Oracle Applications Labs

Oracle Applications Labs Best Practices:

Implementing Large-Scale Demantra Table Rebuilds To Improve Performance with Zero Downtime

An Oracle Technical White Paper
January 2013

ORACLE

Abstract

Oracle implemented Demantra and Value Chain Planning in 2011 for the manufacturing of all of its Sun related hardware and software and for Service Parts Planning. Demantra is used to forecast sales of Exadata and other Oracle hardware products as well as for Service Parts planning and fulfillment of service requirements all over the world.

Oracle's growing number of products and locations produced significant growth in Demantra table size and storage requirements. Periodic table rebuilds were required both to improve performance and reduce storage requirements. However, table rebuilds required taking Demantra offline for up to six hours at a time. This downtime impacted demand planners who needed around-the-clock access to Demantra to satisfy Oracle's global planning needs, and interrupted demand signals to internal and external manufacturers.

Oracle IT was able to eliminate almost all of this downtime by using a standard but little-known feature of the Oracle database, DBMS_REDEFINITION. This feature provides online table re-definition, allowing tables to be rebuilt for improved performance without subjecting users to any downtime. By using the DBMS_REDEFINITION package, Oracle IT also eliminated the need for downtime to make table structure modifications.

Oracle IT recommends that customers with similar large-scale Demantra implementations follow the directions outlined in this paper for using the DBMS_REDEFINITION package to rebuild their tables to achieve similar results without downtime.

Backed Against the Wall: Oracle’s Largest-Ever Internal Value Chain Planning Database Table

In the Spring of 2012, the Oracle Service Parts Forecasting database table was growing at a rate of two gigabytes per week. At this rate, this Demantra table of 800 gigabytes would grow to more than a terabyte within months. Oracle IT knew the table was a problem, because performance of the Service Parts Forecasting had noticeably degraded. However a Catch-22 prevented the team from getting a clear picture of the table’s health.

One indicator of overall table health is its row chaining percentage. A chained row is a row of data too large to fit into a single database data block. The data then carries over to the next block. The lower the row chaining percentage, the healthier the table. If a table’s row chaining percentage for a table is over 15%, the table should be rebuilt. To determine the row chaining percentage and other health characteristics of the Service Parts Forecasting database, the standard “Analyze Table” program would need to be run. However the program slowed performance of the table so much that analysis could not be run while the database was in production. In Oracle’s 24/7 world, it was always in production.

A table rebuild was absolutely necessary, even though the table analysis was not able to be run. There was little doubt that rebuilding the table – including truncating records to remove unnecessary older data – would improve performance. With appropriate table partitioning, the table could be even be rebuilt without rebuilding indexes and suffering the consequent row chaining. However while the end result was appealing, the path to get there was not.

The standard rebuild process starts with the extraction of the data from the table.   A conventional data extract of the table data should be able to take place while users are on the system. However attempting to extract from this large and fragmented table caused users to experience unacceptable delays. Worksheets that under normal circumstances would normally only take a few seconds to open suddenly took three to four minutes. This was unacceptable in such a heavily used system.

The data extract was only part of the problem. After extract, completely rebuilding the table would require taking it offline for up to one day. While this type of operation normally took place over a weekend, even weekend downtime was unacceptable to the Demand Planners and IT support staff. Taking the database offline meant global business process users could not access their information. The team looked for another solution.

An Online Solution

Confronted with the task of rebuilding Oracle's largest-ever internal table without taking it offline, the team considered creating a custom online rebuild tool. However discussions with other Oracle IT teams surfaced a standard but little-known database feature called DBMS_REDEFINITION as a potential solution. The DBMS_REDEFINITION package already existed in the Oracle Database (Oracle Database Administrator's Guide (10g Release 2 (10.2)) and could potentially enable table structure modifications "on-line" without significantly affecting the availability of the application.

