

Oracle Database 12c – Built for Data Warehousing

ORACLE WHITE PAPER | FEBRUARY 2015





Contents

Executive Summary	1
Overview	2
A Brief Introduction to Oracle's Information Management Reference Architecture	3
Modeling Your Data	6
Hardware Architecture Considerations	9
Data Management - Managing High Data Volumes	11
Data Ingestion - Efficient Loading and Transformation	14
Information Interpretation - Optimizing Query Performance	17
System Management	24
Conclusion	28
Further Reading	29



Executive Summary

Data warehousing is undergoing a major transition with the arrival of the big data era. The Enterprise Data Warehouse (EDW) has traditionally sourced data solely from other databases, but organizations have increasingly recognized that there is a huge amount of data in today's world that is not captured in operational databases. Information systems need to harness all sources of data and allow enterprises to analyze and extract business value from them.

However, the new information architecture of big data and data warehousing systems will pursue big data as an extension to their previous data warehousing architectures. The EDW, built upon a relational database, continues to be the primary analytic database for storing a company's core data: financial records, customer data, point-of-sale data and so forth. The new information architecture requires data to flow between big data and data warehousing systems in order to deliver a unified foundation for analytics, and the data warehouse will become the unified foundation of analysis. With the explosion in data diversity and data volumes, there is a demand for more information sooner; analytics must be delivered to an ever-widening set of users and applications. EDWs must be faster, more efficient, more flexible, and easier to manage than they have ever been before.

Within this rapidly evolving business context, what must you consider in order to implement a successful Oracle Database 12c data warehouse and to build the unified foundation for analysis? This paper is intended for data warehousing architects, developers and administrators; so that they can become conceptually familiar with all of the relevant technology components of the Oracle database and then be in a position to answer that question.

Overview

The first part of this paper presents some high-level context: it briefly introduces Oracle's Information Management Reference Architecture followed by an overview of how best to structure your data for data warehousing. The rest of the paper is dedicated to defining key technical challenges and how Oracle Database 12c functionality is used to address them.

Specifically, this paper will cover:

- » Choosing a scalable data warehousing platform.
 - » Introducing Oracle Exadata Database Machine.
- » Managing and storing large volumes of data effectively.
 - » Partitioning for manageability.
 - » Storing data efficiently with compression.
- » Loading and transforming data efficiently.
 - » Loading in batch and near real-time.
 - » Using set-based and row-based processing.
 - » Improving response times and scaling with parallelism.
 - » Using a consistent and flexible approach with external tables.
- » Optimizing query performance.
 - » Partitioning for performance.
 - » Improving response times and scaling with parallelism.
 - » Features for optimizing the physical data model.
- » Managing an EDW.
 - » Managing and allocating resources with Database Resource Management.
 - » Monitoring and managing the database using graphical user interfaces: Oracle Enterprise Manager Cloud Control or Oracle Enterprise Manager Express.
 - » Managing optimizer statistics for optimum query performance.

This paper is by no means a complete guide for data warehousing with Oracle technology but it can be considered a good starting point and reference guide. Other Oracle collateral is referenced in this document so that you have a path to drill down into the next level of detail. Ultimately, Oracle's Database documentation, especially the Oracle Data Warehousing Guide and the Oracle VLDB and Partitioning Guide, detail all data warehousing features of Oracle Database 12c at the finest level of granularity.

A Brief Introduction to Oracle's Information Management Reference Architecture

Introduction

Oracle's Information Management Reference Architecture describes, at a high level, the components of an enterprise information architecture. It defines the components necessary to collect, store, and use enterprise data for the purposes of information access and analysis.

The reference architecture defines a number of components or layers. It is important to have a broad understanding of each of these components because they are all likely to be represented in the final physical design of most EDWs; at least to some extent:

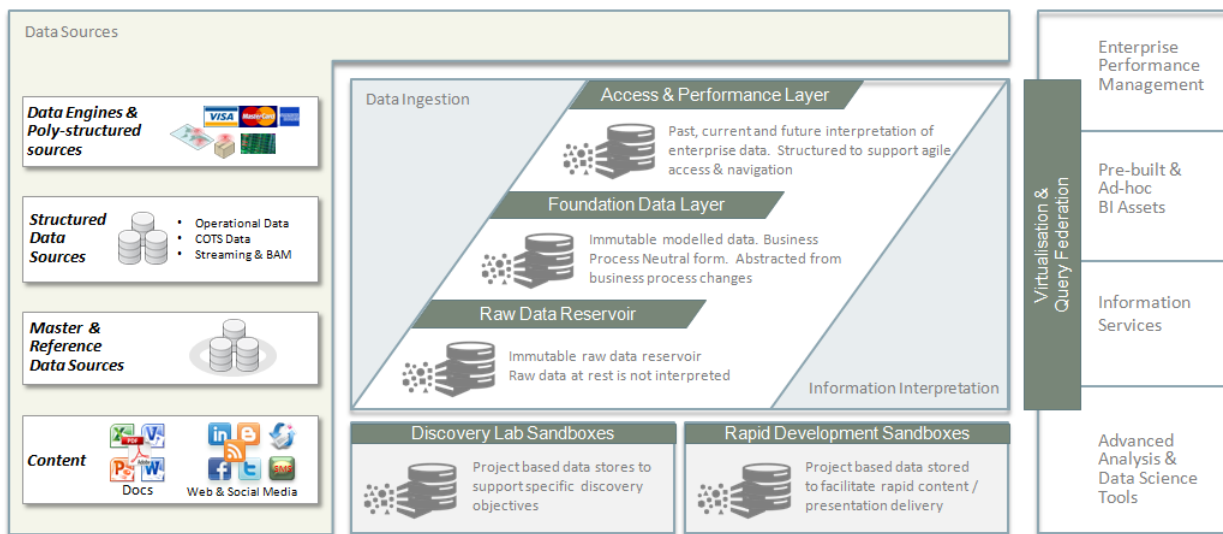


Figure 1: Core Components of Oracle's Information Management Reference Architecture. See Reference 1 in the *Further Reading* section at the end of this paper.

It is useful to start by exploring the data management layers shown in the centre of Figure 1 (Raw, Foundation and Access and Performance) because they are fundamental to understanding the other components in the architecture.

Data Management Layers

The reference architecture defines three layers for storing data, and the extent to which these layers are required in a particular solution will depend on its functional and non-functional requirements. The creation of these layers and the progression of data from the Raw Data Reservoir to Foundation and then onto Access and Performance is motivated by a need for:

- » Increased data quality and enrichment
- » Increased formalization of definition
- » Increased "ease of use", simplified data model
- » Reduced cost of query concurrency

The Raw Data Reservoir is a data store that holds data at the finest level of granularity. In its most basic form, a Raw Data Reservoir will be a file system staging area, where transient flat files are stored prior to being loaded into

the data warehouse. Increasingly, big data technologies (such as the Hadoop Distributed File System) are used to stage data, but also to offer long term persistence and pre-defined ETL/ELT processing.

The Foundation Data Layer is used to store the complete and trusted enterprise dataset in a well defined and structured way. The Access and Performance Layer will present some or all of the foundation layer data in an application friendly, easy-to-navigate star (or snowflake) schema that is accessed using a wide variety of tools for BI and Analytics. The Access and Performance Layer will often need to support high concurrency, fast response times and consistent performance.

The reference architecture is completely agnostic to the underlying physical implementation but, for the purposes of this paper, it will be assumed that the Foundation Data Layer and the Access and Performance Layer will be implemented using Oracle Database 12c. The Raw Data Reservoir will be assumed to be either a simple file system or a Hadoop distributed file system. Data ingestion and Information Interpretation will be discussed within the context of how they interact with an Oracle database.

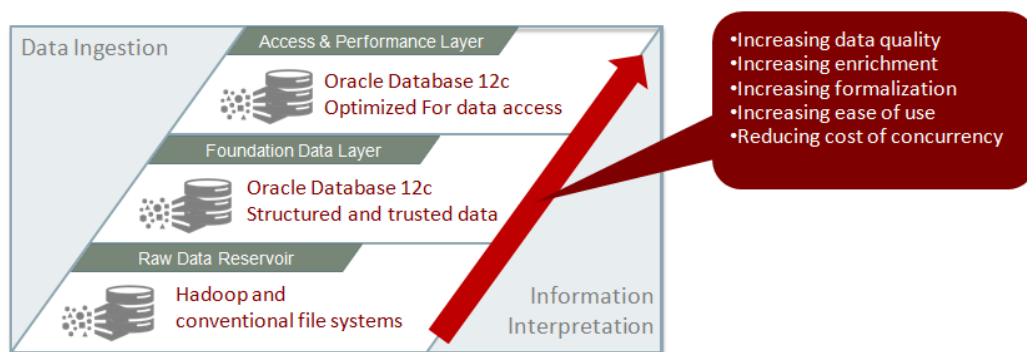


Figure 2: The physical implementation proposed in this paper.

