ORACLE FINANCIAL SERVICES

ORACLE DATABASE 12c

Oracle Database In-Memory with Oracle's JD Edwards EnterpriseOne

A database administrator's guide to optimizing Oracle Database In-Memory with JD Edwards EnterpriseOne Applications

ORACLE TECHNICAL BRIEF | JANUARY 2015 | VERSION 1

ORACLE

Disclaimer

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Table of Contents

.

Introduction: Why Do I Need an In-Memory Database?

Oracle's JD Edwards EnterpriseOne is an enterprise resource planning (ERP) software application that combines business value, standards-based technology, and deep industry experience into a business solution. Enterprises of all sizes, industries, and geographies use JD Edwards EnterpriseOne to capture, manage, and analyze the data that makes their businesses compete and succeed.

As is typical of ERP systems, JD Edwards EnterpriseOne provides a set of software modules that allows end users to capture and report on financial transactions, inventory, manufacturing operations, projects, customers, employees, and many other activities of a modern enterprise. As part of their daily activity end users need to query data from the system to support traditional ERP transactions, such as invoicing a customer or shipping a product. However, in an increasingly competitive business environment, executives, managers, and empowered employees also need data from the system to support their decision-making, such as analyzing sales trends or responding to unplanned events. As businesses grow and as time passes, the data collected within the system can grow at an astounding rate.

To keep the enterprise operating efficiently, administrators are charged with the responsibility of tuning the hardware, ERP software, and database to run at optimal performance. Unfortunately, administrators are pressured from both sides: the system must be continually tuned as human and nonhuman users exercise the database with transactional data; meanwhile, users, managers, and executives conceive of new queries, dimensions, and usage patterns, which the system may not be tuned to satisfy. How are system and database administrators to deal with incessant data expansion and at the same time fulfill new requests for that data? A variety of approaches are available, but each comes with its compromises and trade-offs:

- **Batch jobs.** If an end user's query for data returns too slowly to be practical within an interactive context, the traditional solution is to write a batch report that runs off-line. While this solution does not actually increase the speed at which the system returns the data, at least the user is free to go on with other tasks while the system batch job works in the background to return the data. If the use case is repetitive and reusable, for example a financial report that runs at the end of every month, then investment in developing the batch report may be justified. But if the data request is an infrequent case, the cost does not justify the "one-off" request.

- **Database indexes.** Database administrators can create indexes over the database tables, which can accelerate the database's ability to return data to the application. However, an

index is often applicable to a specific or narrow set of usage patterns. Creating an index requires the engagement of a capable database administrator, and the index itself consumes system resources, such as CPU, memory, and disk space. As users request more and more multidimensional data, and as the set of indexes grows, the system can actually decrease its efficiency to complete database requests to a point of diminishing return.

- **Just say "no."** Sometimes an end user has an unanticipated "ad hoc" need for data. Some of these "ad hoc" needs for data stem from unanticipated business situations that require a company to make educated, informed decisions very quickly in order to reduce risk, remain competitive, retain customer satisfaction, just to name a few situations. Being unpredictable, it is not possible to predetermine the need for an index over that data or a report to fetch it asynchronously in a batch process. If that data resides in a very large table, the database request might not just create an additional load on the system, but also may not return the result to the interactive user before an application time-out occurs. And the "fix"? Typically it is to use an application or database security task to prevent users from running such queries. Yes, that approach protects the system and averts application failures, but it does little to satisfy the user's original request for data.

- **Use another system.** When the data is needed to support analytical use cases, sometimes the ERP data is exported to an external data warehouse. In this process, the data can also be transformed from its transaction-optimized row-based format to a query-optimized columnar format. While data warehouses and business intelligence applications can unlock a wealth of value within the data, they do require additional resources, and the data within them can only be as current as the rate at which it is refreshed from the ERP system.

Oracle Database, and specifically the Database In-Memory option, provides an innovative new alternative that addresses these problems without the compromises. With the introduction of Oracle Database In-Memory, a single database can now efficiently support mixed workloads, delivering optimal performance for transactions while simultaneously supporting real-time analytics and reporting. This is possible due to a unique "dual-format" architecture that enables data to be maintained in both the existing Oracle row format, for online transaction processing (OLTP) operations, and a new purely in-memory column format, optimized for analytical processing. While customers may experience some performance benefit to their existing JD Edwards EnterpriseOne applications, the dramatic benefit of Database In-Memory will be realized in unlocking new analytical usage patterns over the same real-time OLTP data without modifying or impacting existing applications.

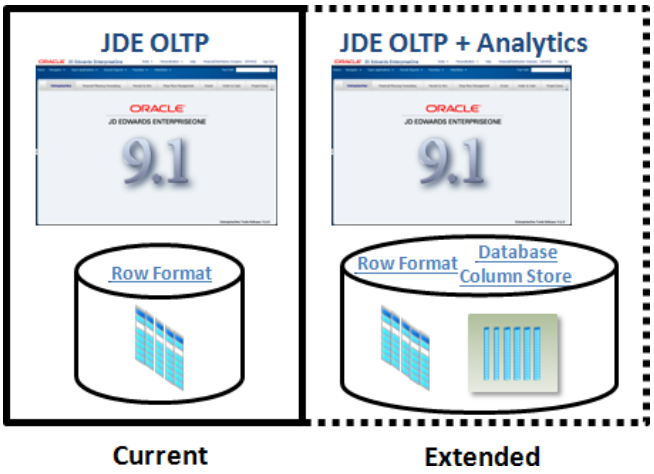

Figure 1: JD Edwards EnterpriseOne with Oracle Database In-Memory.

Figure 1 illustrates the dual-format architecture enabling OLTP and analytics to be performed in EnterpriseOne from the same database. The addition of the database column store can extend the analytical capabilities to the current JD Edwards EnterpriseOne architecture.

The remainder of this technical brief is intended for IT managers and database administrators who are considering or planning to implement JD Edwards EnterpriseOne database with Oracle Database In-Memory. The reader is assumed to have a general understanding of Oracle Database architecture and administration and how JD Edwards EnterpriseOne stores its transactional data in database tables.