How It Works

Broadly speaking, using the DBMS_REDEFINITION package requires many of the same steps as a conventional table redefinition. The steps for this are to (a) check the space requirements and create an interim table, (b) copy data to the interim table, (c) sync data to include latest updates, and (d) swap the two tables just as the conventional method. The difference with the DBMS_REDEFINITION is that the copy is faster and the package performs the sync of the newly available data since the application is not offline. In a conventional redefinition, there is no need to sync new data because no new data is being created while the table is offline.

Extensive Advance Testing

The team began testing the The DBMS_REDEFINITION package on smaller portions of the extracted data. Positive results from this initial testing encouraged the team to proceed to full-scale testing.

Full testing of the DBMS_REDEFINITION package took two weeks across all and was conducted across test environments. This meant running all collections, loads and publishes on the production database copy of the production instance. Performance metrics showed a 30% improvement on the test environments.

Testing also made it clear that partitioning the 800 gigabyte table would be key to a successful table rebuild. While partitioning is certainly not necessary for every table, the sheer size of this table made it a requirement. Partitioning the table by month would enable Oracle to drop off very specific segments of the data at regular intervals as needed.

Two additional concerns were also covered in testing. First, the team needed to ensure that users could still make updates to the system during online rebuild, despite the volume of background activity during the data extract. Second, the team needed to determine whether high CPU utilization during the data extract would cause the system to grind to a halt.

In fact, testing showed that DBMS_REDEFINITION could be run while users were online, and that the users could access and update Demantra worksheets while the on-line redefinition was taking place without noticeable performance degradation. This data indicated that the online table rebuild using DBMS_REDEFINITION was a viable option.

Ready Set Rebuild

Simple wrapper scripts were created around the package to enable the testing to begin. An Oracle customer would also need to follow this step because the wrapper scripts are unique to the table being rebuilt. The scripts check the system and table to ensure that both are ready for redefinition. Each of the steps describing the code used is included in this paper.

The rebuilding process actually re-defines the structure of the t_ep_spf_data table by adding partitions and copying the data. This table contains service parts forecast data and service parts shipment history and the re-definition will improve Demantra performance. In Oracle's case, the Demantra table was around 800 GB including indexes. However, it should be noted that it took eight hours to just analyze the table and check the table characteristics.

Full Speed Ahead

Conventional steps for rebuilding the table – in the standard manner – still apply with the implementation of DBMS_REDEFINITION. A separate table is created; data is extracted. The data is still put in a new table and swapped for the old. However, prior to using the DBMS_REDEFINITION package, any table rebuilding required downtime. In Oracle's case, this downtime took approximately six hours once every six months.

The objective of leveraging the DBMS_REDEFINITION package was to use the same rebuilding steps as before, but without imposing *any* downtime on the users.

It turned out that the DBMS_REDEFINITION package actually allowed the entire process to be done online, while users were on the system. This feature has always been part of the standard 10g Release 2 product, yet Oracle IT had not widely utilized the package this way previously.

To ease maintenance further, partitioning would be done while rebuilding the table, which in turn improved row chaining. The whole table re-definition process improves row chaining, but in Oracle's case, because the tables were very large, the tables were partitioned first, during the online redefinition process. The use of DBMS_REDEFINITION does not always require partitioning however.

This on-line re-definition process is not limited to just the Demantra application. This tool can be used to modify the logical or physical structure of any table to improve performance. This makes it a valuable tool in re-building tables to improve row chaining or performing row re-ordering for any number of applications.

Steps for Rebuilding Tables in Demantra using DBMS_REDEFINITION

The complete set of steps for utilizing DBMS_REDEFINITION to rebuild the table is outlined below. For more detailed background refer to the Oracle Database Administrator's Guide (10g Release 2 (10.2), http://docs.oracle.com/cd/B19306_01/server.102/b14231/tables.htm#i1006754

Figure 1. The Steps for Table Redefinition

- **Check Space Requirements**
- **Create the Interim Table**
  - **Get the table structure**
  - **Remove all the constraints**
  - **Partition the table by using 'partition by interval'**
- **Check If Tables Can Be Redefined**
- **Copy Data to the Interim Table**
  - **Copy the Records**
  - **Copy the Indexes and Constraints**

Check Space Requirements

Before beginning, the script below must be run to ensure there is enough free space for the re-definition. The re-definition procedure requires having two copies of the same table.

