Spotlight: Strictly Partitioned Tables

A Technical Corner article from the Rdb Journal

By Ian Smith

April 30, 2000

Spotlight: Strictly Partitioned Tables

In Rdb7, Oracle Rdb introduced strictly partitioned tables. The savings in I/O for sequential scans can be significant.

ismith@us.oracle.com

Ian Smith, Consulting Engineer

April 30, 2000

Oracle Rdb uses a special database object known as a STORAGE MAP to describe the physical distribution of table rows across multiple storage areas. This functionality was first made available in Rdb V3.0, released in 1986. The advantage of a storage map is that it allows very large tables to be spread across many physical devices to accommodate the data and to make use of device parallelism with asynchronous I/O.

However, the partitioning ranges for tables were only used at INSERT time. A subsequent UPDATE statement may alter the values of the partitioning columns. Rdb does not move the data in this case because the row may be referenced by indices or other applications. As there was no guarantee that the data still obeyed the original partitioning scheme, the Oracle Rdb Optimizer did not make use of the partitioning scheme to eliminate area scans.

When a table's storage map has the attribute PARTITIONING IS NOT UPDATABLE, then mapping of data to a storage area is strictly enforced. This is known as **strict partitioning**. This feature was introduced with Oracle Rdb7.

In this case the partitioning columns may not be updated and the Rdb optimizer uses this knowledge when optimizing sequential queries of the table. The savings in I/O for sequential scans can be significant.

Technical Tips: Using Strictly Partitioned Tables

Strictly partitioned indices have also been supported by Oracle Rdb since Rdb V3.0. In general, most production systems will use indices as supporting structures for query retrieval. However, strict partitioning is often useful when queries need to perform sequential access to a table, such as during reporting, or during table maintenance.

How do I know if strict partitioning is working?

Use either the SET FLAGS statement or the logical RDMS$SET_FLAGS to enable the STRATEGY and EXECUTION tracing during query evaluation.

The Rdb Technical Corner is a regular feature of the Oracle Rdb Web Journal. (www.oracle.com/rdb/rdb_journal) The examples in this article use SQL language from Oracle Rdb7 and later versions.

If a table is strictly partitioned and the query allows the use of this feature then the STRATEGY dump includes the text "(partitioned scan#nn)" after the table name. The **#nn** indicates the leaf number for this sequential scan (there may be several within a single query). For instance:

```
select name
    from T
    where badge between 8 and 10;
~S#0045
Conjunct        Get     Retrieval sequentially of relation T (partitioned scan#1)
~E#0045.1: Strict Partitioning using 2 areas
        partition 9 (larea=57)
        partition 10 (larea=58)
  NAME
  i
  j
 2 rows selected
```

The EXECUTION dump displays the selected partitions for the current execution of the query.

How can multiple processes sequentially scan the same table?

By default, a sequential scan locks the entire table so that ISOLATION LEVEL SERIALIZABLE can be enforced. You can change the SET TRANSACTION statement to reduce the ISOLATION LEVEL to REPEATABLE READ, or READ COMMITTED. Now the table will be locked for shared read initially. If an UPDATE or DELETE is performed, then each partition will be locked as required.

If each process selects data from different partitions, then they will execute concurrently.

Can I perform PROTECTED or EXCLUSIVE access to a partition?

Starting with Rdb V7.0.1.3 a new PARTITION clause was added to the SET TRANSACTION ... RESERVING statement that allows the partition number to be locked for PROTECTED or EXCLUSIVE access by the transaction. This mechanism is used by RMU/LOAD/PARALLEL so that parallel executors can perform EXCLUSIVE loads of a partitioned table.

For applications which use strict partitioning to perform parallel maintenance, such as bulk loads, mass updates or row purging this can have many benefits. While each process works in parallel on parts of the table, each can reserve the table partitions for exclusive access and so avoid snapshot file I/O, use fewer locks and less virtual memory (which translates to lower page faulting).

```
SQL> set transaction read write
cont> reserving employees PARTITION (2)
cont> for exclusive write;
```

You can reserve more than one partition by listing the partition numbers separated by commas. Any indices which are identically partitioned will also be locked in the same way, all other indices are locked for SHARED access.