This technical brief will guide you through answering the following three questions:

1. *If Database In-Memory is going to store the JD Edwards EnterpriseOne tables in memory, how much more physical memory (RAM) will my database server need?*
2. *Which JD Edwards EnterpriseOne tables should I consider putting into memory, and which ones should I leave in traditional row format? How do I decide?*
3. *If Database In-Memory is scanning JD Edwards EnterpriseOne tables in memory, can I drop my database indexes? How do I decide?*

Overview of Oracle Database In-Memory

Oracle Database has traditionally stored data in a row format. In a row format database, each new transaction or record stored in the database is represented as a new row in a table. That row is made up of multiple columns, with each column representing a different attribute about that record. A row format is ideal for OLTP systems like JD Edwards EnterpriseOne, as it allows quick access to all of the columns in a record since all of the data for a given record are kept together in-memory and on-storage.

A column format database stores each of the attributes about a transaction or record in a separate column structure. A column format is ideal for analytics, as it allows for faster data retrieval when only a few columns are selected but the query accesses a large portion of the data set. But what happens when a DML operation (insert, update or delete) occurs on each format? A row format is incredibly efficient for processing DML as it manipulates an entire record in one operation i.e. insert a row, update a row or delete a row. A column format is not so efficient at processing row-wise DML: in order to insert or delete a single record in a column format all of the columnar structures in the table must be changed.

Up until now you have been forced to pick just one format and suffer the tradeoff of either suboptimal OLTP or sub-optimal analytics performance. Oracle Database In-Memory provides the best of both worlds by allowing data to be simultaneously populated in both an in-memory row format and a new in-memory column format.

As an OLTP system, JD Edwards EnterpriseOne was developed with data stored in row format by design. Over time, as the logic of the applications and usage patterns of end users become more understood, database indexes were developed to optimize the efficiency of the database requests that are issued to the database server. Database indexes provide a fast way for the database server to find and retrieve data in row format when queries to that row involve a column(s) on which the index is built. Indexes are most effective and practical when the logic of an application or the usage patterns of end users are foreknown.

If the data being requested by an application or end user does not have a prebuilt index to guide its search and retrieval, the database server is relegated to use a process called a full table scan to find the data. As its name implies, the database server must scan data row-by-row searching for the data contained in some field within the row. For example, if a user submits a query to the database to find all invoices greater than one million dollars issued to customers in California, the database server might have to scan millions of records of invoices of all dollar amounts and customers in all geographies. Therefore, full table scans can surely be resource-intensive and a perceived performance bottleneck to end users.

While indexes generally result in more optimized search and retrieval of data compared to full table scans, there are situations for which a full table scan is necessary, more efficient, or perhaps is the only option. Full table scans are not necessarily to be avoided at all costs. For example, a full table scan might be more efficient if the query will return a large number of rows in proportion to the total number of rows in the table. The Oracle Database includes a feature called the “optimizer,” which automatically decides the best way for the server to retrieve the data based on factors such as the characteristics of the query, statistics on the database, and whether indexes exist.

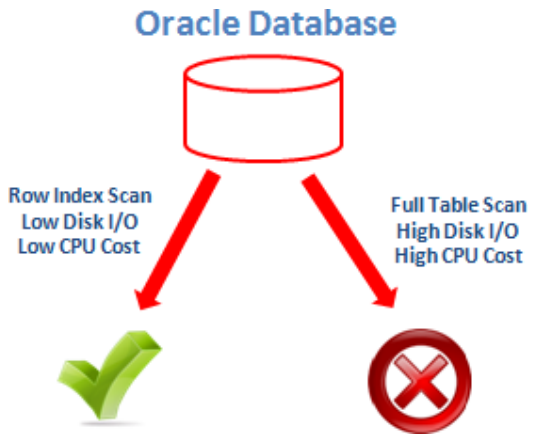

Figure 2: Row Index versus Full Table Scans

Figure 2 illustrates the need to evaluate the general benefit of an indexed row scan and needing to evaluate the occurrences of full table scans. A full table scan that results in a large consumption of I/O and has a high cost to the CPU should be avoided unless required. Database requests that cause high impact full table scans should be identified in the EnterpriseOne application and evaluated for performance and need to the customer.

Row format is a database design of an object where a single row in an Oracle table has multiple columns and each column represents a characteristic of that row record. This design is analogous to what is commonly seen in a typical spreadsheet, where we have rows of records and each record in the spreadsheet has column attributes. The design of row format is well suited for OLTP applications such as JD Edwards EnterpriseOne. Row format is also efficient in processing information concerning a row when a DML (insert, update, or delete) operation is initiated.

Column format stores the data attributes in separate column structures. Column format is an efficient way to process analytic type of information (sum, difference, product, etc). Column formats are also superior in performance when only a few queried data is returned over a large amount of data. Columnar format is not so efficient in processing DML operations. In a DML operation in a columnar format, a single update will have to modify all of the columnar structures for the change to be made. Columnar format is thus more efficient on query (select) type operations or when analytic type functionality is requested. A full table scan of data in columnar format is dramatically faster and more efficient than a full table scan of data in row format. This is particularly true when the column format data already resides in memory and is current with the row-format data being updated by OLTP applications.

The benefit of the Database In-Memory column store is that now, both row and column formats are present to the JD Edwards EnterpriseOne application for the Oracle Database optimizer to choose which is faster at getting at the data. This allows a balanced solution for both OLTP and analytic usage patterns. The question that the Oracle Database optimizer has to determine is whether a row index scan performance is a better plan than a columnar full scan. In general for most OLTP transactions when an index is present, the optimizer will choose a row index scan over a columnar format full table scan. Conversely, if the SQL query contains analytic type functions, a full table scan over columnar format data would provide a superior explain plan.

