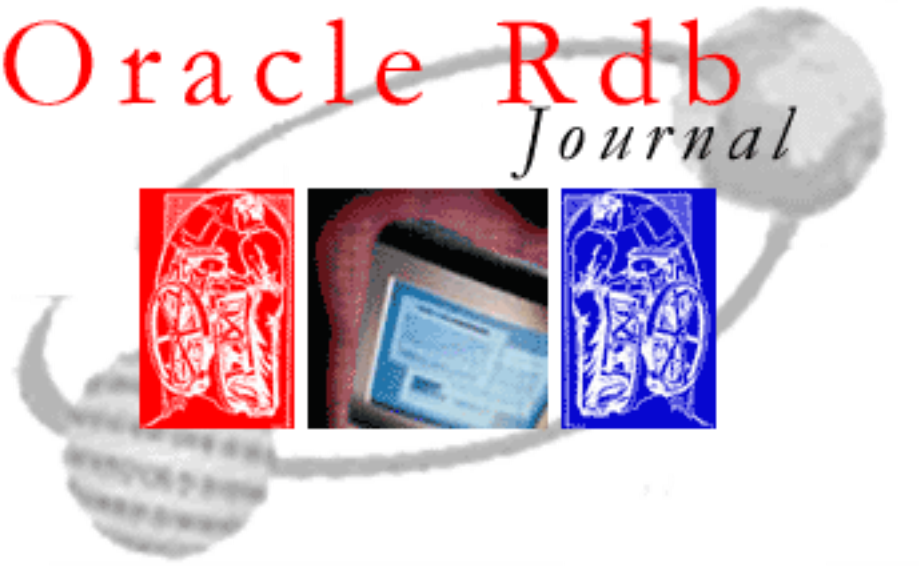

DISTINCT, derived tables, GROUP BY and Sort

A Technical Corner article from the Rdb Journal
By Ian Smith
January 15, 2000

Rdb Journal technical corner

Feature Spotlight: DISTINCT, derived tables, GROUP BY and Sort

Ian Smith, Consulting Engineer
ismith@us.oracle.com

January 15, 1999

The Rdb Technical Corner is a regular feature of the Oracle Rdb Web Journal. The examples in this article use SQL language from Oracle Rdb V6.0 and later versions.

Over the last twenty years or so, I have seen my share of applications written for relational databases, along with many styles and flavors of SQL queries. I am convinced that of all the components of the SELECT statement, the DISTINCT modifier causes the most confusion. By the way, the HAVING clause runs a close second, but that will have to be the topic of a future Technical Corner.

DISTINCT is a modifier for the SELECT list. It is the opposite of the ALL modifier. Which raises an interesting question, when was the last time you used ALL in a SELECT statement? As the default, most programmers never bother to use it.

```
SQL> select all * from work_status;
 STATUS_CODE    STATUS_NAME    STATUS_TYPE
 0              INACTIVE       RECORD EXPIRED
 1              ACTIVE         FULL TIME
 2              ACTIVE         PART TIME
3 rows selected
```

DISTINCT modifies the "select" action so that it returns the unique values for all the columns (or value expressions) listed. This certainly sounds simple enough, but it can be hazardous in the hands of the novice or unprepared SQL programmer.

From time to time while performing my job as a development engineer on the Oracle Rdb product, I see queries from real customers from real production systems which strike me as queries which that should be rewritten. I should add that they come my way when the customer is concerned about performance.

This poor style is partly the fault of the SQL language which allows programmers to get the same results from many different types of query which leads to some poor programming practices.

In this technical corner I'd like to show an inefficient query that uses DISTINCT as an expensive tool to solve an easy small problem. This same problem has passed across my desk (well, across my workstation screen) about once a year in the last 15 years. I hope that, by describing this problem, I see fewer reports

of the problem, which I hope means that many queries have been improved.

The Problem Query

Have you even seen queries like the one below? I was presented with a problem query very much like this quite recently. I have coded a similar query using the PERSONNEL database so that readers can confirm my findings.

```
SQL> select distinct (employee_id),
cont>        (select last_name
cont>         from employees e
cont>         where e.employee_id = jh.employee_id)
cont> from job_history jh
cont> where employee_id < '00166';
 EMPLOYEE_ID
 00164          Toliver
 00165          Smith
2 rows selected
```

There are some points to note about this query. I suspect the SQL programmer believed that DISTINCT was a function. The enclosing parentheses around the column name is normal syntax allowed for any value expression--but it may trick the programmers into thinking they have restricted the DISTINCT to just the one column.

