are retrieved via a sorted index. When looking at the detailed strategy report, check if a sort operation is performed using incorrect sort keys or if the sort is done in the wrong place in the query execution tree structure.

Wrong Results:

If a query returns wrong results, examination of the query strategy in detail might show you something otherwise overlooked. The problem is too broad to give you any specific recommendation on what to look for if the problem is in how the query was formulated. It is worth knowing that Rdb, to improve query performance, sometimes moves filter expressions (e.g., WHERE x='a') to lower branches of the query execution tree. This is done to eliminate data rows as soon as possible. For example, if one can reduce the number of rows from each branch of a join operation rather than after the join has been performed, the number of joins can be reduced. Study the placement of these filter expressions. If they are placed incorrectly in or are missing from the query execution tree, wrong results might ensue.

Another example of wrong results involves a match join strategy. Wrong results might be due to one of the inputs to the join being sorted in some order that does not agree with the order required for matching of the join keys.

Example 2

This next example shows much of the new, detailed notation. It is based on a query on Rdb's PERSONNEL database. Additional indexes and a view definition (not shown) were added in order to arrive at the given optimizer strategy. Some rows in the results were omitted to shorten the output. The example, below, has key numbers in the right margin. You can find an explanation for each key element of the strategy following the detailed strategy output. Further reference information appears in Table 1.

```
SQL> ! List all employees whose names begin with M-N-O-P and who
SQL> ! currently live in some New England state. For the employee's
SQL> ! state of residence, list the names of colleges from which s/he
SQL> ! received degrees.
SQL>
SQL> select
cont> cast (e.state as char(2)) as st,
cont> cast (e.employee_id as char(5)) as empid,
cont> cast (trim(trailing ' ' from e.last_name)||', '||e.first_name
cont> as char(19)) as full_name,
cont> cast (cd.college_code as char(4)) as coll
cont> from employees e left outer join college_degrees_view cd
cont> on cd.state = e.state
cont> where e.last_name between 'M' and 'P' and
cont> e.state in ('CN', 'MA', 'ME', 'NH', 'RI', 'VT')
cont> group by e.state,e.last_name,e.first_name,e.employee_id,
cont> cd.college_code;
```

```
Tables:                                                                  1
        0 = EMPLOYEES                                                    1
        1 = COLLEGES                                                     1
        2 = DEGREES                                                      1
Reduce: 0.STATE, 0.LAST_NAME, 0.FIRST_NAME, 0.EMPLOYEE_ID,               2
        2.COLLEGE_CODE                                                   2
Sort: 0.STATE(a), 0.LAST_NAME(a),                                        3
      0.FIRST_NAME(a),                                                   3
      0.EMPLOYEE_ID(a),2.COLLEGE_CODE(a)                                 3
Cross block of 2 entries (Left Outer Join)
  Cross block entry 1
    Leaf#01 BgrOnly 0:EMPLOYEES Card=100                                 4
    Bool: (0.LAST_NAME >= 'M') AND (0.LAST_NAME <= 'P') AND              5
          ((0.STATE = 'CN') OR (0.STATE = 'MA') OR                       5
          (0.STATE = 'ME') OR (0.STATE = 'NH') OR                        5
          (0.STATE = 'RI') OR (0.STATE = 'VT'))                          5
    BgrNdx1 EMP_INDEX2 [(2:2)6] Fan=11
      Keys: r0: (0.STATE = 'CN') AND (0.LAST_NAME >= 'M') AND            6
                (0.LAST_NAME <= 'P')                                     6
            r1: (0.STATE = 'MA') AND (0.LAST_NAME >= 'M') AND            6
                (0.LAST_NAME <= 'P')                                     6
            r2: (0.STATE = 'ME') AND (0.LAST_NAME >= 'M') AND            6
                (0.LAST_NAME <= 'P')                                     6
            r3: (0.STATE = 'NH') AND (0.LAST_NAME >= 'M') AND            6
                (0.LAST_NAME <= 'P')                                     6
            r4: (0.STATE = 'RI') AND (0.LAST_NAME >= 'M') AND            6
                (0.LAST_NAME <= 'P')                                     6
            r5: (0.STATE = 'VT') AND (0.LAST_NAME >= 'M') AND            6
                (0.LAST_NAME <= 'P')                                     6
      Bool(Range): r0: 0.STATE = 'CN'                                    7
                   r1: 0.STATE = 'MA'                                    7
                   r2: 0.STATE = 'ME'                                    7
                   r3: 0.STATE = 'NH'                                    7
                   r4: 0.STATE = 'RI'                                    7
                   r5: 0.STATE = 'VT'                                    7
      Bool(Common): (0.LAST_NAME >= 'M') AND (0.LAST_NAME <= 'P')        8
  Cross block entry 2
    Conjunct: 1.COLLEGE_CODE = 2.COLLEGE_CODE                            9
    Match
      Outer loop (zig-zag)
        Leaf#02 Sorted 1:COLLEGES Card=16                                4
        Bool: 1.STATE = 0.STATE                                          5
        FgrNdx COLL_COLLEGE_CODE [0:0] Fan=17
        BgrNdx1 COL_INDEX1 [1:1] Fan=15
          Keys: 1.STATE = 0.STATE                                       10
      Inner loop (zig-zag)
        Get Retrieval by index of relation 2:DEGREES                     4
          Index name DEG_COLLEGE_CODE [0:0]
```

```
ST    EMPID    FULL_NAME             COLL
MA    00435    MacDonald, Johanna    HVDU
MA    00435    MacDonald, Johanna    MIT
MA    00232    McElroy, Mary         HVDU
MA    00232    McElroy, Mary         MIT
      …        …
NH    00168    Nash, Norman          NULL
NH    00183    Nash, Walter          NULL
NH    00199    Nunez, Larry          NULL
NH    00190    O'Sullivan, Rick      NULL
18 rows selected
```