As mentioned above, there is no absolute requirement for an EDW to implement all of these data layers. The service levels of a particular EDW might make it appropriate to expose some or even all of the Foundation Data Layer to the Information Interpretation component, potentially mitigating the need for a separate Access and Performance Layer. However, because business requirements are rarely static, it is important to design a solution with this overarching architecture in mind. This will make it easier to use the features available in Oracle Database 12c to evolve the EDW as the scope and the needs of the business change over time.

Data Ingestion

Data Ingestion is responsible for moving, cleaning and transforming data. It uses techniques such as near real-time streaming, Extract Transform and Load (ETL) and Extract Load and Transform (ELT). Data Ingestion loads data from external systems, but it also presents, loads and transforms data between different data layers. Finally, Data Ingestion may be used to move and transform data inside each data layer.

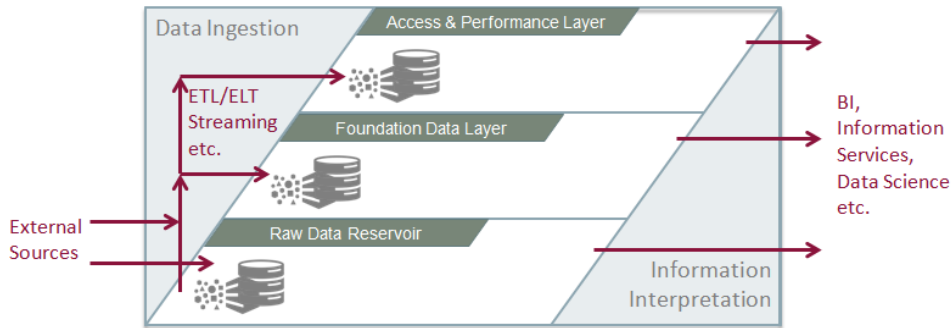


Figure 3: Data movement and presentation in an EDW.

Information Interpretation

As indicated in Figure 3, the Information Interpretation component presents information to external systems and services. It may interface with any of the data management layers. The Information Interpretation component will be implemented using a variety of Analytical, Data Science and BI tools. Oracle Advanced Analytics and rich SQL capabilities, such as SQL pattern matching, make it possible to analyze data inside the Oracle database, simplifying the Information Interpretation layer, improving performance and reducing the need to move data between different systems and data layers.

High-Level EDW Technical Requirements

Using the components in the Information Management Reference Architecture as a foundation, it is possible to detail the key high-level technical requirements for an EDW.

A flexible and successful EDW will require:

- » A scalable and balanced hardware platform.
- » The ability to store and manage large volumes of data cost effectively, while meeting reliability, availability and performance service levels.
- » Efficient data load and transformation, in batch and real time.
- » Big data integration for easy data movement and logical data presentation.
- » Application-transparent optimizations for physical data models: optimizations that will enable a system to support high levels of concurrency and low query response times, even in ad-hoc query environments.
- » Integrated analytics and feature-rich analytical SQL for high performance Information Interpretation.
- » Tools to monitor, manage and optimize the environment as a whole.

This paper will present, at a high level, how to meet these requirements with Oracle Database 12c.

Modeling Your Data

This section introduces the logical data models used in data warehousing. It then presents a brief view of how logical models are evolved into a physical implementation within an Oracle 12c relational database.

In this paper a logical model for a data warehouse will be treated more as a conceptual or abstract model, an ideological view of what the data warehouse should be. The physical model will describe how the data warehouse is actually built in an Oracle database.

The Logical Model

A logical model is an essential part of the development process for a data warehouse. It allows you to define the types of information needed in the data warehouse to answer the business questions and the logical relationships between different parts of the information. It should be simple, easily understood and have no regard for the physical database, the hardware that will be used to run the system or the tools that end users will use to access it.

There are two classic models used for data warehouse, Third Normal Form (3NF) and dimensional or Star (Snowflake) Schema.

Third Normal Form (3NF) is a classical relational-database modelling technique that minimizes data redundancy through normalization. Using Figure 4 as an example, rows in the physical employees table will not include any department attributes such as “department name”. In the physical implementation, each employee row will have a foreign key that will link it to the relevant department row. In this way, “department name” is not repeated for each employee row; it is held once inside a single row of the departments table. Normalization, the process of removing redundancy, will typically result in a large number of tables. By definition, 3NF preserves a detailed record of each transaction without any data redundancy and allows for rich encoding of attributes and all relationships between data elements. Users typically require a solid understanding of the data in order to navigate the more elaborate structures reliably. A 3NF data model is commonly used in the Foundation Data Layer.

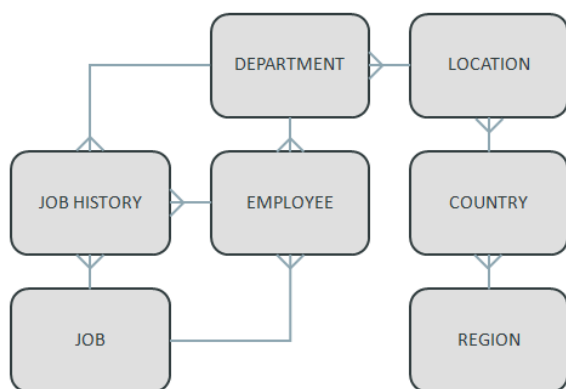


Figure 4: An entity relationship diagram for a typical 3NF schema.

The star schema is so-called because the diagram resembles a star, with points radiating from a center. The center of the star consists of one or more fact tables and the points of the star are the dimension tables.



Figure 5: A graphical representation of a star schema.

Fact tables are the large tables that store business measurements and typically have foreign keys to the dimension tables. Dimension tables (also known as lookup or reference tables) contain relatively static or descriptive data in the data warehouse. Using Figure 5 as an example, a corresponding physical 3NF implementation of this schema might have tables for products, product categories and product types. In contrast to this, the star schema's physical implementation will typically have a single products table, which will simply include columns for product category and product type. From this example, it's easy to see that the snowflake schema accepts some level of data redundancy; the product type and product category column values will be repeated throughout the products table. The cost of this redundancy is offset with the benefit of simplicity because fewer tables need to be understood, navigated and joined when queries are being constructed. This, at least in part, is what makes navigation of the model so straightforward for end users, so it frequently used in the Access and Performance Layer.

There is often much discussion regarding the 'best' modeling approach to take for any given data warehouse with each style, classic 3NF or dimensional having their own strengths and weaknesses. Most modern EDWs need to embrace the benefits of each model type rather than rely on just one. This is the approach that Oracle adopts in its Information Management Reference Architecture: the majority of EDW implementations use a mixture of both model forms. It is important that you design your model according to your specific business needs.

The Physical Model

The starting point for the physical model is the logical model. The physical model should mirror the logical model as much as possible, although some changes in the structure of the tables and columns may be necessary. In addition, the physical model will include staging or maintenance tables that are usually not included in the logical model.

The transition from the logical to the physical model is influenced by non-functional requirements: data integrity requirements may demand the use of certain constraints, primary keys and foreign keys. Scalability and response-time requirements will influence the use of particular optimizations such as data partitioning, pre-built aggregations and indexes.

Oracle Database 12c and Oracle's Engineered Systems provide optimizations and features that are all designed to operate transparently with respect to the Information Interpretation component. This offers great flexibility; the

physical model can be optimized and evolved without any requirement to make changes to the applications that interact with the database. The Oracle database provides optimizations that can:

- » Simplify the overall physical implementation. A simplified physical model has operational benefits: it will reduce the number of physical structures that we need to store, manage and maintain. For example, if we enable an In-Memory column store or use an Oracle Exadata Database Machine (both discussed later), it is likely that we will reduce our reliance other physical optimizations such as indexing.
- » Improve scalability and query concurrency. Features that optimize the physical model are not mandatory, but they optimize the way data is accessed. The effect of this is to reduce the amount of system resources that are consumed when queries and transformations are executed. In this way, the physical optimizations available with the Oracle database make it possible to deliver data warehouses that support many concurrent users and processes, but are able to maintain consistently low response times.
- » Reduce the risk of keeping pace with change and reduce the need to get everything “right first time”. It is possible to extend the capabilities of an existing implementation without having to change applications. For example, partitioning can be added to enhance manageability, availability and performance without requiring any changes to be made to the tools and products that constitute the Information Interpretation component.

Figure 6 illustrates some of the features that distinguish the logical and physical design.

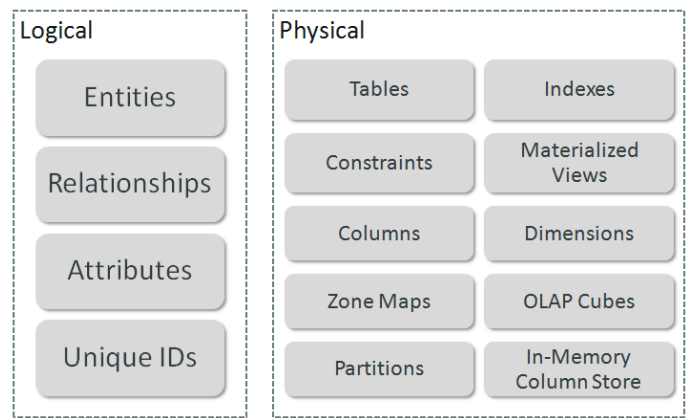


Figure 6: Comparing the logical design with various physical attributes of an Oracle database.