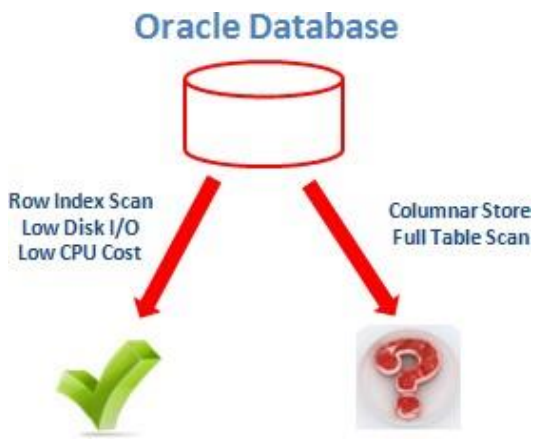

Figure 3: Row Index Scan or Columnar Scan

Figure 3 illustrates the choice the Oracle optimizer has to make between a row index scan, which is known to perform well in an OLTP EnterpriseOne environment and the questionable performance benefit of the same table in the column store. The column store will benefit those EnterpriseOne analytical queries, but how does the column store work with the EnterpriseOne application? This is the question that this document will attempt to answer.

Special Highlight 1: Columnar format is INEFFICIENT with DML operations, which are typical of OLTP transactions, but can provide EFFICIENT performance with full table scans and non-indexed column queries, which are typical of analytic usage patterns.

To provide a system that is optimally tuned for users who need both OLTP and analytic usage patterns, the database administrator must understand how the Oracle Database optimizer chooses whether to use a database index over row-format data or scan over data in the in-memory column store.

Determining the Memory Requirements for JD Edwards EnterpriseOne Data

Question #1: If Database In-Memory is going to store the JD Edwards EnterpriseOne tables in memory, how much more physical memory (RAM) will my database server need?

Database In-Memory maintains the column store (the column-format representation of the data) within the System Global Area (SGA) in a new structure called the In-Memory Area. Via database initialization parameters the database administrator can control the size of SGA, the size of the In-Memory Area within SGA, and which database tables to load (or remove) from the In-Memory Area.

Database In-Memory provides automatic compression of the data as it populates the in-memory column store. Therefore, the amount of physical memory and the allocation of SGA and the In-Memory Area is only a fraction of the size that the database table occupies on disk. In this context, the term “compression” simply refers to the ratio of the size of the table on disk divided by the size of the table in memory.

For tables that have already been put into the in-memory column store, the following SQL statements can be used to calculate the compression ratio achieved:

```
-- Script to determine the Oracle 12c In-Memory Compression Ratio
set pagesize 999;
column segment_name format a30 trunc
set linesize 125
SELECT segment_name, round(inmemory_size/1024/1024/1024,4) as im_gb,
  round(bytes/1024/1024/1024,4) as disk_gb, round(BYTES_NOT_POPULATED/1024/1024/1024,4) as gb_not_populated,
  round(bytes/inmemory_size, 4) as compression_ratio
FROM gv$im_segments
order by owner, segment_name, populate_status;
```

SQL Code 1: Displaying the Column Store Compression Ratios

Oracle also provides a utility called the Oracle Compression Advisor (DBMS_COMPRESSION), which has been enhanced to support in-memory compression. Refer to the *Oracle Database In-Memory* technical brief and Oracle Database product documentation for more information about this utility.

Compression Results from Lab Testing

The table below presents data that was collected during lab testing of JD Edwards EnterpriseOne with Database In-Memory. For a set of common data tables, the size of the table on disk and the size of the table in the in-memory column store are listed. The compression ratio is simply the size on disk divided by the size in memory. Naturally the compression ratio you realize is highly dependent on your data. Data that is highly repetitive or has many null values will have a higher compression ratio.

Segment Name	In-Memory Size (GB)	Disk Size (GB)	COMPRESSION RATIO
F0006	0.0021	0.0195	9.4118
F0010	0.0011	0.0002	0.1667
F0011	0.0011	0.0049	4.4444
F0015	0.0011	0.0001	0.1111
F00165	0.0930	0.3047	3.2270
F0101	0.0079	0.1016	12.8000
F0111	0.0070	0.1172	16.8421
F03012	0.0011	0.0703	64.0000
F03B11	0.0179	1.0625	59.4130
F03B13	0.0011	0.0020	1.7778
F03B14	0.0011	0.0146	13.3333
F0411	0.0031	0.0547	17.9200
F060116	0.0040	0.0342	8.4848
F0901	0.0346	1.3750	29.7319
F0902	0.0932	1.6250	17.4355
F0911	0.2604	14.3125	54.9557
F30006	0.0011	0.0007	0.6111
F30008	0.0011	0.0002	0.2222
F3002	0.0021	0.0439	21.1765
F3003	0.0011	0.0352	32.0000
F3111	0.2636	11.5625	43.8620
F3112	0.0159	0.8672	54.4368
F34004	0.0011	0.0001	0.1111
F3403	0.0011	0.0001	0.1111
F3411	0.0011	0.0068	6.2222
F3412	0.0011	0.0005	0.4444
F3413	0.0011	0.0029	3.5556
F4072	0.0011	0.0078	7.1111
F4095	0.0011	0.0039	2.6667
F4096	0.0011	0.0001	0.1111
F4100	0.0011	0.0039	3.5556
F41001	0.0011	0.0002	0.1667
F41002	0.0011	0.0880	8.0000
F4101	0.1680	2.3750	14.6782
F4102	0.1078	1.5625	14.4878
F41021	0.0217	0.4609	21.2732
F4105	0.0040	0.0439	10.9091
F4106	0.0090	0.2656	29.6054
F4108	0.0989	1.0000	10.1073
F4111	0.0118	0.1797	15.1753
F4115	0.0110	0.0680	6.2222
F4201	0.4017	29.3125	72.9651
F42019	0.3565	27.9375	78.3647
F4211	3.2130	252.3125	78.5283
F42119	1.9493	200.1875	102.6982
F4215	0.0060	0.2813	46.5455
F4301	0.0140	0.7500	53.6594
F43090	0.0089	0.1641	18.4410
F4311	0.1263	12.9375	102.4495
F43121	0.0031	0.1953	64.0000
F43199	0.0070	0.3359	47.8609
F4801	0.2290	1.6875	73.5319
F4801T	0.2290	1.6875	73.5319
F4802	0.0011	0.0049	4.4444
F49211	0.0112	1.4375	128.6995
F4941	0.0050	0.1875	37.4634
F4942	0.0560	1.1875	21.1939
F4945	0.0520	0.7109	13.6714
F4950	0.0011	0.0009	0.8333
F4972	0.0011	0.0059	5.3330
F4981	0.0011	0.0007	0.6667
	7.9314 (sum)	**569.0035 (sum)**	**26.9797 (average)**

Table 1: In-Memory Compression of JD Edwards EnterpriseOne Tables

Note: The compression ratios are not exact but are estimations based on the results of the SQL Code 1 rounding function.