To get the space used for the table t_ep_spf_data, use the following script:

```
SELECT SEGMENT_NAME "INDEX", TABLESPACE_NAME,COUNT(DISTINCT(PARTITION_NAME)) PARTITIONS,
ROUND(sum(bytes)/(1024*1024*1024),4) "Size (GB)",  ROUND(sum(bytes)/(1024*1024),4) "Size (MB)",
SUM(BYTES) "Size (Bytes)"

FROM  DBA_extents

WHERE segment_name like upper('%t_ep_spf_data%')

AND (segment_type = 'TABLE' OR SEGMENT_TYPE = 'TABLE PARTITION')

AND OWNER = 'DEMANTRA'

and tablespace_name = 'TS_DP'

GROUP BY SEGMENT_NAME, TABLESPACE_NAME;
```

Similarly, to get the space used by the indexes, use this script:

```
SELECT SEGMENT_NAME "Index", TABLESPACE_NAME, ROUND(sum(bytes)/(1024*1024*1024),4) "Size (GB)"

FROM DBA_extents

WHERE segment_name in (select  index_name from    dba_indexes where   owner     = 'DEMANTRA' and table_name = 'T_EP_SPF_DATA')

AND (segment_type = 'INDEX' )

AND OWNER = 'DEMANTRA'

GROUP BY SEGMENT_NAME, TABLESPACE_NAME
```

If the table needs partitioning, as the Oracle table did, approximately 10% more free space will be required than the one reported by the above scripts.

Create the Interim Table

In this implementation, the size of the table t_ep_spf_data was around 120 GB. The collections, publishes and loads that used this table had performance issues due to the table size. As a solution, the table would be partitioned by month. By creating a new partition for each month the updates and inserts became much faster. Then, if a new record needed to created and the corresponding date was not available in any of the partitions, a new partition was created automatically

Indexes were also partitioned so in the future, the partitions of the table can be safely truncated when that data is no longer needed. Use the following steps to create the interim table.

- Get the table structure with the following script:

```
select DBMS_METADATA.GET_DDL('TABLE','T_EP_SPF_DATA') from DUAL;
```

- Remove all the constraints, including null constrains returned by the above script.
- Partition the table by using ‘partition by interval’. In this case the interval is a month.

The final script to create the interim table is similar to the script below. Most of the columns have been removed in order to keep the document readable.

```
CREATE TABLE DEMANTRA.T_EP_SPF_DATA_NEW
  (
           ITEM_ID NUMBER(10,0),
             LOCATION_ID NUMBER(10,0),
             SALES_DATE DATE,
        T_EP_SPF_ID NUMBER(10,0),

        LAST_UPDATE_DATE DATE DEFAULT SYSTIMESTAMP,

        IS_SELF NUMBER(10,0) DEFAULT 1 ,

.

.

.

SPF_CALC_BIAS_LAG_1 NUMBER(20,10),

        OBS_ERROR_STD NUMBER(20,10),

        OUTLIER NUMBER(20,10)

)


PCTFREE 40 PCTUSED 50 INITRANS 50 MAXTRANS 255 LOGGING

STORAGE(INITIAL 81920 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645 PCTINCREASE 0
FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT)  TABLESPACE TS_DP

  PARTITION BY RANGE (

    SALES_DATE

  )

INTERVAL (NUMTOYMINTERVAL(1,'MONTH'))  (

   PARTITION P_SPF_DATA_12_2008 VALUES LESS THAN (TO_DATE( ' 2009-01-04 00:00:00','SYYYY-MM-DD
HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))

  )   ;
```

Check If Tables Can Be Redefined

To redefine the interim table, it must not have any constraints or indexes. This includes null constraints.