Explanation of keyed lines:

1. The detailed strategy for every query begins with a list of tables to be joined. The numbers are called 'context numbers' and are shorthand notations for representing each table.

2. The list of columns following the Reduce keyword comes from the GROUP BY clause in the query. The results are reduced by eliminating duplicate values for the given set of columns taken in combination.

3. The list of columns following the Sort keyword comes from the GROUP BY clause. The ordering of the results by the given keys is a precursor to the operation of reducing the rows to unique values on those columns.

4. In the detailed strategy the table names are preceded by their context numbers. This may seem uninteresting except for cases where the same table is used more than once.

5. The Bool keywords following the dynamic Leaf#nn notation define expressions for filtering the results of each dynamic leaf data retrieval.

6. The Keys notation here follows a background index which has six index key ranges. The keys for each index range, r0, r1, etc., are listed. The Keys notation does not appear in an un-detailed strategy report.

7. The background index (EMP_INDEX2) preceding this Bool(Range) notation has a filter expression which can be applied to the index range entries without having to access the rows from the table. Such a filter expression is called a Key-only Boolean. A range list index retrieval can have a Key-only Boolean with two parts: one part applies only to a specific index range, and a common part applies to each index range. The entire filter expression for a given range is the AND of the range-specific Boolean expression and the common Boolean filter expression.

8. This is the common Boolean part of the Key-only Boolean expression as described in note 7.

9. A Conjunct is a filter expression which consists of one or more predicates. It is like a Key-only Boolean filter with the exception that it is performed after rows of data from the table have been retrieved.

10. The Keys notation here follows a single-range index retrieval. Unlike the Keys in item 6, which show multiple index ranges, this single-range case does not use the r0, r1, notation to distinguish one range from the next.

Example 3

Here is an example derived from a customer bug report. The following query should have returned 3 rows. Instead it returned 6 rows.

```
select col1 from
    (select t2.A as col1,
            t2.B as col2,
            t2.C as col3
    from table1 t1, table2 t2
    where t1.id = t2.id) as
    vt (col1, col2, col3)
where
    vt.col3 > 0 and
    vt.col2 >= 0 and
    (vt.col1 <= 3 or 'x' = 'y');
```

The detailed strategy for this query was as follows. If you examine the strategy you'll see that the predicates (vt.col1 <= 3 or 'x' = 'y') are missing from the inner loop of the match join. In their place for the index key Boolean condition is an error message. For this example, the Rdb optimizer incorrectly generated the keyonly Boolean code for index TABLE2_IDX with no conditional expression (no predicates). Do not expect to always see an error message in the detailed strategy output. Sometimes the problem might be that the Boolean expression appears in the wrong place in the strategy output.

```
Tables:
  0 = TABLE1
  1 = TABLE2
Merge of 1 entries
  Merge block entry 1
    Conjunct: 0.ID = 1.ID
    Match
      Outer loop (zig-zag)
        Index only retrieval of relation 0:TABLE1
          Index name TABLE1_IDX [0:0]
      Inner loop (zig-zag)
        Conjunct: (1.C > 0) AND (1.B >= 0)
        Get Retrieval by index of relation 1:TABLE2
          Index name TABLE2_IDX [0:0]
          Bool: <error: common keyonly boolean no predicates>
       ID
       1
       2
       3
       4
       5
       6
6 rows selected
```