However, the effect of this query is that Rdb must eliminate duplicate values for the EMPLOYEE_ID and also the subselect on LAST_NAME. Does this really make sense for this simple query? I know that the LAST_NAME is derived from the unique EMPLOYEE_ID, so there is no need to apply sorting and reduction to the subselect result. However, the instructions are clear to Rdb, even if they are were not for the programmer.

So what does the optimizer make of this query?

```
Reduce  Sort
Cross block of 2 entries
  Cross block entry 1
    Cross block of 2 entries
      Cross block entry 1
        Index only retrieval of relation JOB_HISTORY
          Index name  JOB_HISTORY_SORT [0:0]
      Cross block entry 2
        Aggregate       Get     Retrieval by index of relation EMPLOYEES

          Index name  EMPLOYEES_HASH [1:1]       Direct lookup
  Cross block entry 2
    Aggregate       Get     Retrieval by index of relation EMPLOYEES
      Index name  EMPLOYEES_HASH [1:1]       Direct lookup
```

Amazingly, we have three table accesses in the strategy when I really expected just two. This is because SQL separates the DISTINCT into two pieces: one to reduce the rows to the unique set of values and one

to calculate the returned results.

Is there a better way to write this query?

The other relational interface provided with Rdb is called RDO. (Bear with me on this: I will not suggest that folks start writing RDO queries. I shall not get into arguments over the merits of the SQL Language: it is here to stay.) The RDO language was developed at a time when there was no accepted standard relational language. Most relational vendors had their own languages that provided selection, ordering, and the basic relational operators. When SQL was adopted as an international standard, RDO language development ceased. However, it continues to be maintained and is still widely used. The observant RDO user might even notice a few new features added from time to time.

RDO doesn't have a DISTINCT modifier, but rather a REDUCED TO clause similar in function to the SQL ORDER BY or GROUP BY clauses. This allows the selected columns and expressions to be kept separate from the distinct, or reduction, operation. Here is our query written using the RDO language.

```
RDO> for jh in job_history
cont>     with jh.employee_id < '00166'
cont>     reduced to jh.employee_id
cont>
cont>     print jh.employee_id,
cont>           first e.last_name
cont>             from e in employees
cont>             with e.employee_id = jh.employee_id
cont>
cont> end_for
 EMPLOYEE_ID
 00164          Toliver
 00165          Smith
```

As a brief word of explanation: in RDO, the FOR loop selects the rows; the WITH clause is just the same as a SQL WHERE clause; the FIRST FROM clause performs the same function as the SQL subselect that we observed earlier; and the PRINT statement displays the result interactively.

The optimizer strategy is just what I expected: just two table accesses.

```
Reduce
Cross block of 2 entries
  Cross block entry 1
    Index only retrieval of relation JOB_HISTORY
      Index name  JOB_HISTORY_SORT [0:1]
  Cross block entry 2
    Aggregate        Get     Retrieval by index of relation EMPLOYEES
      Index name  EMPLOYEES_HASH [1:1]        Direct lookup
```

Also note that there is no "Sort" present. This is because the Optimizer could use the sorted index JOB_HISTORY_SORT directly. In this case, the RDO language allowed the programmer to represent the query more precisely than DISTINCT allowed in SQL.

Back to our SQL problem

So we have seen that this small misunderstanding and our poor SQL coding has forced Rdb to do more work than is necessary to deliver the rows. We need to rework the query to do at least as well as RDO.

We need to keep the subselects out of the DISTINCT select list, since by including them we are forcing a real sort of the data. By its nature, sort is CPU- and I/O-intensive, as it must reorder a set of rows by comparing values from every row. The RDO solution was much more efficient because it could use the pre-defined order in the index.

The actual problem presented to me contained 60 subselects in the SELECT DISTINCT clause, which meant that there were 121 separate table accesses in that single query. The customer had exceeded the Rdb limit of 32 table contexts. Fortunately Oracle Rdb7 had raised the limit, but the query didn't perform very well.

Can we find solutions to this class of problems? Our goal is to use DISTINCT only on those columns that need to be reduced to a unique subset. This will result in lower sort overhead (shorter records to sort means less virtual memory and smaller temporary files). We also hope to encourage the Optimizer to use a sorted index to avoid the sort completely.