The graphical user interface tool, Oracle SQL Developer is useful for the physical design process. Also, there are utilities collectively known as “advisors” that can be used in an on-going manner to identify where optimizations will be useful. Examples include the SQL Tuning Advisor and the SQL Access Advisor.

This paper focuses on the implementation of a physical model. In particular, it covers features and techniques available in Oracle Database 12c that belong in one or more of the following categories:

- » Essential for data warehousing.
- » Useful in the data warehousing context, where we can use them to enhance and simplify physical implementations while remaining transparent to applications.

Hardware Architecture Considerations

Building a successful hardware infrastructure carries risk: the interplay of hardware, firmware and software is complex. Designing and building a balanced infrastructure that is both reliable and easy to scale out quickly requires considerable expertise. Oracle's Exadata Database Machine not only delivers a reliable, high performance platform, but scalability is also fundamental to its design.

Oracle Exadata Database Machine

The Oracle Exadata Database Machine is a high performance, purpose-built Engineered System for the Oracle database that mitigates the risk and complexity that is inherent in build-your-own solutions. Its primary technical benefits in a data warehousing context can be summarized as follows:

- » Extreme IO performance; the ability to read and write data at very high rates.
- » Extreme scalability; both in terms of scalable storage infrastructure and also the ability to scale even a single database across multiple database servers and multiple Exadata racks.
- » A balanced configuration; the components of the system are matched in terms of performance so that they are able to work together as a coherent whole.
- » Absolute application transparency.

Scalability and performance is fundamental to the design: Exadata Storage Servers enable scans and other resource-consuming features to be off-loaded away from the database servers and multiple Exadata racks can be connected together using a unified Infiniband fabric. Multiple database servers can be connected together to form a single cluster than can be distributed across multiple Exadata racks.

Exadata is transparent to applications and is largely transparent to the administration of the database; many of its features require minimal or no manual intervention or configuration.

Exadata is very effective at enabling data warehouses to fulfill new business requirements with minimal configuration or database management overhead. A full technical overview is presented in Reference 2 (see *Further Reading*, below), but this is a summary of the features that most clearly and directly benefit data warehousing environments:

- » Data warehousing environments require high performance interconnects between database servers and the storage infrastructure because queries and transformations are likely to read and write large volumes of data.
 - » An Infiniband¹ fabric connects together the Exadata database servers and the Exadata Storage Servers.
- » A high performance data channel or interconnect is required between database servers to enable parallel processing to scale across a cluster of machines.
 - » Exadata uses the Infiniband fabric to connect the cluster members together.
- » Offloading data scans to storage infrastructure will benefit data warehousing workloads.
 - » Exadata Smart Scan offloads scans (automatically and transparently) to Exadata Storage Servers, reducing database server CPU consumption and the amount of data that needs to be transported across the Infiniband fabric.

¹ Infiniband bandwidth in Exadata X4 is 40Gbits/sec. See <http://www.oracle.com/us/products/servers-storage/networking/infiniband/index.html>

- » Data residing in flash memory can be accessed much faster than data residing on disk.
 - » Exadata Smart Flash Cache provides a flash memory cache layer on the storage cells. Data placement on disk and flash is managed automatically by the Exadata Storage Server Software. Scan rates and maximum IOs per second (IOPS) for flash memory significantly exceed the corresponding rates for disk.
- » Queries and transformation that scan data will benefit from optimizations that transparently reduce the total amount of data that needs to be accessed.
 - » Exadata Storage Indexes are in-memory index structures. They are automatically maintained inside the storage cells to enable Smart Scan to skip regions of table data that are filtered out by query predicates.
- » Compression will reduce the amount of storage required to hold data, and it will often reduce the amount of data that needs to be scanned when executing queries.
 - » Exadata Hybrid Columnar Compression (EHCC, also referred to more simply as HCC) can deliver high compression ratios and (often) improved query response times. EHCC is transparent to the application layer and is covered in more detail below.

Non-Exadata Platforms

If an Oracle Exadata Database Machine is not used, care must be taken to ensure that all hardware resources – CPU, memory, network and IO – are built as balanced system to avoid any hardware-related bottlenecks. IO paths in particular must have sufficient bandwidth to support the required workload. Storage management (such as the Oracle Automatic Storage Manager) will be necessary to guarantee ease of management and to distribute IO across storage devices to avoid being constrained by IO “hot spots”.

Oracle Real Application Clusters (RAC)

RAC technology enables Oracle databases to scale horizontally across multiple machines that have shared access to a common storage infrastructure. RAC is well suited to EDW because it can be combined with Oracle parallel processing to scale workloads across a cluster of servers. RAC requires a data interconnect between all the servers in the cluster. To maximize the scalability and flexibility of the cluster, the RAC interconnect should be designed to sustain a throughput that is similar to the disk subsystem. If this design recommendation is followed, queries will scale well when executed in parallel across the entire cluster. Parallel execution is covered in more detail later on.

RAC is a key technology used in the Oracle Exadata Database Machine.

Storage

By definition, EDWs require a large amount of storage so it is especially important to ensure that manageability and scalability is kept as simple as possible. Oracle Automatic Storage Management (ASM) is a feature built into the Oracle database that significantly contributes to the reliability, availability, manageability and performance of an EDW. ASM distributes database IO across all available storage devices, it includes data protection features and allows a data warehouse to be scaled up (or down) in a seamless manner without outage. For more information on ASM, see Reference 2 in *Further Reading*, below.

Many data warehouse operations are based upon large table scans and other IO-intensive operations. All components in the storage infrastructure must be optimized to sustain high levels of IO throughput. This includes all critical components on the IO path such as the host bus adapters (HBAs), fiber channel (FC) connections, PCI flash cards, storage controllers and storage units.

Careful storage infrastructure design is particularly important in modern systems because Oracle Database 12c has the potential to read storage at rates in excess of 500 megabytes per second *per processor core*. Crucially, the overall system will need *balance*: CPU power and memory capacity is matched to the maximum IO bandwidth and IOPS available from disk, flash memory or solid state disk.

Data Management - Managing High Data Volumes

Managing hundreds of terabytes or even petabytes of data demands efficient and scalable data management. Oracle Database 12c has features specifically designed to give you the ability to make this possible. Larger tables must be partitioned for optimal and scalable data management and data warehousing environments naturally lend themselves to time-based partitioning. Subdividing partitions into smaller sub-partitions is used to improve the manageability of warehouses that store extremely large volumes of data and this technique is also useful for improving query join and scan performance.

Compression optimizes the cost of ownership and improves performance. It integrates extremely well with partitioning, enabling the database administrator to optimize storage utilization and target the use of compression appropriately throughout the data lifecycle.


Partitioning for Manageability

Partitioning allows a table or index to be subdivided into smaller pieces. Each discrete piece of the database object is called a partition and each partition has its own name and (optionally) its own storage characteristics. From the perspective of a database administrator, a partitioned object has multiple pieces that can be managed either collectively or individually. From the perspective of the application, a partitioned table is identical to a non-partitioned table; no modifications are necessary when accessing a partitioned table using queries or data manipulation language (DML) commands such as INSERT, UPDATE, DELETE and MERGE.

Partitioning can provide tremendous benefits to a wide variety of applications by improving manageability, availability, and performance without having to make any modifications to the Information Interpretation component (the performance improvements of Partitioning are discussed later in this paper). A typical usage of partitioning for manageability is to support a “rolling window” load process in a data warehouse. Suppose data is loaded into a table on an hourly basis; that table could be range-partitioned so that newer partitions contain hourly data while older partitions are partitioned by day, week, or even by month. The load process is simply the addition of a new partition. Another advantage of using partitioning is when it is time to remove data; an entire partition can be dropped, truncated or “removed” in a single, fast operation. The “removal” in this scenario is a so-called partition exchange that replaces a partition containing data with an empty table; the data is logically removed from the table but still existent as standalone table in the database for archival purposes.

Partitions themselves can be divided into sub-partitions. This is known as composite partitioning. It can be useful for very large datasets where it becomes beneficial to subdivide large partitions down into smaller and more manageable units. Composite-partitioned tables and indexes can yield performance benefits; covered later in this paper.

EDWs frequently use time-based range partitioning on fact tables because this readily supports the most common use-case: large volumes of time-based data arriving continuously with a requirement to compress old data for long-term retention (maintaining the ability to query it). It is common for more recent data to be queried more frequently, and again this partitioning model works particularly well with this scenario (covered in this paper).



The performance gains provided by partitioning may enable database administrators to complete maintenance operations on large database objects in relatively small batch windows. Specifically, partitioning can be used to break down a variety of tasks into smaller, more manageable units of work. For example:

- » Fact tables can be reorganized one partition at a time without disturbing query activity. For example, compressing individual partitions and moving data to different storage tiers.
- » Indexes can be built on a partition-by-partition basis.
- » Bulk updates can be done incrementally, one partition at a time (potentially updating multiple partitions concurrently).
- » Bulk deletion is easy and fast using *truncate* statements on table partitions.
- » Optimizer statistics can be gathered incrementally at the partition level.