To validate whether the tables can be redefined, use the custom script “PDIT_TABLE_REDEF.checks”.

```
-- PDIT_TABLE_REDEF.checks

PROCEDURE checks(

   tab_owner       VARCHAR2,

   tab_name        VARCHAR2,

   tab_redef_name  VARCHAR2

   )

AS

  invalid_triggers NUMBER;

  constraints      NUMBER;

  indexes_count    NUMBER;
```

```
 col_count       NUMBER;
BEGIN

 ---------------------------------------
 -- Check if table can be redefined
 ---------------------------------------
 dbms_application_info.set_action( '1 Redef' );
 dbms_redefinition.can_redef_table( tab_owner,
                                    tab_name,
                                    dbms_redefinition.cons_use_pk);
 ---------------------------------------
 -- Check for invalid triggers otherwise copying dependent objects will fail
 ---------------------------------------
 select  count(*)
   into  invalid_triggers
   from  dba_objects obj
        ,dba_triggers trig
  where  trig.table_owner  = tab_owner
    and  trig.table_name   = tab_name
    and  trig.trigger_name = obj.object_name
    and  trig.owner        = obj.owner
    and  obj.status        = 'INVALID';
 if invalid_triggers > 0
 then
   DBMS_OUTPUT.PUT_LINE( 'ERROR : Table as invalid triggers. Can''t continue' );
   return;
 end if;
 ---------------------------------------
 -- Check to see of interim table has constraints
 -- copying dependent objects will fail
 ---------------------------------------
 select  count(*)
   into  constraints
   from  dba_constraints
  where  owner      = tab_owner
    and  table_name = tab_redef_name;
 if constraints > 0
 then
```

```
   DBMS_OUTPUT.PUT_LINE( 'ERROR : Interim table has constraints. Can''t continue' );
 end if;
 ---------------------------------------
 -- Check to see of interim table has indexes
 -- copying dependent objects will fail
 ---------------------------------------
 select  count(*)
   into  indexes_count
   from  dba_indexes
  where  table_owner = tab_owner
    and  table_name  = tab_redef_name
    and  index_type != 'LOB';

 if indexes_count > 0
 then
   DBMS_OUTPUT.PUT_LINE( 'ERROR : Interim table has indexes. Can''t continue' );
 end if;

----------------------------------------
 -- Check both tables have the same columns
 ---------------------------------------
 select  count(*)
   into  col_count
   from  ( select  column_name
                ,count(*)
           from  dba_tab_columns
          where  table_name IN ( tab_name, tab_redef_name )
            and  owner     = tab_owner
         having  count(*) != 2
          group  by column_name
        );
 if col_count > 0
 then
   DBMS_OUTPUT.PUT_LINE( 'ERROR : The interim table has different columns to the main table. Can''t
continue' );
 end if;
END checks;
```

Finally, execute the “PDIT_TABLE_REDEF.checks” script with the right parameters. Oracle used the name t_ep_spf_data_new for the interim table.

```
exec PDIT_TABLE_REDEF.checks('DEMANTRA','T_EP_SPF_DATA','T_EP_SPF_DATA_NEW');
```

Copy Data to the Interim Table

Copying the data consists of two steps: the copy of the records and creation of the indexes and constraints.

Copy the Records

At this point, the data from the original table is copied to the interim table. Until the entire redefinition process is completed, any change in the original table will be automatically captured in the interim table. The procedure to do the copy is “PDIT_TABLE_REDEF.DO_COPY” as shown below.

```
-- PDIT_TABLE_REDEF.DO_COPY
PROCEDURE do_copy (
  tab_owner      VARCHAR2,
  tab_name       VARCHAR2,
  tab_redef_name  VARCHAR2,
  degree         NUMBER   DEFAULT 8,
  order_by_cols   VARCHAR2 DEFAULT NULL
  )
AS
 nbr_errors      NUMBER;
 invalid_triggers NUMBER;
 ddl           CLOB;
BEGIN

 ---------------------------------------
 -- Force Parallel
 ---------------------------------------
 if degree > 1
 then
   execute immediate 'ALTER SESSION ENABLE PARALLEL DML';
   execute immediate 'ALTER SESSION FORCE PARALLEL DML PARALLEL '   || degree;
   execute immediate 'ALTER SESSION FORCE PARALLEL DDL PARALLEL '   || degree;
```

```
      execute immediate 'ALTER SESSION FORCE PARALLEL QUERY PARALLEL ' || degree;
    end if;
    -- Resumable transaction 6hr timeout
    execute immediate 'ALTER SESSION ENABLE RESUMABLE TIMEOUT 21600 NAME ''Reorg of ' || tab_name
  || '''';


    ---------------------------------------
    -- Start Redefinition
    ---------------------------------------
    dbms_application_info.set_action( 'Copying rows' );
    dbms_redefinition.start_redef_table( uname       => tab_owner,
                                         orig_table   => tab_name,
                                         int_table    => tab_redef_name,
                                         options_flag => dbms_redefinition.cons_use_pk,
                                         orderby_cols => order_by_cols
                                       );
  END do_copy;
```