After the error was corrected, the detailed strategy for the query appeared as follows. The changes are in both the Conjunct and the Bool portion of the Inner loop for the match join.

```
Tables:
  0 = TABLE1
  1 = TABLE2
Merge of 1 entries
  Merge block entry 1
    Conjunct: 0.ID = 1.ID
    Match
      Outer loop (zig-zag)
        Index only retrieval of relation 0:TABLE1
          Index name TABLE1_IDX [0:0]
      Inner loop (zig-zag)
        Conjunct: (1.C > 0) AND (1.B >= 0) AND ((1.A <= 3) OR ('x' = 'y'))
        Get Retrieval by index of relation 1:TABLE2
          Index name TABLE2_IDX [0:0]
          Bool: (1.A <= 3) OR ('x' = 'y')
        ID
        1
        2
        3
3 rows selected
```

Strategy Output Notation

Now that you have a general idea of what you might see in a detailed strategy report, here is a table that lists both the old and new strategy notation. The old notation comes from the *Rdb7 Guide to Performance and Tuning, Section C.1* and *Table C-2 Output Definitions for the S Flag*. Remember, you cannot see detailed strategy output simply by defining RDMS$DEBUG_FLAGS S. Instead, you must define RDMS$SET_FLAGS, as explained earlier, or use the interactive SQL statement SET FLAGS.

When normal (un-detailed) strategy is requested, you will see only the keywords as they appear in the *Notation* column of Table 1, below. When detailed strategy is enabled, the keywords will appear followed by a colon and the details for each keyword. Examples showing the details appear in the *Description and Examples* column of Table 1.

Table 1: Output Definitions for the STRATEGY and DETAIL,STRATEGY Flags

Notation	Description and Examples

Aggregate	Indicates use of a statistical function, such as, COUNT, SUM, AVG, MIN, MAX, STDDEV, or VARIANCE. Example of detailed strategy, which can include a Boolean filter expression:
	
	`Aggregate: COUNT (0.X)`
	`  Bool: NOT MISSING (0.X)`
	
	The Aggregate keyword can represent more than one aggregate operation in the query. When detailed strategy is requested, these aggregate types will be listed one after the other.
	
	In versions of Rdb after V7.0-62 and V7.1.0.0, the notation will include a number, such as 3:, to identify each aggregate function, as in the following:
	
	`Aggregate: 3:COUNT (0.X)`
	
	Elsewhere in the query strategy, notation such as <agg3> will refer to the results of the aggregate operation.

Aggregate-F1	Indicates a check for the existence of a single value, such as a query containing EXISTS or ANY. Example of detailed strategy: `Aggregate-F1: COUNT-ANY (0.X)`
Aggregate-F2	Indicates a check for uniqueness. Used for SQL queries containing SINGLE. Example of detailed strategy: `Aggregate-F2: COUNT-SINGLE (0.X)`
<agg*n*>	When detailed strategy is requested, a term such as <agg3> represents the result of aggregate function number 3 (see description of Aggregate). For instance: `Conjunct: 0.SALARY_AMOUNT > <agg3>`
BgrNdx*n*	Indicates background index number *n,* e.g, BgrNdx1.
BgrOnly	Indicates the background only leaf retrieval type.
Bool	Boolean filters indicate processing of WHERE predicates. They can appear as part of an Aggregate, RLEAF, or index retrieval operation. When shown below an index retrieval, Bool signifies key-only Boolean optimization. The optimizer uses this method to filter out dbkeys before fetching rows, thus saving I/O operations. Example of detailed strategy: `Bool: 1.STATUS = 'B'`

Notation	Description and Examples
Bool(Common)	This notation is used only for detailed strategy for an index key Boolean where a range list is used. For range list retrieval, the keyonly Boolean can consist of two parts, a common part which applies to each range of the index, and a range-specific part (see next section). For example: `Bool(Common): 0.X = Y` The 0.X=Y predicate applies to each index range.

Bool(Range)	This notation is used only for detailed strategy for an index key Boolean where a range list is used. For range list retrieval, the keyonly Boolean can consist of two parts, a common part which applies to each range of the index (see previous section), and a range-specific part. For example:
	
	`Bool(Range): r0: 0.X = 3`
	`r1: 0.X > 7`
	
	Each listed Boolean range (r0, r1, etc.) applies only to a single index range. The predicate 0.X = 3 is applied only to the first range for the index. The predicate 0.X > 7 is applied only to the second range. If there is also a common part to the index key Boolean (see previous section), the common Boolean expression is effectively ANDed with each range-specific predicate.
	