Partitioning is central to Oracle's ability to manage data throughout its lifecycle. For example:

- » When and how to compress table data, while maintaining the ability to query it.
- » Storing “hot” data on fast storage and “cool” data on slower, cheaper storage.

The Oracle database SQL Access Advisor (documented in the Oracle Performance Tuning Guide) is useful for identifying tables that would benefit from partitioning.

Data Compression

While compression can of course reduce the amount of storage space required for a data warehouse environment, it can also improve data warehouse query performance. In order to query compressed data, the data must first be read from disk (or flash) and then the data must be uncompressed; often, the performance gains from reading fewer database blocks from disk far outweigh any CPU overhead for decompressing data, resulting in noticeably faster query performance. Since compression is transparent to applications, any data can be compressed without requiring any modifications to BI tools or applications.


Compression used in combination with partitioning is particularly powerful. Data in different partitions can be subject to different compression choices and some partitions might not be compressed at all. It is common to apply no compression to recent partitions that are subject to large numbers of inserts and updates (for example, data in the current quarter) and then to apply varying degrees of compression to older partitions depending on their age and access frequency.

Oracle offers three types of compression; basic compression, OLTP compression (a component of the Advanced Compression option), and Hybrid Columnar Compression (HCC). HCC is available on the Oracle Exadata Database Machine, Oracle SuperCluster, Sun ZFS Storage Appliance and the Pillar Axiom Storage System².

With basic compression, Oracle compresses data by eliminating duplicate values in a database block. Basic compression works only for direct path load operations (which are covered in the Data Ingestion section, below). If the data is modified using any kind of conventional DML operation (such as INSERT, UPDATE, DELETE or MERGE), the overall compression ratio will be degraded. Re-compression using partition movement is an effective way to restore maximum compression ratios if rows have been subject to significant updates.

With OLTP compression, just like basic compression, Oracle compresses data by eliminating duplicate values in a database block. But unlike standard compression OLTP compression allows data to remain compressed during all types of data manipulation operations, including conventional DML such as INSERT and UPDATE. Compression ratios for basic and OLTP are usually around 1.5X-3X.

² See the Oracle Licensing Guide for official licensing and packaging and for a detailed breakdown of all available features.



In Basic and OLTP compression, column data for a particular row is stored sequentially within a single database block. Having data from columns with different data types stored close together limits the amount of storage savings achievable with compression technology. Oracle's Hybrid Columnar Compression uses an alternative approach; data is stored in a "columnar" format where data is organized and stored by column. Storing column data together, with the same data type and similar characteristics, dramatically increases the storage savings achieved from compression. However, storing data in this manner can negatively impact database performance when application queries access more than one or two columns, perform even a modest number of updates, or insert small numbers of rows per transaction. Oracle's HCC technology uses a different approach; it utilizes a combination of both row and columnar methods for storing data. This hybrid approach achieves the compression benefits of columnar storage, while avoiding the performance shortfalls of a pure columnar format.

A logical construct called the compression unit is used to store a set of hybrid columnar-compressed rows. When data is loaded, column values for a set of rows are grouped together and compressed. After the column data for a set of rows has been compressed, it is stored in a compression unit. HCC requires data to be loaded and moved using direct path operations.

Data that is compressed more "aggressively" will generally require more CPU to uncompress and read, so HCC has a number of different compression levels to balance the achievable compression ratio against the CPU cost of querying the data. For example, if data is not likely to be queried often, it can be compressed maximally using "archive high". This level of compression is relatively costly to query (in terms of CPU) but very high compression ratios are possible. For data that is queried frequently, the less aggressive options of "query high" or "query low" will be more appropriate.

HCC has the potential to achieve very high compression ratios; 5X to 10X is common, but the ratio achievable is dependent on the content and row ordering of the target data set. To evaluate the potential benefit of all types of compression, the Oracle database includes a Compression Advisor to estimate the amount of storage that will be saved if compression is used on a particular data set. The advisor can be used to estimate HCC compression ratios even in environments that do not support HCC.

If HCC data is modified using conventional DML, the overall compression ratio will be degraded. Re-compression using partition movement is an effective way to restore maximum compression ratios if rows have been subject to significant updates.

Compressing data imposes a performance penalty during data load and data movement. Reading compressed data incurs a performance penalty that is offset by the need to read fewer blocks from storage. Different types of compression and different types of system will affect that balance. In Exadata, it is common for HCC to deliver high compression ratios *and* improved query performance.

In Oracle Database 12c, the Advanced Compression Option includes new features that simplify the management of compression. The Heat Map feature automatically tracks modification and query timestamps at the row and segment levels, providing detailed insights into how data is being accessed. Automatic Data Optimization (ADO) automatically moves and compresses data according to user-defined policies based on the information collected by Heat Map. For more information on ADO, see Reference 5 in *Further Reading*, below.

If compression is used, consider sorting data as it is loaded because this will achieve the best possible compression ratios. Attribute clustering (covered later) can be used to order data automatically when data is loaded and when tables and partitions are moved.

Data Ingestion - Efficient Loading and Transformation

There are many different ways of loading and transforming data in Oracle Database 12c. In most cases, there will be a requirement to move data quickly with minimal impact on overall system service levels. The most appropriate method of loading data will depend on how and when the data for loading is delivered, the volumes of data involved and the required latency between data delivery and data visibility. Oracle partitioning and the database's transactional model offer great flexibility in how a data can be loaded and made visible to applications.

Implementing Raw Data Reservoirs

Raw data reservoirs can be implemented in a variety of different ways. This paper will consider the most common scenarios: file staging areas and big data environments.

File Staging Areas on Conventional File Systems

Modern systems, such as the Oracle Exadata Database Machine, can ingest multiple terabytes of data per hour. To deliver data from a staging area at this rate requires a considerable IO infrastructure as well as a high performance channel between storage units and the database (such as Infiniband or multiple Ethernet or fibre channels). Volume management is usually required to distribute IO across multiple storage devices.

Staging areas for high performance systems should be designed and tested carefully to ensure that they will be capable of delivering data at the required rate. Network storage devices are likely to require multiple network paths to the database servers.

Oracle's File Systems

Oracle Automatic Storage Management Cluster File System (Oracle ACFS) is a scalable file system that extends Oracle Automatic Storage Management (Oracle ASM) functionality to support files maintained outside of an Oracle database.

The Oracle Database File System (DBFS) creates a standard file system interface on top of files and directories that are stored in Oracle database tables. A DBFS staging area can be used in a very similar way to a conventional file system staging areas, except that the files are stored transparently inside an Oracle database.

Both ACFS and DBFS leverage the performance and manageability benefits available with Oracle ASM. ACFS is supported on Exadata from release 12.1.0.2, making both ACFS and DBFS recommended for use in staging areas on Oracle Exadata deployments.

For more information on DBFS performance in Exadata, see Reference 8 in *Further Reading*, below.

Hadoop Distributed File Systems (HDFS)

Big data systems offer tremendous opportunities for storing and processing very large volumes of data. For the purposes of this paper, it will be assumed that a subset of the data residing in HDFS (or even *all* of the data) will be queried from the Oracle database or loaded into the Oracle database.

Oracle's Big Data Connectors and Oracle Big Data SQL enables data residing on HDFS to be read and queried by an Oracle database transparently using external tables (see below).

Maximum performance between HDFS and the Oracle database will depend on the performance of the network path between the Hadoop cluster and the Oracle database servers. Consider an Oracle Exadata Database Machine which can transfer data from an Oracle Big Data Appliance (BDA) at rates in excess of 10 terabytes per hour

because it is able to utilize a shared Infiniband fabric. For information on Oracle Big Data Connectors, see Reference 9 in *Further Reading*, below.

External Tables

External tables provide a consistent, optimized and flexible approach to batch and micro-batch loading from raw data reservoirs implemented using “conventional” file systems, DBFS and HDFS.

Oracle external tables offer a high performance and flexible approach for loading data from Raw Data Reservoirs. External tables can be used to read data from many different types of data sources, including the following:

- » Flat files on conventional file systems, Oracle ACFS and Oracle DBFS.
- » Big data sources using the *Oracle SQL Connector for Hadoop* for reading HDFS-based data.
- » Oracle Big Data Appliance data using *Oracle Big Data SQL* (for in-place querying and reading of HDFS-based data).

Once an external table is created it can be queried in the normal manner using standard SELECT statements. Crucially, this hides the underlying implementation of the Raw Data Reservoir from the Data Ingestion and Information Interpretation components. For example, it offers the potential to replace conventional file system staging areas with HDFS reservoirs without modifying data ingestion processes. External tables make it easy to scale up data load rates transparently because it is easy to use them with Oracle parallel execution (covered later).