Finally, to copy the data, in this case using 16 workers, execute the following:

```
exec PDIT_TABLE_REDEF.DO_COPY('DEMANTRA','T_EP_SPF_DATA','T_EP_SPF_DATA_NEW', degree => 16);
```

Copy the Indexes and Constraints

Instead of using the standard function “dbms_redefinition.COPY_TABLE_DEPENDENTS” to create the indexes, a custom code was used to create them. This is because the standard code copies the original structure of the indexes. Oracle also made two changes to the indexes. First, the indexes were partitioned, then storage parameters were added.

Indexes need to be registered after they are created in order to distinguish between new and old to the standard code upon switchover. For this task use the procedure “DBMS_REDEFINITION.REGISTER_DEPENDENT_OBJECT”.

The Oracle database creates a new column for function results only after the index is created. An example of this index is “T_EP_SPF_DATA_QTY_FORM_FI”.

To collect statistics in the interim table, execute the “do_index” procedure using the following code:

```
-- PDIT_TABLE_REDEF.DO_INDEX
PROCEDURE do_index (
  tab_owner      VARCHAR2,
  tab_name       VARCHAR2,
  tab_redef_name  VARCHAR2,
  degree         NUMBER   DEFAULT 8,
  order_by_cols   VARCHAR2 DEFAULT NULL
  )
AS
 nbr_errors      NUMBER;
 invalid_triggers NUMBER;
 ddl            CLOB;
BEGIN

 --------------------------------------
 -- Force Parallel
 --------------------------------------
 if degree > 1
 then
   execute immediate 'ALTER SESSION ENABLE PARALLEL DML';
   execute immediate 'ALTER SESSION FORCE PARALLEL DML PARALLEL '   || degree;
   execute immediate 'ALTER SESSION FORCE PARALLEL DDL PARALLEL '   || degree;
   execute immediate 'ALTER SESSION FORCE PARALLEL QUERY PARALLEL ' || degree;
 end if;

 -- Resumable transaction 6hr timeout
 execute immediate 'ALTER SESSION ENABLE RESUMABLE TIMEOUT 21600 NAME ''Reorg of ' || tab_name
|| '''';
 --------------------------------------
 -- Copy Primary key
 --------------------------------------
for pk_index in (
select c.constraint_name, (select  rtrim(xmlagg(xmlelement(e,column_name,',').extract('//text()') ORDER BY
column_position),',')
FROM dba_ind_columns
Where table_name=tab_name

and index_name = c.constraint_name

group by index_name)  as  pk_columns from dba_constraints c where table_name = tab_name

and constraint_type in ('P')

and  not exists   (select constraint_name from  dba_constraints where table_name= tab_redef_name and
constraint_type in ('P'))

)

LOOP

 execute immediate 'ALTER TABLE '|| tab_owner ||'.' || tab_redef_name || ' ADD ( CONSTRAINT ' ||
substr(pk_index.constraint_name ,1,26)|| '_R PRIMARY KEY ( ' || pk_index.pk_columns || ' ) USING INDEX
PCTFREE 40 INITRANS 50 MAXTRANS 255 NOLOGGING  STORAGE( BUFFER_POOL DEFAULT FLASH_CACHE
DEFAULT CELL_FLASH_CACHE DEFAULT) TABLESPACE TS_DP  LOCAL  ENABLE )' ;

  --Register the primary key as constrain and  index

DBMS_REDEFINITION.REGISTER_DEPENDENT_OBJECT(tab_owner,tab_name,tab_redef_name,dbms_redefi
nition.cons_constraint,tab_owner,pk_index.constraint_name , substr(pk_index.constraint_name ,1,26)
||'_R');

DBMS_REDEFINITION.REGISTER_DEPENDENT_OBJECT(tab_owner,tab_name,tab_redef_name,dbms_redefi
nition.cons_index,tab_owner,pk_index.constraint_name ,substr(pk_index.constraint_name ,1,26) ||'_R');

END LOOP;


--Creating the rest of the indexes

FOR c_indexes IN (  select index_name, rtrim(xmlagg(xmlelement(e,column_name,',').extract('//text()')
ORDER BY column_position),',') as pk_columns

FROM dba_ind_columns

Where table_name=tab_name

and index_name <> (select constraint_name from dba_constraints where table_name = tab_name and