	Note that there is no correspondence necessarily between the index ranges shown in the index segment notation, such as, [1:0,1:1], and the index key range Booleans which might follow. That's because the index range segments are sorted and compressed into notation (described farther on) such as [*(l:h)n*].

Card=*n*	Indicates table cardinality stored in the column RDB$CARDINALITY in the system table RDB$RELATIONS.
Cross block of *n* entries	Indicates a cross (or nested loop) join method for *n* entries.
Direct Lookup	Indicates that the index used has no duplicates and that an exact key match predicate is used for retrieval, which returns one or zero dbkeys.
<error: ...>	This notation is used during detailed strategy output whenever some internal error is detected. The text of the error message gives some clue to the nature of the problem.
Fan = *n*	Indicates the average fanout factor of an index B-tree node based on the B-tree node size, index key length, and the initial percent fill of the index node. The fanout of a B-tree node is the number of child nodes (branches) attached to a given node. The higher the average fanout, the fewer levels the B-tree contains, which promotes faster single key access but greater potential for deadlock in a multiuser environment.
FFirst	Indicates the fast first leaf retrieval type.

Notation	Description and Examples
FgrNdx	Indicates a foreground index.
Firstn	Indicates use of a LIMIT TO *n* ROWS clause. This information is not used by the optimizer as part of the optimization process, but rather as a limitation of the output from the query.
Get	Indicates execution of an I/O operation for data record retrieval.
Index only retrieval of relation	Indicates that the requested information was retrieved from within the index. No data record access was required.

Keys	The Keys notation appears only when detailed strategy is produced. It lists the keys used for index retrieval. This first example is for a regular (non-range-type) index retrieval using two index segments:
	
	`Keys: (0.WAREHOUSE = '451') AND`
	`(1.STOCK_NUMBER = '7075002')`
	
	Keys for range list index retrieval are listed separately for each range (r0, r1, etc.):
	
	`Keys: r0: 0.EMPLOYEE_ID = '00164'`
	`r1: 0.EMPLOYEE_ID = '00200'`

Leaf#01	Indicates the first dynamic leaf optimization node in the execution tree. Subsequent leaf nodes in the same tree will have different numeric identifiers. This use of the term leaf is distinct from its use in partitioned scan notation (see that notation farther on).
<mapped field>	This notation appears only in detailed strategies. It shows an indirect reference to a column which cannot be traced back to a single base table column. For example, you might have a view that is a UNION of two tables. The view columns are like those of the underlying tables, but they are not exactly the same because they represent a different set of data.
Match	Indicates a match join method.
Max key lookup Min key lookup	Indicates a direct index key lookup instead of an entire index scan. Used when the optimizer can use index only retrieval and the query contains the MAX or MIN statistical function.
Merge of *n* entries	Indicates use of the merge strategy to return rows from multiple tables. In the merge strategy, rows from different tables are merged and delivered to the user.
NdxOnly	Indicates the index only leaf retrieval type.

Notation	Description and Examples
OR index retrieval	Indicates static OR optimization using a hashed index or multiple sorted indexes to retrieve data from a single table. Rows selected by each index are delivered one after another. Duplicate dbkeys are discarded by applying Conjunct, so that the same row is not delivered twice.
(partitioned scan#*n*)	Indicates a particular instance of a sequential scan of one partition of a strictly-partitioned table. The number *n* is the leaf (or instance) number for the sequential scan. This use of the term leaf is distinct from its use in the dynamic leaf optimization retrieval strategies (see Leaf#01 notation).
Reduce	Indicates elimination of duplicate rows based on the values of one or more columns. Found in queries using the DISTINCT or UNION DISTINCT operators. A sort is frequently part of the duplicate elimination process and the Sort notation is therefore often included in the output. Example of detailed strategy: `Reduce: 0.LAST_NAME, 0.FIRST_NAME`
Retrieval by DBK of relation	Indicates that the requested data was retrieved using direct dbkey access.
Retrieval by index of relation	Indicates use of an index to retrieve data.
Retrieval sequentially of relation	Indicates that data retrieval is done without the use of an index. Access to the data is performed by a sequential scan of the table.
Reverse Scan	Indicates that the index is scanned from back to front. In Rdb, scanning an index in reverse order typically requires more I/O operations than does scanning the index in the forward direction.
Sort	Indicates the requested data had an output order specified, that a sort was done on behalf of a match join strategy, or that a sort was required for a DISTINCT or UNION DISTINCT operation. In the detailed strategy, the Sort keyword is followed by the list of sort keys. Each key is shown as either ascending (a) or descending (d). For example: `Sort: 0.NAME(a), 1.SCORE(d)`
Sorted	Indicates the sorted order leaf retrieval type.