The most common approach when loading data from an external table is to use a parallel CREATE TABLE AS SELECT (CTAS) statement or a parallel INSERT AS SELECT (IAS) statement, specifically in direct-path mode, into an existing database table. Multiple rows are usually processed for each CTAS or IAS operation, so external tables are most appropriate for processing and loading data in batch.

For more detail on loading data via external tables, see Reference 1 in *Further Reading*, below. More on the Oracle Big Data Connectors can be found in Reference 9 and more on Oracle Big Data SQL can be found in Reference 11.

Batch Loading

Direct path load is the key to high-performance data loads. Direct path loading is commonly used in combination with external tables.

Direct path loads deliver very high load performance. Formatted database blocks are written directly to the database, bypassing the standard SQL processing engine and the database buffer cache. Direct path load operations use standard data manipulation language (DML) and data dictionary language (DDL) syntax, but in most cases it needs to be enabled explicitly. Many ETL tools offer the option to choose a direct path load over conventional path.

Scaling up load rates is easy to achieve with direct path load and external tables because CTAS and direct path IAS can use Oracle parallel execution.

Direct path loads (and, in particular, parallel direct path loads) are most appropriate for loading large volumes of data in batch. If the size of the data batch is too small, the cost of direct path may be higher than the conventional path equivalent.

Direct path loads are required if there is a need to compress data on load with hybrid columnar and basic compression. It is also required if there is a need to cluster data on load using table attribute clustering (covered later).

Guidelines for Maximum Load Performance

Consider partition exchange loading for instantaneous data publishing.

One of the benefits of partitioning is the ability to load data very quickly and easily with minimal impact on the business users by using the exchange partition command. Direct path load used in combination with partition exchange yields the fastest possible data load performance. This technique is used where there is a need to sustain extremely high load rates.

The first step is to create and load a non-partitioned staging table that has columns matching the partitioned table to be loaded. The exchange partition command allows the data in the staging table to be swapped into a partition in the main partitioned table. The exchange command does not physically move data; instead it updates the data dictionary to exchange a pointer from the partition to the table and vice versa. What's more, indexes can be included in the exchange if indexes on the staging table match local partitioned indexes on the partitioned table. Because there is no physical movement of data, an exchange is a very fast operation: it is far less likely to impact performance than any traditional data-movement approaches such as INSERT.

For more details on partition exchange loading, see Reference 1 in *Further Reading*, below.

Real-Time Continuous Load and Micro-Batch Loading

Oracle's read consistency model and absolutely non-blocking conventional DML are key for continuous data loads.

The conventional path load mechanism refers to the use of standard data manipulation language (DML) commands such as INSERT, UPDATE and MERGE to deliver data into the database. Most ETL tools offer conventional path loading by default. Conventional path loading can be used to load data in micro-batches (via external tables, for example) but it is also suited to situations where there is a requirement to process the continuous arrival of data, particularly where new data must be visible to queries immediately. In this case, continuously applying updates to a database becomes more appropriate than the more batch-oriented, external table approach. Oracle GoldenGate is a good example of a product that can use conventional path loads to stream updates into a database for real-time data integration.

Conventional INSERT and MERGE statements write data into the database via the buffer cache. Loading this way will generally consume more system resources than the direct path load approach. Conventional path inserts may also require more input/output operations per second (IOPS) against the storage subsystem than the direct path approach, but increasing CPU capacities and the wide availability of high performance storage subsystems have made high-rate, conventional path loads into "live", queryable, indexed tables much more viable than ever before. Oracle's read-consistent transactional model allows data to be loaded continuously without blocking queries or affecting the consistency of query results: an EDW with near real-time data load and query is an appropriate use-case for the Oracle database.

Multiple loading processes or threads will be required to scale up the load rate of a conventional path load. In this case, the loading application will be responsible for the creation and management of parallel loading processes or threads.

Optimizing Transformation

Unless you are processing trickle-feeds, any kind of set-based data processing is the fastest method of data processing.

Load processes may include data transformations, or transformations may follow data loading steps. In many cases, transformation tools such as Oracle Data Integrator (ODI) will be used to achieve this, but it is worth bearing in mind some general principles for processing large volumes of data in a timely fashion.

Transforming data using bulk CREATE TABLE AS SELECT (CTAS), MERGE and INSERT AS SELECT (IAS) operations is very efficient. It is very easy to scale-out these operations using parallel query, DDL and DML should there be a need to shorten elapsed times. Bulk operations like this are generally referred to as “set-based”. Set-based CTAS and IAS approaches are recommended for data movement between the different layers of the EDW, especially if the amount of transformation required is minimal.

The alternative to set-based processing is row-by-row processing. This usually involves running a loop for a very large number of iterations, with each adding or changing a small amount of data using conventional path DML. This approach is often considered more intuitive than set-based processing when applying business logic but it is almost always less efficient. This in itself may not be a problem, but scaling out a task that incorporates a single-threaded loop is often difficult. If row-by-row processing is proposed, a method for scaling out across multiple streams of processing needs to be established early in the design process. In this case, Oracle Partitioning is very useful for subdividing datasets into separate units of work to be dealt with by multiple processing threads.

Information Interpretation - Optimizing Query Performance

Oracle Database 12c has many optimizations designed to enhance query performance. It is important to remember that there is no restriction placed on which optimization can be used with which schema or layer (assuming that the layer is within an Oracle RDBMS). The characteristics of the different layers will have some bearing on which optimizations are the most appropriate: service levels, data access profiles and data volumes for the Foundation Data Layer are likely to be quite different to the Access and Performance Layer. This paper will make recommendations regarding where each optimization is likely to yield the maximum benefit.

The architecture of your EDW hardware platform will influence which optimizations are most appropriate. For example, platforms with large amounts of DRAM are in a position to take advantage of In-Memory column stores. Exadata scan performance may reduce the need to choose additional optimizations (for example, choosing raw scan performance instead of star schema optimization using bitmap indexes).

Optimizations are not mutually exclusive: they can be used together. They are transparent to the application layer so there is no requirement to use them all or to make the “right choice first time”. Start with partitioning and add further optimizations as the need arises.

Parallelism for Query Performance

Parallel execution is a key feature for large-scale data warehousing and is always recommended.

The key to getting maximum performance from a data warehouse is to utilize all available hardware resources effectively: multiple CPUs, multiple IO channels, multiple storage arrays and disk drives, and large volumes of memory. Parallel execution is a key feature for enabling even individual SQL statements to use large amounts of

machine resources when it is deemed appropriate, and it can be utilized irrespective of which type of data model is used.

In an EDW, parallelism is particularly useful in the Foundation Data Layer because large volumes of data will need to be moved, copied and transformed. For the Access and Performance layer, parallelism can be used throughout or in a more tactical manner to reduce the response times of particular queries (Auto Degree of Parallelism should be used to invoke parallelism when it is appropriate).

All resource intensive operations benefit from parallel execution, including:

- » Complex queries that access large amounts of data.
- » Loading and manipulating large volumes of data.
- » Building indexes on large tables.
- » Gathering Optimizer statistics.
- » Backing up and restoring databases.

SQL parallel execution in the Oracle Database is based on the principles of a coordinator (often called the Query Coordinator or Parallel Execution Coordinator) and parallel servers. The parallel execution coordinator is the session that initiates the parallel SQL statement. The parallel servers are the individual sessions that perform work in parallel.

Figure 7 represents a parallel query selecting and sorting rows from a SALES table. A set of four parallel execution servers scan the data in the table and any rows that are not filtered out by query predicates are passed on to a second set of parallel execution servers which sort the rows before passing the final result on to the parallel execution coordinator. The whole process operates transparently; there is no requirement for the query issuer to be aware that this process is taking place. What's more, the parallel execution servers can be distributed transparently across multiple servers in a cluster: a single, parallel SQL statement can make use of all available resources across an entire RAC cluster if necessary.

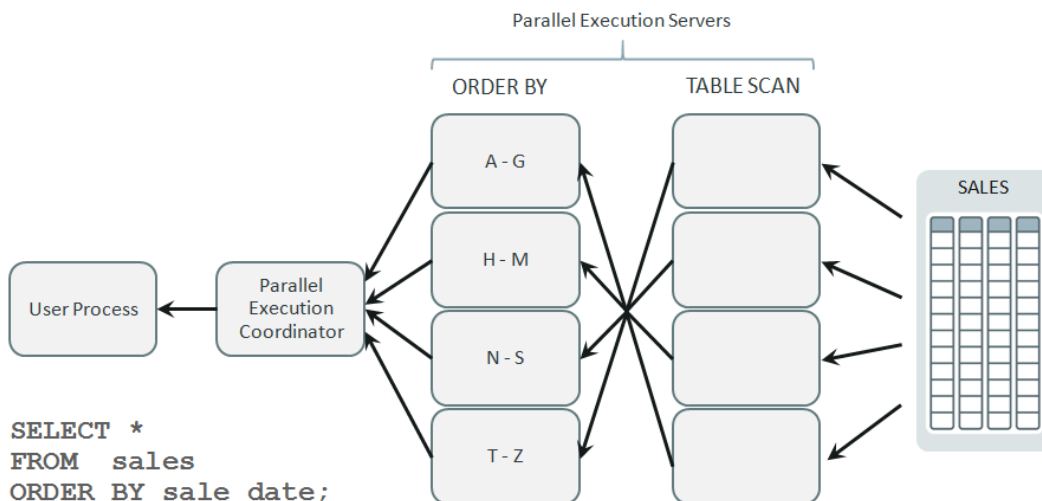


Figure 7: A representation of parallel query execution.