constraint_type in ('P') )

and index_name <> 'T_EP_SPF_DATA_QTY_FORM_FI'

and not exists (select i.index_name from  dba_indexes i where table_name=tab_redef_name  and
i.index_name = substr(index_name,1,26) || '_R')

group by index_name


)

LOOP

execute immediate 'CREATE INDEX '|| tab_owner ||'.'||substr(c_indexes.index_name,1,26) || '_R ON '||
tab_owner ||'.'||tab_redef_name || ' ( '|| c_indexes.pk_columns || ' ) PCTFREE 40 INITRANS 50
MAXTRANS 255 NOLOGGING STORAGE (BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT
CELL_FLASH_CACHE DEFAULT) TABLESPACE TS_DP LOCAL PARALLEL';

--Registering the indexes
DBMS_REDEFINITION.REGISTER_DEPENDENT_OBJECT(tab_owner,tab_name,tab_redef_name,dbms_redefi
nition.cons_index,tab_owner,c_indexes.index_name , substr(c_indexes.index_name,1,26)||'_R');

END LOOP;
```

```
--Create index with functions
   execute immediate 'CREATE INDEX DEMANTRA.T_EP_SPF_DATA_QTY_FORM_FI_R ON
DEMANTRA.T_EP_SPF_DATA_NEW
   (
     T_EP_SPF_LATEST_REV_ID,
     LOCATION_ID,
     SALES_DATE,
     NVL(SPF_SHIPMENT_OVER,ACTUAL_QUANTITY_DEP)
   )
PCTFREE 10 INITRANS 40 MAXTRANS 255
STORAGE  (
   BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT
  )
  TABLESPACE TS_DP LOCAL NOLOGGING  PARALLEL';


--Register the index
DBMS_REDEFINITION.REGISTER_DEPENDENT_OBJECT(tab_owner,tab_name,tab_redef_name,dbms_redefi
nition.cons_index,tab_owner,'T_EP_SPF_DATA_QTY_FORM_FI','T_EP_SPF_DATA_QTY_FORM_FI_R');


--Coping the constraints
  dbms_application_info.set_action( 'Copy Dependents' );
  dbms_redefinition.COPY_TABLE_DEPENDENTS(
    uname           => tab_owner,
    orig_table       => tab_name,
    int_table        => tab_redef_name,
    copy_indexes     => 0, --NOTICE THAT WE ARE NOT COPING THE INDEXES
    copy_triggers    => TRUE,
    copy_constraints => TRUE,
    copy_privileges  => TRUE,
    ignore_errors    => FALSE,
    num_errors       => nbr_errors,
    copy_statistics  => FALSE,
    copy_mvlog       => FALSE);
  IF nbr_errors > 0
  THEN
   DBMS_OUTPUT.put_line ( 'Copy dependants errors : ' || nbr_errors );
   return;
  END IF;
  ----------------------------------------------
  -- Reset index parallelisms after they are copied
  ---------------------------------------------
  dbms_application_info.set_action( 'Resetting Index parallelism' );
  FOR c_indexes IN (  select  owner, index_name
                 from    dba_indexes
                 where   owner      = tab_owner
                 and
table_name = tab_redef_name
                 and     index_type != 'LOB' )
  LOOP
    execute immediate 'alter index ' || c_indexes.owner || '.' || c_indexes.index_name || ' noparallel
logging' ;
  END LOOP;

--------------------------------------
  -- synchronize new table with interim data before stats gather
  --------------------------------------
  dbms_application_info.set_action( 'Sync interim table 1' );
  dbms_redefinition.sync_interim_table( tab_owner, tab_name, tab_redef_name );

  --------------------------------------
  -- Gather stats on the interim table before you exchange it
  --------------------------------------
    dbms_application_info.set_action( 'Interim table gather stats' );
    dbms_stats.gather_table_stats( ownname         => tab_owner,
                          tabname         => tab_redef_name,
                          estimate_percent => g_estimate_percent,
                          degree          => degree,
                          cascade         => TRUE,
                          method_opt      => g_method_opt
                         );
  END do_index;
```