Compression ratios vary depending on number of records, cardinality, and distribution of data. As you can see in Table 1, the compression ratio varies widely from table to table. The compression ratios listed in Table 1 are notably an estimate and not an exact calculated result. As seen in *SQL Code 1*, there is rounding of the reported results for the in-memory size, disk size, and the compression ratio columns. Thus, the above numbers are not to be taken as precise values.

Table 1 further illustrates that for the JD Edwards EnterpriseOne test database, approximately 570GB of data and indexes on disk occupied as little as 8GB memory in the In-Memory Area. The tables that are defined to be in column store is configurable and draws from the total memory that is allocated to the SGA (System Global Area) of the Oracle Database instance at startup time or at the first use of that table. Given that the column store takes a fraction of the memory (8GB) for a large data set (570GB), it is a small cost of memory in the SGA to have both data in row and column formats available to any JD Edwards EnterpriseOne application for a performance opportunity.

Special Highlight 2: Columnar format compression allows a dual format (row and column format) to be present with a small SGA memory footprint cost.

Because this data was collected from an Oracle lab test system, it is believed that the average compression ratio of 27x shown in Table 1 is on the high side. Data that is very repetitive and contains many null values will generally compress at a higher rate. With real-world data, customers should expect the ratio to be in the range of 10-20x.

An example set of SQL is provided in *SQL code 2 to configure* the necessary amount of SGA for use in the column store.

```
alter system set sga_target=45G scope=spfile;
alter system set memory_target=55G scope=spfile;
alter system set memory_max_target=55G scope=spfile;
alter system set sga_max_size=45G scope=spfile;
alter system set pga_aggregate_limit=10G scope=spfile;
alter system set pga_aggregate_target=8G scope=spfile;
alter system set inmemory_size=8G scope=spfile;
```

SQL Code 2: Memory Parameters for Oracle Column Store

In the above example 55GB is available for the Oracle Database; 8 GB of that 55GB memory is used for the In-Memory Area. The other parameters are the memory, SGA (System Global Area), and PGA (Program Global Area) memory settings used in testing. Because memory settings in the Oracle Database are static, a recycling of the Oracle Database services is required for these values to become permanent. This is just an example of values; the Oracle Database administrator must determine what is proper for their environment.

See the Oracle Database Administration Guide for more information about configuring the In-Memory Column Store: (https://docs.oracle.com/database/121/NEWFT/chapter12102.htm#NEWFT003).

Identifying Which JD Edwards EnterpriseOne Tables to Load into the Column Store

Question #2: Which JD Edwards EnterpriseOne tables should I consider putting into memory, and which ones should I leave in traditional row format? How do I decide?

The next section of this document will describe a methodology for identifying the EnterpriseOne tables that will be placed in the column store for standard OLTP EnterpriseOne processes. As a reference, a brief explanation to the traditional approach, an explanation of EnterpriseOne indexes, and then the Oracle column store methodology will be discussed.

Database indexes are traditionally used for two purposes:

1. To ensure integrity of data within a table or set of tables. These indexes are referred to as 'primary key indexes'.
2. To accelerate application performance by giving applications a 'shortcut' for how to find the right data.

Traditional Approach

Figure 4 below shows a high level overview of the traditional approach of the Oracle column store implementation as discussed in the literature (same reference as above).

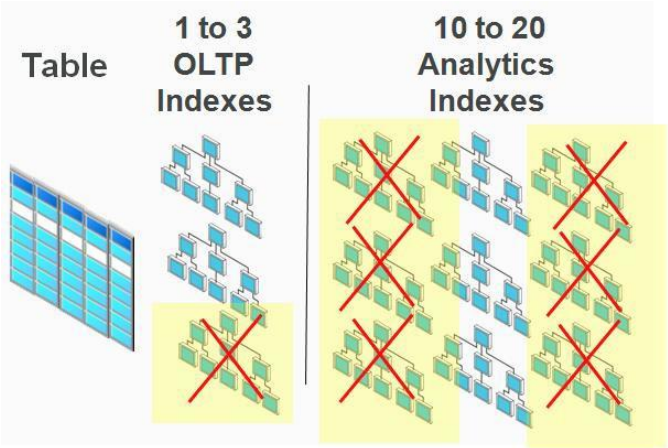

Figure 4: EnterpriseOne Indexes

The traditional approach in Figure 4 involves the following steps:

1. Identify an application table and place it in the column store.
2. The table will have associated with it OLTP indexes and analytical indexes.
3. Analyze the performance of the application with the table in columnar format.
4. Based on the analysis remove any unused indexes in OLTP and analytical indexes.

So how do we apply the above traditional approach to EnterpriseOne? To answer this question, we will first explain the EnterpriseOne index architecture.

EnterpriseOne Indexes Architecture

What tables and indexes in EnterpriseOne does the Oracle database column store represent in the traditional model?

It is now important for the user to understand the different types of indexes that are used by the EnterpriseOne application and how to identify them. EnterpriseOne breaks down indexes into two categories; data integrity or constraint indexes and non-unique indexes. A non-unique index can be used by the interactive application or batch process for performance or it can be used as an analytical index. The following explanation on the definitions of the indexes should bring more clarity to the discussion.