Applying DISTINCT to a minimal column list could be done using a view or a derived table, as shown in this example.

```
SQL> select employee_id,
cont>         (select last_name
cont>          from employees e
cont>          where e.employee_id = jh.employee_id)
cont> from (select distinct employee_id
cont>       from job_history
cont>       where employee_id < '00166') jh (employee_id);
 EMPLOYEE_ID
 00164          Toliver
 00165          Smith
2 rows selected
```

The FROM clause contains a table expression rather than a table name. This expression delivers the unique values we want from the JOB_HISTORY table to the outer query, which calculates the result values.

The Optimizer strategy looks very close to the RDO result: just two table accesses.

```
Cross block of 2 entries
  Cross block entry 1
    Merge of 1 entries
      Merge block entry 1
      Reduce  Index only retrieval of relation JOB_HISTORY
        Index name  JOB_HISTORY_SORT [0:1]
  Cross block entry 2
    Aggregate        Get     Retrieval by index of relation EMPLOYEES
      Index name  EMPLOYEES_HASH [1:1]        Direct lookup
```

The "Merge block" is an artifact of the derived table and is simply a mechanism to deliver the results to the outer query.

Applied to the customer's problem, this technique reduced the table accesses to just 61. We then proceeded to replace several of these subselects with other derived tables, so that the number of tables was further reduced. The resulting query ran on prior versions of Rdb, as it used less than 32 table contexts. Performance was pretty good too!

Well, are there any other solutions?

There are certainly other ways to solve this query. Most readers are probably wondering why I didn't immediately use GROUP BY. I did this partly to emphasize the separation of selection from reduction, but also to promote derived tables as a viable solution for these types of problem.

Here is our problem query coded using GROUP BY.

```
SQL> select employee_id,
cont>        (select last_name
cont>         from employees e
cont>         where e.employee_id = jh.employee_id)
cont> from job_history jh
cont> where employee_id < '00166'
cont> group by employee_id;
 EMPLOYEE_ID
 00164          Toliver
 00165          Smith
2 rows selected
```

As you can see, the result is also very close to that from RDO.

```
Cross block of 2 entries
  Cross block entry 1
    Reduce  Index only retrieval of relation JOB_HISTORY
      Index name  JOB_HISTORY_SORT [0:1]
  Cross block entry 2
    Aggregate        Get     Retrieval by index of relation EMPLOYEES
      Index name  EMPLOYEES_HASH [1:1]        Direct lookup
```

SQL certainly provides many ways of solving the same problem. The goal is to choose the solution

wisely.

Other complex expressions to remove

This article has targeted the subselects within a SELECT DISTINCT statement, because they do I/O and therefore are usually the most expensive components of the query. However, there are other complex expressions that should be removed from the DISTINCT clause, in order to reduce the evaluation overhead of the query. These include CASE, COALESCE, NULLIF, NVL, DECODE, calls to user-defined functions, and other complicated mathematical, date, or string expressions.

Technical Tips: Avoiding Sort

Examining the Optimizer strategy is one way to see whether a sort is being performed. However, there are are some standard operators that will often cause a sort to occur. You should keep these in mind when tuning queries and tuning sort for Oracle Rdb.

- As described in this article, DISTINCT will often cause sort to be used so that rows with the same values can be eliminated.
- GROUP BY and ORDER BY will order the set of data using sort if no suitable index is available. This is always true when these clauses use columns from different tables or when they contain expressions.
- The UNION operator, by default, returns only the distinct rows from the merged tables. If you know that the results are already unique, or if you can tolerate the duplicate rows, use the UNION ALL operator.
- Joins between tables with no suitable sorted index may require a sort so that the matching rows can be found and returned for the query.

The Rdb optimizer Optimizer will attempt to eliminate unnecessary sorts if possible. For instance, an exaggerated query that over-specifies the ordering will still only perform one sort operation.

```
SQL> select distinct job_start
cont> from job_history
cont> group by job_start
cont> order by job_start;
Reduce  Sort     Get     Retrieval by index of relation JOB_HISTORY
  Index name  JH_EMPLOYEE_ID [0:0]
 JOB_START
  1-Jul-1975
 10-Sep-1975
  .
  .
  .
```

If a suitable index is available, then even this will disappear.

```
SQL> select distinct job_start
cont> from job_history
cont> group by job_start
cont> order by job_start;
Reduce  Index only retrieval of relation JOB_HISTORY
  Index name  JH_JOB_START [0:0]
 JOB_START
  1-Jul-1975
 10-Sep-1975
  .
  .
  .
```

As always, comments, corrections, suggestions, and interesting stories about Oracle Rdb are welcome.