This action completes the data synchronization process and the system is ready for use.

Results

Through the rebuild, Oracle achieved improved system performance of greater than 30%. The key result however is that the company was able to accomplish this re-build without imposing downtime on users. This gave Oracle the freedom to plan for and implement table rebuilds without the constraint of planning for downtime which could run into hours or even days.

The traditional method of table rebuilding would probably have required up to five continuous days of downtime rather than hours for this large and fragmented table. However, using the on-line redefinition tool we were able to reduce the downtime to near zero. The users did not even realize that the rebuild was being done in the background while they continued to use the Demantra system.

Table 1. Traditional Rebuild vs. DBMS_Redefinition

	TRADITIONAL REBUILD	DBMS_REDEFINITION
CALENDAR TIME LAPSED	~ not known -- possibly 5 estimated days for the t_ep_spf_data table. ~ 6+ hours for smaller tables	21 hours (16 hours to do the re-def and 5 hours for the data synchronization)
USER DOWNTIME	~ not known -- possibly 5 estimated days for the t_ep_spf_data table. ~ 6+ hours for smaller tables	0 hours *

*The last step of the online redefinition is the data sync process. This process is faster if users are not performing large updates in the system because it is syncing up later updates before switching the tables. Of course it is much faster if the users do not make ‘any’ updates.

Partitioning resulted in faster queries. Maintenance was easier because Oracle IT implemented a process to purge old data by monthly partitions to control data growth. Time consuming rebuilding of indexes was not necessary because the tables were partitioned.

When row chaining now exceeds 15%, the team can respond significantly faster because the table rebuild can be performed by partition instead of on the whole table. In short, the maintenance of tables became much easier.

After the re-build, queries on the spf_data table ran 30% faster, resulting in faster collections, publishing and loading of the data.

Conclusion

The DBMS_REDEFINITION package enabled Oracle IT to rebuild the company's 800GB Spares Planning database table without any user downtime. While is hardly surprising that rebuilding this table improved system performance by 30%, the ability to do the rebuild online with zero user downtime proved extremely useful given Oracle's Oracle's 24/7 Demantra user base.

As data is added and the table grows beyond a terabyte, using the DBMS_REDEFINITION package will allow Oracle IT manage future performance issue with relative ease. Oracle recommends that customers with large Demantra data tables monitor critical tables for row chaining and rebuild their tables regularly using the method described in this document.

In addition, since the DBMS_REDEFINITION package is a native feature of the Oracle database, Oracle customers may also find it useful for rebuilding tables in other Oracle applications.

Implementing Large-Scale Demantra Table Rebuilds To Improve Performance with Zero Downtime
January 2013
Author: Paul Dominic, Senior Director
Contributing Authors:
Miguel Angel Palomera Perez, IT Sr. Consultant
Andy Campbell, IT Senior Systems Consultant
Andrea Kim, Senior Manager

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com