Data Integrity Indexes - Data integrity indexes are defined as an index built over the table that maintains the unique constraints of the data in EnterpriseOne for referential integrity. Data integrity indexes are well defined and can be easily queried by the database. The SQL presented in *SQL Code 3* will list all data integrity indexes for tables in the CRPDTA Oracle schema for the EnterpriseOne PY environment. The CRPDTA schema is where EnterpriseOne stored the data for the PY environment path code. See EnterpriseOne Guides for more details on database schema and data layout (http://www.oracle.com/technetwork/documentation/jdedent-098169.html).

```
-- SQL to gather EnterpriseOne UNIQUE Index Information
set echo on
set pagesize 10000
set linesize 200
set COLSEP '|'
SET RECSEPCHAR '~'
SET RECSEP off
column index_name format a10 trunc;
column column_name format a20 trunc;
column COLUMN_NAME format a30 trunc;
spool E1_unique_indexes_tmp.out
SELECT c.index_name, listagg(column_name, ',') within group (order by column_name) as column_name
  FROM dba_indexes i, dba_ind_columns c
WHERE c.table_owner = UPPER('CRPDTA')
  AND i.owner = UPPER('CRPDTA')
  AND i.uniqueness = UPPER('UNIQUE')
  AND i.index_name = c.index_name
  GROUP BY c.index_name;
```

SQL Code 3: EnterpriseOne Data Integrity Indexes

A short output of part of the results of initiating *SQL Code 3* is shown in Figure 5. For the majority of the EnterpriseOne tables, the unique constraint index for data integrity is designated with a '_0'. In the output below, a data integrity index for the Account Ledger table F0911 (F0911_0) is enforced by uniqueness in data for each row in the table for columns GLDCT, GLDGJ, GLDOC, GLEXTL, and GLJEL.

```
SQL> SELECT  c.index_name, listagg(column_name, ',') within group (order by column_name) as column_name
  2    FROM dba_indexes i, dba_ind_columns c
  3  WHERE c.table_owner = UPPER('CRPDTA')
  4    AND  i.owner = UPPER('CRPDTA')
  5    AND  i.uniqueness = UPPER('UNIQUE')
  6    AND  i.index_name = c.index_name
  7    GROUP BY c.index_name;

INDEX_NAME|COLUMN_NAME

----------|------------------------------

F0000194_0|SYEDBT,SYEDLN,SYEDTN,SYEDUS
F00022_0  |UKOBNM
F0006D_0  |LJLNGP,LJMCU
F0006S1_0 |SHSEFT,SHSVER
F0006S_0  |MCMCU,MCSEFT,MCSVER
F0006Z1_0 |SZEDBT,SZEDLN,SZEDTN,SZEDUS
F0006_0   |MCMCU
F0007Z1_0 |SZEDBT,SZEDLN,SZEDTN,SZEDUS
F0007_0   |CZCTRY,CZMMCU,CZMT,CZSHFT,CZWD
F0007_1   |CZMMCU,CZMT,CZSHFT,CZWDCK,CZWD
F0008B_0  |CDDFYJ,CDDTPN,CDFQ
```

Figure 5: Results of SQL Code 3

A similar output for NON-unique indexes can be generated by simply substituting 'NONUNIQUE' instead of 'UNIQUE' in the SQL command in *SQL Code 3*.

To maintain the integrity of the JD Edwards EnterpriseOne applications, never drop the unique constraint indexes.

Performance Indexes – Performance indexes are defined as those indexes that have been added to EnterpriseOne schema tables to address application performance issues at the database layer through its design or standard usage of the EnterpriseOne application. EnterpriseOne has been developed primarily for the purpose of OLTP transactions and so the non-unique indexes that were developed are mainly for the reasons of increasing its performance.

Analytics Indexes – EnterpriseOne does not distinguish between performance indexes for OLTP and those indexes that are used by certain reports for use in analytic data warehouse type queries that perform analytics. This poses a problem in the traditional approach of implementing the Oracle column store in EnterpriseOne. Although we can easily identify the unique constraint indexes, which never can be dropped to maintain data integrity, a deeper Oracle database analysis will be required for OLTP and analytic processes to extend the foundation of EnterpriseOne for more complex use for analytics.

Oracle Column Store Methodology for EnterpriseOne

Discussion of a methodology to determine what tables should be placed in the Oracle Column Store is the topic of this section of the document. There are two activity sections for this topic:

1. Identify ALL the tables that are used by the selected EnterpriseOne application and then to remove those tables that are not used by the Oracle optimizer in the column store as to tune only required tables.
2. Identify the EnterpriseOne tables that provide performance benefit when placed in the column store that benefits interactive user Ad-hoc inquiry activities.

Activity 1: Identify In-Memory Columnar Tables

The first activity in identifying EnterpriseOne tables for the column store involves identifying the processes that you want to test to see if they can benefit from the in-memory columnar format. It is best to test processes in isolation to clearly identify the tables involved and whether the SQL generated by this process would be used by the Oracle database optimizer in servicing the EnterpriseOne requests. In this testing, the Oracle database optimizer can choose from either row or column formats.

This approach will slowly add tables to the Oracle database column store that provide benefit to the EnterpriseOne application. This approach is taken to minimize the SGA memory required by the column store and provide a clear EnterpriseOne beneficial use of this feature.

The first activity involves identifying the processes that you want to test to see if they can benefit from the Oracle database column store. Test the processes in isolation to provide a clear relationship between performance characterization benefits.

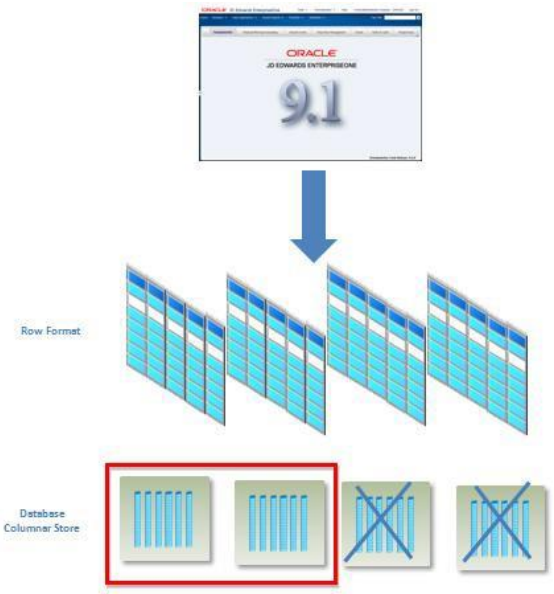

Figure 6: Columnar Format Methodology

Figure 6 above depicts the steps involved in this first activity. There are 5 steps for this activity:

Step 1: Initiate the EnterpriseOne process and collect an AWR report for this activity.