By default, the Oracle Database is configured to support parallel execution out-of-the-box. In addition to this, the Oracle Database 12c feature Automatic Degree of Parallelism (Auto DOP) can be used to control how and when parallelism is used for each individual SQL statement.

Parallel execution provides a very powerful and scalable framework to speed up SQL operations but if it is not used in a controlled way, it can consume all available machine resources or adversely affect the performance of competing workloads. For this reason, parallelism should be used in combination with workload management so that these types of issues can be avoided. This is covered in a later section.

For more information on parallel execution and Auto DOP, see Reference 7 in *Further Reading*, below.

Partitioning for Performance

Partitioning is the key to high performance and scalability in data warehousing environments.

In data warehousing environments, where querying large datasets is common, partitioning will offer significant performance benefits. For example, assume that business users predominately accesses sales data on a quarterly basis, e.g. total sales per Q3. In this case, range partitioning the sales table by quarter will ensure that the data is accessed in a very efficient manner, as only one partition will need to be scanned to resolve a typical quarterly query. Without partitioning, it might be necessary to scan the entire table. The ability to avoid scanning irrelevant partitions is known as partition pruning.

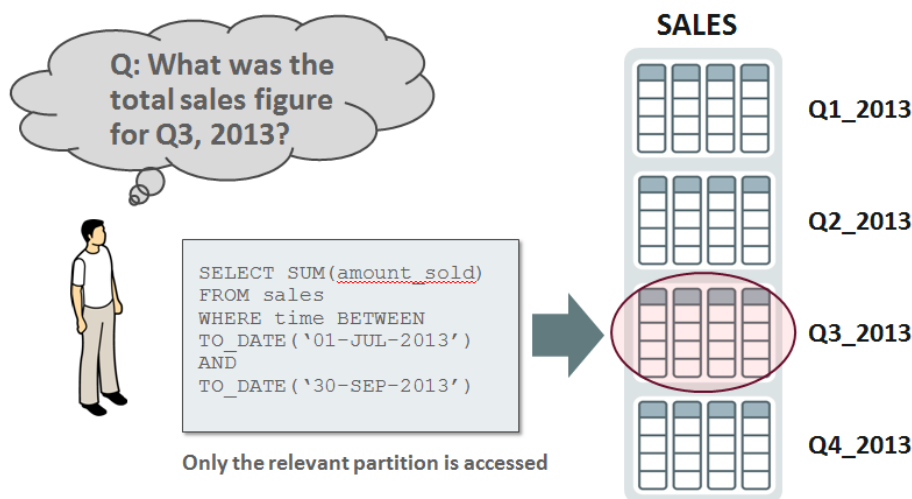


Figure 8: Partition pruning - reducing the amount of data that needs to be scanned.

In data warehousing environments, time-based range partitioning is frequently used. Time-based (and numeric) range partitions can be created automatically, as needed using an Oracle Database feature called interval partitioning.

It is very common to use range (or interval) partitioning to divide a table into date ranges based on a date column, but it is possible to partition using a variety of different column data types.

As mentioned above, it is possible to create composite partitioned objects by dividing partitions up into sub-partitions. Many combinations of partitions and sub-partition types are available, but common combinations are Range-Hash, Range-List, Interval-Hash and Interval-List partitioning.

Range-List or Interval-List partitioning is commonly used for data warehouse fact tables that are frequently filtered by a common pair of dimensions. In the following example, Range-List partitioning is used to divide a SALES table up into time-based partitions (quarters) and region-based sub-partitions (groups of US states). Queries that filter on quarter and region only need to scan a single sub-partition instead of (potentially) the entire SALES table:



Figure 9: Composite partitioned table (range-list): scanning SALES for “Q1 West” (highlighted).

Partitions can be divided up into sub-partitions based on a hash value derived from column values (using a linear hashing algorithm). Sub-partitioning by hash is used predominately for performance reasons on very large tables (typically in the region of hundreds of gigabytes or more in size), often to enable partition-wise joins. Partition-wise joins reduce query response time by minimizing the amount of data exchanged among parallel execution servers because individual parallel execution servers can process the join between matching sub-partitions:

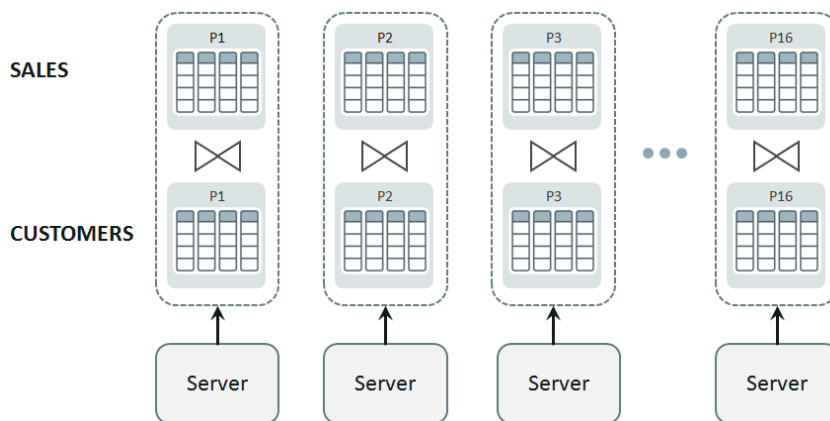



Figure 10: Reducing the communication between parallel execution servers with Partition-wise joins.

This optimization happens automatically. It significantly reduces response times and improves both CPU and memory resource usage. In a RAC data warehouse, this can reduce response times by limiting amount of data flowing over the RAC interconnect when queries are distributed across multiple nodes in the cluster. This is the key to achieving good scalability for massive join operations. It is also possible to derive some performance benefit if only one of the joined tables is sub-partitioned: this is known as a partial partition-wise join.



For partition-wise joins, the number of partitions determines the maximum degree of parallelism that can be used in the join, so a sufficient number of hash partitions should be used to make full use of system resources when required (8, 16 or 32 sub-partitions is common).

For more information about Oracle Partitioning, see Reference 2 in *Further Reading*, below.

Zone Maps and Attribute Clustering

Consider using zone maps and attribute clustering for optimizing star and snowflake schemas.

Zone maps and attribute clustering are features that can be used to reduce IO in database applications that would otherwise scan large volumes of data. These two features can be used individually, but they are designed to work together. Zone maps are similar in principle to Exadata Storage Indexes but they have additional benefits that are particularly useful in data warehousing environments.

A zone map is a data structure that conceptually divides a table up into contiguous regions of blocks called zones. For each zone, the zone map records the minimum and maximum values for specified columns. Queries that filter on zone map columns have the potential to be optimized; it is possible to avoid scanning zones that contain ranges of column values that are known to be outside the range defined by the queries predicates. Zone maps therefore enable a query to prune zones (and potentially full partitions) based on column predicates and significantly reduce the amount of data that need to be scanned.

Zone maps are particularly relevant in data warehousing environments because it is possible to specify zone map columns from both fact and dimension tables. This makes it possible to prune fact table scans based on queries that filter by dimensional hierarchies. For example, consider a SALES fact table that is filtered by “region name” and “sub region name” using a join with the REGIONS dimension table. A zone map can be created on the SALES table that specifies the “region name” and “sub region name” columns in the REGIONS dimension table. Zones can be pruned from the SALES table scan if the query filters rows based on “region name” and “sub region name” or “region name” alone.

Attribute clustering is a table-level directive that is used to “order”³ rows so that they are stored in close physical proximity based on column values. Row ordering can be based on column values in the attribute clustered table alone, but also on column values in tables that join with the attribute clustered table. This is particularly relevant in a data warehousing environment because it becomes possible to use zone maps to optimize fact/dimension table joins. Ordering fact table rows is used to minimize the number of zones that are likely to contain a particular fact table column value or a corresponding dimension attribute value. Returning to the example above, attribute clustering can be used on the SALES table to keep all rows that correspond to the region name “TEXAS” together, within a minimum number of zones. In this way, any query that filters on the “TEXAS” will need to scan a minimum number of zones to find the matching “TEXAS” rows.

Zone maps can be used instead of indexes on fact tables and they offer an alternative to star schema optimization with bitmap indexes (discussed below). They can be thought of as coarse index structures, requiring significantly less storage than bitmap or b-tree indexes. However, they differ from indexes in how they are synchronised with changes made to table data. Indexes are always synchronized with the data that is held in tables, but zone maps are similar to materialized views (discussed below); they must be refreshed to be synchronized with table data. This can be performed at commit time for direct path operations, so zone maps are most suited to environments where the underlying table data is created using direct path operations.

³ “Clustering” can be implemented as classical linear ordering (ORDER BY) or as multi-dimensional ordering similar to Z-Order.