How does the WITH LIMIT clause work to partition the data?

The WITH LIMIT clause provides the upper limit of data which can be stored in the table (or index). Perhaps the easiest way to see this is to use the RMU/EXTRACT command to extract the STORAGE_MAP and INDEX items from the database. When you use /OPTION=FULL, RMU/EXTRACT annotates the storage map with the Boolean expression used by Rdb at runtime to select the partition. As more columns are added to partition the table the more complex this Boolean expression appears.

```
create storage map EMPLOYEES_MAP
    for EMPLOYEES
    placement via index EMPLOYEES_HASH
    store
        using (EMPLOYEE_ID)
        -- Partition:
        --         (EMPLOYEE_ID <= '00200')
            in EMPIDS_LOW
                with limit of ('00200')
        -- Partition:
        --         (EMPLOYEE_ID <= '00400')
            in EMPIDS_MID
                with limit of ('00400')
            otherwise in EMPIDS_OVER;
```

How do I force a query to be SEQUENTIAL?

Prior to Rdb V7.0.4 a query outline must be used to guarantee a sequential access strategy. The following example demonstrates the process. Use SET FLAGS 'OUTLINE' to generate a template outline for the query, also use SET NOEXECUTE to avoid actually fetching, deleting or updating rows that would normally be affected by the query.

```
SQL> set flags 'outline';
SQL> set noexecute;
SQL> select last_name, first_name, employee_id
cont>  from employees limit to 1 row;
-- Rdb Generated Outline :  1-MAY-2000 16:11
create outline QO_21FD853E88576893_00000000
id '21FD853E88576893F9E37D2F18EA64A2'
mode 0
as (
  query (
-- For loop
    subquery (
      EMPLOYEES 0 access path index EMP_EMPLOYEE_ID
      )
    )
  )
compliance optional     ;
0 rows selected
```

Notice that the access method was INDEX via the index EMP_EMPLOYEE_ID. Now reenabled execute, turn off outline generation and enable the strategy flag so we can see the result. Use the template and change the access method to SEQUENTIAL as shown in this example on the EMPLOYEES table.

```
SQL> set execute;
SQL> set flags 'nooutline, strategy';
SQL> create outline my_seq_query
cont> id '21FD853E88576893F9E37D2F18EA64A2'
cont> mode 0
cont> as (
cont>   query (
cont>     subquery (
cont>       EMPLOYEES 0 access sequential
cont>       )
cont>     )
cont>   )
cont> compliance optional     ;
SQL> select last_name, first_name, employee_id
cont>  from employees limit to 1 row;
~S: Outline "MY_SEQ_QUERY" used
Firstn  Get     Retrieval sequentially of relation EMPLOYEES
 LAST_NAME        FIRST_NAME   EMPLOYEE_ID
 O'Sullivan       Rick         00190
1 row selected
```

The strategy display clearly shows the query outline being used and also the modified access strategy.

Starting with Rdb V7.0.4 a new OPTIMIZE clause has been added to force the optimizer to use sequential access. This avoids creating a query outline for every ad hoc query or maintenance script.

```
SQL> select * from employees limit to 1 row;
Firstn  Get     Retrieval by index of relation EMPLOYEES
  Index name  EMP_EMPLOYEE_ID [0:0]
...
1 row selected
SQL> select * from employees limit to 1 row
cont> optimize for SEQUENTIAL ACCESS;
Firstn  Get     Retrieval sequentially of relation EMPLOYEES
...
1 row selected
```

Ian Smith is a Consulting Member of the Technical Staff of Oracle Corporation and a Technical Leader for the Oracle Rdb product engineering group located in Nashua NH, U.S.A.

Ian has been a member of the engineering team since Jan 1989 and has been working with Rdb since the beta testing of the first version in 1984. Between 1982 and 1989 Ian was a database consultant working with customers to build and tune Rdb and DBMS (CODASYL) systems.

Ian is currently the SQL language and relational interface architect for Oracle Rdb and has designed many of the new features incorporated into Rdb in the last decade.