Collecting the AWR report will allow a database administrator to view all SQL ID queries to the database and identify all EnterpriseOne application tables that are involved in this process. These tables will be placed into Oracle column store.

Step 2: Place all identified EnterpriseOne tables involved in this process into column store.

For example, the crpdta.F0901 was identified in Step 1 as a candidate table for database column store. To place the table into columnar format and to force this table to be used upon the next query, perform the following:

```
-- Turn ON the Columnar Format Flag for the Table CRPDTA.F0901
ALTER TABLE CRPDTA.F0901 INMEMORY MEMCOMPRESS FOR QUERY DUPLICATE;
-- Force the LOAD of the CRPDTA.F0901 Table into the Columnar Format Memory (RAM)
select /*+ noparallel(F0901) full (F0901) */ count(*) from CRPDTA.F0901;
```

The table crpdta.F0901 now exists in both row and column store structures and, with the second SQL above, will force a load of this table into the SGA memory for future EnterpriseOne queries.

Step 3: Initiate the process again, generating an AWR report for this period of time.

Repeat the steps that were performed in Step 1, collecting a new AWR report.

Step 4: Using the AWR report snap_id begin and ending boundaries, review all of the execution plans.

Review all of the execution plans of the SQL statements issued to the database. The following example is an execution plan from an EnterpriseOne Sales Order Entry process that identified the table CRPDTA.F4201 that was using the database column store:

```
SQL_ID d7387g4hkr6ky
--------------------
SELECT  *  FROM CRPDTA.F4201  WHERE  ( SHAN8 = :KEY1 AND SHTRDJ <=
:KEY2 AND SHTRDJ >= :KEY3 AND SHDCTO IN  ( :KEY4,:KEY5 )  )  ORDER BY
SHAN8 ASC,SHTRDJ DESC,SHDOCO DESC

Plan hash value: 386102416

-------------------------------------------------------------------------------------
| Id  | Operation                   | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |       |       |       |  1041K(100)|          |
|   1 |  SORT ORDER BY              |       |     2 |  3480 |  1041K  (1)| 00:00:41 |
|   2 |   FILTER                    |       |       |       |            |          |
|   3 |    TABLE ACCESS INMEMORY FULL| F4201 |     2 |  3480 |  1041K  (1)| 00:00:41 |
-------------------------------------------------------------------------------------
```

Figure 7: Execution Plan Showing Columnar Scan

Figure 7 shows that the Oracle optimizer chose the CRPDTA.F4201 as an EnterpriseOne table that could be more efficient when placed into column store for the EnterpriseOne scenario that was defined. Repeating this analysis for all Oracle SQL_ID explain plans resulted in a list of Oracle database column store tables.

When a table is used in the column store, the execution plan will only report it in the format of 'TABLE ACCESS INMEMORY FULL' whereas in row format we can distinguish access to the row format as a 'RANGE SCAN', 'FULL TABLE SCAN' and other access methods.

Step 5: Remove the In-Memory Column Store Tables NOT used by EnterpriseOne

This last step is to optimize the use of the Oracle database column store tables and thus reducing the unnecessary overhead of SGA memory needed to maintain these. Disabling the database column store tables can be done using the following SQL command:

```
-- Turn OFF the Column Store Flag on for the table CRPDTA.F0901
ALTER TABLE CRPDTA.F0901 NO INMEMORY;
```

Placing an EnterpriseOne table in and out of the column store does not require a recycle of Oracle services; this functionality is dynamic for the Oracle database.

At this stage we have tuned the EnterpriseOne application for use with the Oracle database column store tables, identified the applications that will benefit, and have gathered statistics as to the benefit of implementation.

This activity should be repeated for a number of EnterpriseOne interactive and batch processes. In the end, a list of EnterpriseOne tables will be identified that will be used by the OLTP and analytic processes that are known to use the Oracle column store. The last part of this activity is to test the EnterpriseOne application processes as a whole and characterize the final impact to performance of the Oracle database column store.

Activity 2: Ad-Hoc Inquiry Applications and Non-indexed Columns

A second finding concerning the Oracle Database In-Memory feature is that it can help with JD Edwards EnterpriseOne processes that are of the inquiry type without the user having to ask for a new performance index to be developed. **Inquiry Application** queries in JD Edwards EnterpriseOne have the Oracle database characteristic of performing a 'FULL TABLE SCAN' on the table that the database query is attempting to access data from. A database request over a non-indexed column is one of the main reasons for a full table scan.

Full table scans are normally the most inefficient way of finding a result set. The scan is a sequential read of each row of the table and the specific columns in the conditional clause of the SQL is then checked for meeting its criteria. Full table scans are very costly and slow due to the heavy amount of I/O reads that are required. Because I/O is a critical component to performance, placing these reads into memory, such as the case of columnar format, should improve performance.

There are a number of categories that EnterpriseOne Inquiry Applications can fall under:

1. **Standard EnterpriseOne Inquiry Applications** - Examples such as customer service inquiry, supplier inquiry, and category code searches have been developed by JD Edwards EnterpriseOne for key application information requests. Specific performance indexes have been added to these applications to increase performance. The Oracle In-Memory column store can be evaluated as a solution to replace these performance indexes.
2. **Standard Customer EnterpriseOne Application Inquiries** - Indexes added by the customer to solve problems of performance with those specific inquiries that may be unique to their business vertical. These inquiries can be re-evaluated by replacing the index with the Oracle In-Memory column store, or where a full table scan is identified avoiding the possibility of adding another index to an existing EnterpriseOne table.
3. **General Ad-Hoc Inquiries** - General Ad-hoc queries are those database requests that are issued by the interactive user that is out of the normal use of the EnterpriseOne application. JD Edwards EnterpriseOne allows the interactive user to perform an inquiry of data on a single or multiple columns in the application grid. As such, the resulting database request may result in a full table scan because the request is over a non-indexed column. The Oracle In-Memory feature may assist in satisfying the performance conundrum that can arise from general Ad-hoc inquiries that perform faster when the table is placed in the column store.