Notation	Description and Examples

<subselect>	In the detailed strategy, the term <subselect> is used in two circumstances. The first is as the argument of an aggregate function, as in,
	
	`Aggregate: 0.COUNT(<subselect>)`
	
	The portion of the strategy which implements the subselect expression immediately follows the Aggregate line(s) in the output.
	
	The second circumstance is when a reference is made to the result of some aggregate operation. Example:
	
	`Conjunct: <subselect> = 0`
	
	In versions of Rdb after V7.0-62 and V7.1.0.0, the notation used in this second circumstance will change to directly reference the aggregate results, as in,
	
	`Conjunct: <agg3> = 0`
	
	See the description of Aggregate for further information.

Tables	This notation only appears in detailed strategies. It lists each table used in the query and assigns each a shorthand 'context' number. For example:
	
	`Tables:`
	`  0 = EMPLOYEES`
	`  1 = DEGREES`
	`  2 = EMPLOYEES`
	
	The context numbers are subsequently used in the detailed strategy preceding table names and column names. For example:
	
	`Index only retrieval of relation 2:EMPLOYEES`
	
	A table name is separated from its context number by a colon. By contrast, a column name is separated from its context number by a period. For example:
	
	`Conjunct: 0.EMPLOYEE_ID = '00164'`

Temporary Relation	Indicates creation of a temporary table to store intermediate results. The temporary table can exist either in memory or on disk.

Notation	Description and Examples
<var> <var*n*>	This notation appears only in detailed strategies. It indicates the use of a declared variable. For example: Conjunct: <var> = '100' In versions of Rdb after V7.0-62 and V7.1.0.0, variables will be distinguished one from the other by number, as in, Conjunct: <var1> = <var3>
(zig-zag)	Indicates a variation of the match retrieval strategy in which keys are skipped when switching retrieval from one leg of the Match join to the other.
[*l:h*]	Indicates the number of low index key (low Ikey) segments and high index key (high Ikey) segments in an index key range. *l* represents the number of low Ikey segments and *h* represents the number of high Ikey segments for the given index range. ● The notation [0:0] indicates that a full scan of the index is done. ● The notation [1:1] indicates there is one low index key segment and one high index key segment in the index key range. Sometimes this indicates the presence of an equality predicate in the query, such as EMPLOYEE_ID = '00164'. In this case the range is a single value. ● The notation [1:0] indicates there is one low index key segment but no high index key segment. That is, the range of index keys to be scanned has a starting point, such as, EMPLOYEE_ID > '00164', but no upper limit. ● The notation [0:1] indicates there is one high index key segment but no low index key segment. That is, the range of index keys to be scanned starts at the beginning of the index and ends at the high index key bound, such as, EMPLOYEE_ID < '00164'.

[*(l:h)n*]	Indicates OR optimization in which the optimizer uses a single sorted index to locate two or more ranges of data rows. *l* represents the number of low Ikey segments in the range; *h* represents the number of high Ikey segments in the range; and, *n* represents the total number of ranges. For example:
	
	`WHERE EMPLOYEE_ID IN ('00164', '00177', '00200')`
	
	`Leaf#01 FFirst EMPLOYEES Card=100`
	`BgrNdx1 EMP_EMPLOYEE_ID [(1:1)3]`
	
	In this case, the three ranges correspond to the three employee ID numbers.

Available in Rdb 7.0.6.2 and Rdb 7.1

The ability to enable detailed strategy output exists as early as Rdb 7.0.6. However, for that version the capability was highly experimental and had a number of deficiencies. The versions considered good enough for general use are Rdb 7.0.6.2 and 7.1.

Work in Progress

The addition of the detailed strategy dump was done gradually over the period of a year as the need arose in engineering to see details of the query strategy to make problem diagnosis easier. More remains that could be done. If you wish to make a case for further changes, please post an enhancement request, component Rdb Optimizer, and reference the detailed strategy output. Also, please provide a small sample database and representative queries.

Jean-Claude Proteau is a Principal Member of Technical Staff within Oracle Rdb Engineering located in Oracle's New England Development Center in Nashua, New Hampshire, USA. Jean-Claude, who prefers to be called Claude, has worked with Rdb since 1985 as a developer of what is today the Replication Option for Rdb. Claude joined the Rdb Engineering Group in 1996 to continue support of the Replication Option for Rdb and later to become a member of the Rdb Optimizer team.