Materialized Views

Consider materialized views for expensive repetitive queries to optimize runtime and resource consumption.

Data summarization and aggregation is an important feature of many data warehouses. Queries will usually analyze a subset or aggregation of the detailed data, so a mechanism to pre-summarize and pre-aggregate data for direct use by queries offers the potential to improve query performance significantly. Queries that access summary data rather than detail data will use fewer system resources, so there is the potential to improve overall system scalability too. Ideally summaries and aggregates should be transparent to the application layer so that they can be optimized and evolved over time without having to make any changes to the application itself.

Summaries or aggregates that are referred to in data warehousing literature are created in the Oracle Database using a schema object called a Materialized View (MV). A feature called query rewrite automatically rewrites SQL queries to access summary tables where appropriate so that materialized views remain transparent to the application.

Materialized views are commonly found in the access and performance layer. Because MVs have to be refreshed and synchronized with base table data, they are most appropriate for environments or schemas where data is not highly volatile or where data is continuously appended. MVs are also appropriate in EDWs that refresh Access and Performance Layers using a complete schema rebuild.

Materialized view design is an activity that will often occur during the logical to physical mapping process. Also, it is beneficial to continuously evaluate the effectiveness of materialized views because business requirements are likely to evolve over time, changing the type of queries that the data warehouse will need to support. Oracle Database 12c provides us with a mechanism to simplify this process; it is called Summary Management. The advisors and components of summary management are described in the Oracle Database Data Warehousing Guide for 12c.

On-line Analytical Processing (OLAP)

Consider leveraging OLAP for star schemas to optimize aggregations and summary management even further.

OLAP is a summary management solution for SQL-based business intelligence applications. The OLAP Option runs within the Oracle Database instance; there are no separate servers, users or database files to manage.

OLAP cubes are multi-dimensional storage structures that are used to store arrays of analytical data such as dimensions and measures. When used as cube organized materialized views, they enable SQL-based applications to access summary data managed within the cube using the query rewrite capabilities of the Oracle Database. Oracle aggregates and manages summary data within the cube and applications continue to query the detail relational tables using SQL. When a query requires summary data, Oracle automatically rewrites the query to the cube. The rewrite is completely transparent to the application.

OLAP cubes are commonly used in the access and performance layer. For more information on OLAP, see Reference 12 in *Further Reading*, below.

Oracle Database In-Memory Option

You should consider leveraging an In-Memory column store (IM column store) for optimal performance in the access and performance layer.

The Oracle Database In-Memory Option was introduced in Oracle Database 12c Release 12.1.0.2. It enables tables or partitions to be stored in memory in a columnar format. The columnar format enables scans to perform much faster than the traditional on-disk formats for analytical style queries.

The column store has been seamlessly integrated into the database, so applications can use this feature transparently without any changes. A DBA simply has to allocate memory to IM column store. The optimizer is aware of IM column store, so whenever a query accesses objects that reside in the column store and would benefit from its columnar format, they are sent there directly.

The In-Memory Transaction Manager (IM transaction manager) keeps the columnar data consistent with the buffer cache so applications remain unaffected by the addition of the IM column store.

The IM column store is particularly suited for use in the access and performance layer:

- » Compression algorithms are used to reduce the size of the data resident in memory. On many modern hardware platforms, there is now the potential to store a large proportion of the access and performance layer in memory (or in some cases, the entire layer).
- » Many analytical queries scan large datasets and are limited by the read performance of the IO subsystem. These types of queries are likely to benefit the most from IM column store.
- » There are optimizations built into the IM column store for optimizing in-memory scanning and join performance. These optimizations are well suited to analytical fact-dimension table joins.
- » IM column store is transparent to applications using the database. It can be used tactically to improve response times in critical areas without changing the application.

The IM column store can be used to simplify the physical implementation of the data warehouse because it commonly reduces the need to build and maintain materialized views or use indexes for star schema optimization; it also makes the data processing more forgiving for suboptimal schema designs. This simplification reduces resource overheads associated with index maintenance and summary management. Fewer indexes and summaries will leave more space for storing data.

In general, hardware platforms with tens of gigabytes of free DRAM are required to usefully impact query performance.


More information on the Oracle Database In-Memory Option can be found in Reference 12 (see *Further Reading*, below)

Star Query Optimization

Consider Oracle's star query transformation to reduce I/O for star schema queries.

Oracle's star query transformation is an optimization technique that reduces the amount of disk reads required to access data joined in a typical fact-dimension table join. This optimization is most appropriate for star schemas so it is usually used in the Access and Performance Layer.

Multiple fact and dimension tables can be included in the query. The optimization requires the use of bitmap indexes to reduce the number of database blocks that need to be read to satisfy the join. It operates transparently so there is no need to modify queries in order to use it and the optimizer will use it when it is appropriate.



Star query transformation is particularly appropriate for use in non-Exadata environment. This is especially true in environments with less capable IO subsystems that will benefit from reducing the amount of fact data that needs to be scanned.

In systems with large amounts of DRAM it may be appropriate to rely on the IM column store instead of star query optimization. Zone maps are another alternative to star query optimization. All of these optimizations can be used together, but IM column stores and zone maps can be used to obviate the need to create bitmap indexes.

System Management

EDWs must support complex and diverse workloads, making it necessary to monitor and balance the competing needs for adequate response times and throughput against the allocation and overall availability of system resources. Oracle Database 12c has features to make this possible.

Workload Management

RAC services, Auto DOP, and Resource Management are fundamental key components for every mixed workload environment.

Workload Management Requirements

EDWs must support a diverse range of workloads, applications and classes of user and each will have its own service level agreement (SLA). There will be a need to maximize the value of hardware investment by making full use of all system resources, so it is natural to operate EDW infrastructures at high levels of utilization. High utilization will increase the likelihood that there will be contention for system resources which, in turn, will mean that there will be a requirement to manage and control those system resources; otherwise service levels will be at risk.

Some classes of workload will need to be allocated fixed or maximum resource limits, often to achieve consistent or fixed response times. For example, an end-of-day batch run will need to complete within an allocated batch window no matter what else may be running on the system. BI users often value response time consistency rather than sporadic “sometimes-fast”, “sometimes-slow” performance. For these types of workloads, resource isolation or segregation may be required to ensure that appropriate resources are available.

Some classes of workload will aim to minimize response times at the expense of system resources. For example, critical reporting runs, analytical services or bulk updates may be granted the ability to consume *all* available machine resources (or at least allowed to consume up to a pre-defined maximum). Using workload management to enable this type of behavior is very useful for maximizing machine utilization to achieve maximum throughput and minimum response times. In these types of environment, workload management can control this behavior, and it can reserve CPU, IO and parallel execution servers for other critical activities. Automated mechanisms for controlling parallel execution are useful because they can be used to enhance or reduce the degree of parallelism available to these types of workload.

An EDW workload profile will change continuously. Many EDWs will change from being query-oriented to batch-oriented throughout a 24-hour period and business requirements are subject to continuous change. For these reasons, the features that Oracle Database 12c provides for workload management are dynamic in nature: they can be relied upon to deliver a flexible solution that is able to adapt to changing processing needs and can evolve with requirements of the business. This flexibility can be used to start simply, followed by monitoring and control to refine the solution. Workload management is inherently an iterative and on-going process of evolution and refinement.

Classifying Workloads

For the purposes of managing system resources, the critical first step is to identify and classify the different types of workload you intend to control. This could be as basic as “ad hoc query”, “end of day batch” and “summary management”. A common approach is to start simply; following up with monitoring and refinement. If the EDW explicitly supports different classes of users and applications with particular service levels, this can be used as a starting point for incorporation into the workload management plan. Consideration will need to be given to how the service level requirements change throughout the day, week, quarter and so on.

For a full discussion of workload management in a data warehousing context, including much more detail on this crucial first step, see Reference 16.

The Components of Workload Management

The components for managing workloads in a data warehousing environment are:

- » Real application clusters (RAC).
 - » RAC services are used to map specific workloads to specific database servers.
- » Database Resource Manager (DBRM).
 - » DMRM controls many aspects of workload management including CPU and parallelism.
- » IO Resource Manager (IORM).
 - » Available on the Oracle Exadata Database Machine for managing the allocation of storage IO to particular workloads.
- » Enterprise Manager Cloud Control (EMCC).
 - » For configuring and monitoring workload management.

These components are used in an iterative manner, represented here:

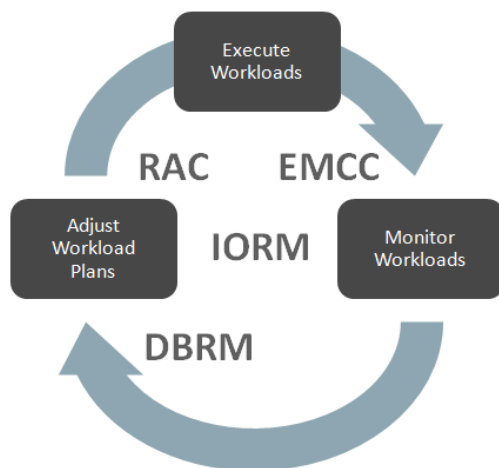


Figure 11: Workload management is an iterative process.