The process of adding EnterpriseOne tables into the column store to improve performance of general Ad-hoc inquiry application activities include the following process:

1. Place the EnterpriseOne tables used in interactive user general Ad-hoc inquiry applications into the column store.
2. Evaluate whether the Oracle In-Memory column store is being used by the inquiry application as is illustrated in Figure 8 that will satisfy an interactive user database request.
3. If interactive users make these requests on a frequent basis and it benefits their productivity, then retaining these tables in the Oracle In-Memory column store should be considered.

```
SQL_ID 16h0htguhybb2
--------------------
SELECT T0.ATAST, T0.ATPRGR, T0.ATCPGP, T0.ATPRFR, T0.ATLBT, T0.ATOLVL,
T1.GPGPTY, T1.GPGPC, T1.GPGPK1, T1.GPGPK2, T1.GPGPK3, T1.GPGPK4,
T1.GPGPK5, T1.GPGPK6, T1.GPGPK7, T1.GPGPK8, T1.GPGPK9, T1.GPGPK10,
T2.GPGPTY, T2.GPGPC, T2.GPGPK1, T2.GPGPK2, T2.GPGPK3, T2.GPGPK4,
T2.GPGPK5, T2.GPGPK6, T2.GPGPK7, T2.GPGPK8, T2.GPGPK9, T2.GPGPK10,
T3.HYPRFR, T3.HYHYID, T3.HYHY01, T3.HYHY02, T3.HYHY03, T3.HYHY04,
T3.HYHY05, T3.HYHY06, T3.HYHY07, T3.HYHY08, T3.HYHY09, T3.HYHY10,
T3.HYHY11, T3.HYHY12, T3.HYHY13, T3.HYHY14, T3.HYHY15, T3.HYHY16,
T3.HYHY17, T3.HYHY18, T3.HYHY19, T3.HYHY20, T3.HYHY21 FROM
PRODDTA.F4071 T0,PRODDTA.F4092 T1,PRODDTA.F4092 T2,PRODDTA.F40073 T3
WHERE  ( T0.ATAST IN  ( :KEY1,:KEY2,:KEY3,:KEY4,:KEY5,:KEY6,:KEY7,:KEY8,
:KEY9,:KEY10,:KEY11,:KEY12,:KEY13,:KEY14,:KEY15,:KEY16 )  AND T0.ATOLVL
= :KEY17 AND T0.ATLBT = :KEY18 AND  ( T0.ATPRGR = :KEY19 OR T0.ATPRGR
<> :KEY20 AND T1.GPGPTY = :KEY21 )  AND  ( T0.ATCPGP = :KEY22 OR
T0.ATCPGP <> :KEY23 AND T2.GPGPTY = :KEY24 )  )  AND  ( T0.ATPRGR =
T1.GPGPC (+)  AND T0.ATCPGP = T2.GPGPC (+)  AND T0.ATPRFR = T3.HYPRFR )
 ORDER BY T0.ATAST ASC

Plan hash value: 2598830176

-----------------------------------------------------------------------------------------------
| Id  | Operation                           | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |          |       |       |    23 (100)|          |
|   1 |  SORT ORDER BY                      |          |     1 |   611 |    23   (5)| 00:00:01 |
|   2 |   FILTER                            |          |       |       |            |          |
|   3 |    HASH JOIN OUTER                  |          |     1 |   611 |    22   (0)| 00:00:01 |
|   4 |     JOIN FILTER CREATE              | :BF0000  |     1 |   381 |    18   (0)| 00:00:01 |
|   5 |      FILTER                         |          |       |       |            |          |
|   6 |       HASH JOIN OUTER               |          |     1 |   381 |    18   (0)| 00:00:01 |
|   7 |        JOIN FILTER CREATE           | :BF0001  |     1 |   151 |    14   (0)| 00:00:01 |
|   8 |         NESTED LOOPS                |          |     1 |   151 |    14   (0)| 00:00:01 |
|   9 |          NESTED LOOPS               |          |     1 |   151 |    14   (0)| 00:00:01 |
|  10 |           TABLE ACCESS STORAGE FULL | F4071    |     1 |    62 |    13   (0)| 00:00:01 |
|  11 |           INDEX RANGE SCAN          | F40073_2 |     1 |       |     0   (0)|          |
|  12 |          TABLE ACCESS BY INDEX ROWID| F40073   |     1 |    89 |     1   (0)| 00:00:01 |
|  13 |        JOIN FILTER USE              | :BF0001  |    99 | 22770 |     4   (0)| 00:00:01 |
|  14 |         TABLE ACCESS STORAGE FULL   | F4092    |    99 | 22770 |     4   (0)| 00:00:01 |
|  15 |     JOIN FILTER USE                 | :BF0000  |    99 | 22770 |     4   (0)| 00:00:01 |
|  16 |      TABLE ACCESS STORAGE FULL      | F4092    |    99 | 22770 |     4   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
```

Figure 8: Full Table Scan Example

Figure 8 above is an example of a full table scan being performed on the I/O storage subsystem from an inquiry application.

Full table scan queries are often suppressed by customers through EnterpriseOne security, hiding the grid column for open searches on non-indexed columns. Non-indexed columns will most likely generate a slow executing query resulting from an Oracle full table scan. Thus, placing the above tables into the column should be more efficient. In the case illustrated above in Figure 8, an improvement of performance was seen in the Ad-hoc query when the tables F4071 and F4092 were placed in the column store. The Oracle optimizer determined that the performance of the column store was superior to that of the I/O storage.

Dropping JD Edwards EnterpriseOne Indexes

Question #3: If Database In-Memory is scanning JD Edwards EnterpriseOne tables in memory, can I drop my database indexes? How do I decide?