Managing CPU, Network and IO

The Oracle Database Resource Manager (DBRM) enables the prioritization of work within an Oracle database. Its use is highly recommended in data warehousing environments. DBRM will control the allocation of CPU to individual workloads once a system becomes CPU-limited (on Exadata and non-Exadata environments). On Oracle Exadata Database Machine deployments, DBRM can be used to manage the allocation of CPU, network and IO.

DBRM uses the concepts of “consumer groups” and “resource plan” to manage individual workloads. Consumer groups are mapped to specific workloads using database session characteristics such as usernames and client types. Resource plans specify how resources are to be distributed among various consumer groups. The resources include percentages of CPU time, maximum idle and wait times and the amount of resource available for individual queries.

DBRM offers the potential to configure and use every database server in a RAC cluster in an identical manner. Instead of using database services to direct different workloads to different groups of database servers, it is possible instead to map all workloads to all database servers and use DBRM to control the amount of resource each workload has available to it.

More information on DBRM can be found in Reference 14 (see *Further Reading*, below).

Managing Parallelism

The number of parallel execution servers associated with a single operation is known as the degree of parallelism (DOP). Oracle's parallel execution framework makes it possible to explicitly chose - or even enforce - a specific DOP. Alternatively, Oracle can be used to control it automatically. Managing parallelism is the primary consideration when managing machine resources in most EDWs. Parallel execution can use significant proportions of machine resource even for a single query, but this is often undesirable in mixed workload environments. For this reason, Oracle Database 12c has the ability to use parallel execution in a controlled manner by constraining DOP as and when required.


In general, well defined workloads such as ETL and batch will fix a particular DOP. Ad-hoc query environments will benefit from using a more dynamic option, namely Oracle's Auto DOP functionality. Oracle automatically controls the DOP of any SQL statement based on its resource requirements (a.k.a. cost of a statement); furthermore, statements will be queued automatically when a DBA-configured system threshold is reached. Additionally, statements will be processed fully in memory when appropriate (note this functionality will leverage the normal buffer cache and is unrelated to the IM column store).

DBRM provides the ability to manage parallel execution in a very flexible and effective manner. For this reason, DBRM is highly recommended for managing parallel execution in EDW environments, irrespective of whether a particular system is CPU or IO-limited. It is possible to control parallel execution in different ways for each consumer group. DBRM is the ultimate deciding factor in determining the maximum degree of parallelism, and no user in a consumer group (using a specific resource plan) will ever be able to run with a higher DOP than the resource group's maximum.

For more information on how parallelism can be used, see Reference 7 in *Further Reading*, below.

Managing RAC Environments

In RAC environments, the option to map different workloads to different servers exists. The preferred method for controlling where parallel execution can occur on a RAC deployment is to use services. With services, connections from a particular application client or group of clients can be directed to specific database nodes. In addition, the parallel server processes for those connections will be constrained to the nodes mapped to the service. This delivers



a good degree of control over the environment and because it is a flexible solution, it is possible to change the service-to-node mappings dynamically, providing the necessary agility for changing workload requirements.

In data warehousing environments a good approach is to manage resources “horizontally” using DBRM rather than “vertically” using services. For example, allow all workloads to use all nodes in the RAC cluster and use DBRM to manage the proportion of IO, CPU and parallel execution servers uniformly across the cluster. In this way, all cluster members are symmetrical. DBRM resource allocations made to consumer groups are effective almost instantaneously, whereas database connections may take longer to migrate between individual servers when service configurations are changed (the time will depend in the type of application used).

For more details on how to manage complex workloads leveraging parallel execution, see Reference 16 in *Further Reading*, below.

Workload Monitoring

Leverage Oracle’s comprehensive database monitoring and workload repository and use Enterprise Manager Cloud Control for visualization.

As discussed above, workload management is an iterative process, so it is important to be able to measure the effects of change. It is also important to understand the causes of change and whether there are improvements or regressions after resource management plans have been changed. The primary mechanism for achieving this in Oracle Database 12c is the Automatic Workload Repository (AWR). This repository stores key database metrics that are captured, by default, on an hourly basis. This is useful for establishing baselines in expected throughput and performance. The AWR provides a continuous and detailed view of the system as a whole.

For more day-to-day workload and performance management, graphical diagnostic and performance monitoring screens are available in Oracle Enterprise Manager Cloud Control 12c (EMCC) or Oracle Enterprise Manager Express (Oracle EM Express). EMCC and EM Express are useful for providing a real time view of system activity, but they are also capable of presenting historical information in AWR and via system activity charts.

EMCC includes a comprehensive interface for controlling and monitoring DBRM.

For more information on workload monitoring, see References 17 and 18 in *Further Reading*, below.

Optimizer Statistics Management

SQL statement optimization relies on good statistics – so provide those.

For good query performance, the Oracle query optimizer must have accurate information on the objects that it needs to access. This information is stored in the data dictionary and must be gathered and maintained so that it is as up-to-date as possible. Gathering statistics requires a proportion of the data in the database to be scanned and sampled to ascertain certain key statistics such as the number of rows in each table and the cardinality of table column values. Gathering these statistics has the potential to be a costly operation in terms of system resource because of the large volumes of data stored in a typical EDW.

For this reason, Oracle has optimizations built into the statistics gathering process to maintain the accuracy of statistics even though only a small sample of data is read. Also, Oracle has mechanisms that leverage partitioning to mitigate the cost of maintaining statistics on very large tables by enabling you to gather statistics incrementally (on a partition-by-partition basis). Global statistics are generated by aggregating statistics from each partition. When a new partition is added to the table you only need to gather statistics for the new partition. The process of maintaining good statistics is covered in detail in Reference 15 (see *Further Reading*, below).

Conclusion

Oracle being optimally suited and built for data warehousing provides you with many features that will enable you to deliver against continuously evolving functional and non-functional requirements. Following the principles outlined in this paper, you ensure that your data warehouse will continue to operate successfully with increasingly challenging service level agreements and functional requirements.

In summary, here are the fundamental principles:

- » Hardware.
 - » Consider using Oracle Exadata Database Machine. Alternatively, design for high IO throughput and consider component failure scenarios with respect to availability and performance service levels.
 - » Consider the implications of future scale-out.
- » Managing and storing large volumes of data effectively.
 - » Use partitioning.
 - » Use compression.
- » Loading and transforming data efficiently.
 - » Use external tables for set-based data loading.
 - » Use set-based data processing whenever possible.
 - » Design and build in the ability to scale out using parallelism.
 - » Consider what is best in your situation: balancing the needs of performance, data visibility and service levels, data arrival patterns and overall ease-of-use.
- » Optimizing query performance.
 - » Use partitioning.
 - » Use parallelism, especially with 3NF schemas.
 - » Star Schemas should use physical optimizations such as star schema optimization, OLAP, In-Memory column stores and zone maps. Choose the most appropriate features for your hardware architecture and for your service levels.
- » System management.
 - » Use DBRM to manage parallelism and monitor using EMCC or Oracle EM Express.
 - » Make optimizer statistics management a priority.

By following these principles, you will have an EDW that will be faster, easy to manage and highly scalable. It will have a lower TCO, happier users and it will continue to scale and evolve along with your business.

Further Reading

The following Oracle white papers and data sheets are referenced in the text:

1. [Information Management and Big Data - A Reference Architecture](#)
2. [A Technical Overview of the Oracle Exadata Database Machine and Exadata Storage Server](#)
3. [A Technical Overview of New Features for Automatic Storage Management in Oracle Database 12c](#)
4. [Partitioning with Oracle Database 12c](#)
5. [Automatic Data Optimization with Oracle Database 12c](#)
6. [Performant and Scalable Data Loading with Oracle Database 12c](#)
7. [Oracle Database Parallel Execution Fundamentals](#)
8. [DBFS use case performance on Exadata configurations](#)
9. [High Performance Connectors for Load and Access of Data from Hadoop to Oracle Database](#)
10. [Oracle Big Data Connectors \(Data Sheet\)](#)
11. [Oracle Big Data SQL: One fast query, on all your data.](#)
12. [Oracle Database In-Memory](#)
13. [On-line Analytical Processing with Oracle Database 12c](#)
14. [Effective Resource Management Using Oracle Database Resource Manager](#)
15. [Best Practices for Gathering Optimizer Statistics with Oracle Database 12c](#)
16. [Parallel Execution and Workload Management for an Operational Data Warehouse](#)
17. [Manageability with Oracle Database 12c](#)
18. [Best Practices for Workload Management of a Data Warehouse on the Oracle Exadata Database Machine](#)

You will find additional information on the “Oracle Data Warehousing Best Practice” web page hosted on the Oracle Technology Network:

<http://www.oracle.com/technetwork/database/bi-datawarehousing/dbbi-tech-info-best-prac-092320.html>



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



Hardware and Software, Engineered to Work Together

Copyright © 2014, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only, and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document, and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group. 0215