The final discussion point of this document is the dropping of JD Edwards EnterpriseOne indexes. Dropping indexes should only be initiated after activities 1 and 2 of the previous section have been followed. The number of tables that the Oracle optimizer uses for column store and the indexes that were used in row format will help in evaluating which indexes, if any, can be dropped.

Special Highlight 3: For the purposes of this exercise, only those tables involved in the column store need to be evaluated for the dropping of indexes.

One method of determining which JD Edwards EnterpriseOne column store indexes to drop is to use the Oracle MONITORING USAGE capabilities. To enable an index to be monitored by the database for future analysis, perform the following command:

```
ALTER INDEX <index_name> MONITORING USAGE
```

where <index name> is the name of the EnterpriseOne table index. For example, if the index for F4201_2 containing the columns SHAN8 and SHVR01 was to be monitored, the following command would be issued:

```
ALTER INDEX CRPDTA.F4201_2 MONITORING USAGE;
```

To disable this feature for the CRPDTA.F4201_2 table use the following Oracle SQL command:

```
ALTER INDEX CRPDTA.F4201_2 NOMONITORING USAGE;
```

The steps involved in dropping an EnterpriseOne table index involve:

1. Identify the EnterpriseOne indexes that are candidates for dropping
2. Monitor the index usage with the Oracle 'ALTER INDEX' command
3. Evaluate over time the usage patterns and determine if the index, in fact can be dropped by evaluating whether the index is used or not by running *SQL Code 4* below

SQL Code 4 lists the Oracle command for information concerning this index usage:

```
-- SQL To determine index usage when monitoring is enabled
-- Must be logged in to SQL as object owner to view usage
SELECT index_name,
    table_name,
    monitoring,
    used,
    start_monitoring,
    end_monitoring
FROM v$object_usage
ORDER BY index_name;
```

SQL Code 4: Index Usage Query

```
INDEX_NAME
--------------------------------------------------------------------------------
TABLE_NAME
--------------------------------------------------------------------------------
MON USE START_MONITORING    END_MONITORING
--- --- ------------------- -------------------
F4201_0
F4201
NO  NO  11/14/2014 12:13:23 11/14/2014 12:23:02

F4201_2
F4201
NO  NO  11/14/2014 12:27:08 11/14/2014 12:36:26
```

Figure 9: Output of Index Monitoring

Figure 9 above shows the results of the database query in SQL Code 4. In SQL Code 4, the v$object_usage view does not contain an owner column. When looking for information in the v$object_usage table for the EnterpriseOne CRPDTA data source, the CRPDTA table owner must issue the above command. Figure 9 illustrates that indexes either F4201_0 or F4201_2 have been used during the capture of the monitoring usage statistics.

Although indexes are useful in the Oracle database's capability in responding to requests quickly, indexes do have an overhead to performance:

- Disk Space - Disk space attributed to indexes for the EnterpriseOne application has been measured to be approximately ¼ of the size of the database in our testing environment.
- Hardware Resources - Both processor and memory resources are consumed with Oracle indexes.
- Maintenance - Maintaining new indexes for custom applications within EnterpriseOne and adjusting for them during upgrade processes is a large task for many customers.
- Performance and Optimization - While indexes can assist with performance in one area of the EnterpriseOne application, it can in fact make other processes run less efficiently. Dropping and adding indexes should be done with care. Furthermore, dropping an index will force the Oracle optimizer to choose an alternate execution path, one that may not be as efficient. This is further complicated by the specifics of the data that is being processed and the data distribution.

Thus any dropping of any unused EnterpriseOne indexes can have benefits in recovered disk space, CPU/Memory resources, and required maintenance.

One last point about the use of the Oracle command to monitor usage, it should be done for a specific time period with knowledge of the workload that is being monitored for best results in analysis. The time period must be sensitive to changes in usage of the table and indexes for processes that fall within specific hours, days, week, quarter, or yearly events that can change the index usage query results with changing EnterpriseOne processes.

Conclusion

The Oracle Database In-Memory option has the potential to dramatically transform how JD Edwards EnterpriseOne customers use their valuable data assets. Without modification to existing applications or disruption to existing operations, JD Edwards EnterpriseOne customers can extend their OLTP data to open new analytic capabilities, and thereby unlock untapped value from the data. By providing a dual-format database, Oracle Database In-Memory provides a technically incremental path to a functionally transformational transition into the age of In-Memory computing.

1) Oracle Database In-Memory can be implemented for JD Edwards EnterpriseOne databases at a low memory cost because of its high compression ratio. That means that many customers may not have to upgrade the memory on their servers if they have sufficient resources in their current SGA to implement Oracle Database In-Memory.

2) The In-Memory Column Store has the added benefit of decreasing the demand on IT to create and maintain customized reports and indexes for analytic applications and usage patterns within JD Edwards EnterpriseOne. Putting JD Edwards EnterpriseOne tables into the In-Memory column store can eliminate the need to create reports and add table indexes because they use non-indexed columns. Similar to the analysis for Ad-hoc queries, customers may find a hidden benefit in this functionality.

3) The methodologies outlined in this technical brief provide database administrators with approaches to implementing and optimizing Oracle Database In-Memory to provide the maximum benefit for both JD Edwards EnterpriseOne OLTP transactions and new analytic usage patterns.

See Also:

» Web site: Oracle Database In-Memory
http://www.oracle.com/us/products/database/options/database-in-memory/overview/index.html

» Technical Brief: Oracle Database In-Memory
http://www.oracle.com/technetwork/database/in-memory/overview/twp-oracle-database-in-memory-2245633.html?ssSourceSiteId=ocomen

» Oracle Database Documentation Library: https://docs.oracle.com/database/121/nav/portal_4.htm

» Technical Brief: Performance Characterization of JD Edwards EnterpriseOne with Oracle Database In-Memory
https://apex.oracle.com/pls/apex/f?p=44785:141:109388643419968:::P141_PAGE_ID%2CP141_SECTION_ID:121%2C1344

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

CONNECT WITH US

blogs.oracle.com/oracle
facebook.com/oracle
twitter.com/oracle
oracle.com

Hardware and Software, Engineered to Work Together

Oracle Database In-Memory with Oracle's JD Edwards EnterpriseOne
January 2015