Oracle® Database

2 Day + Data Warehousing Concepts and Design Guide

12*c* Release 1 (12.1)

E17751-05

July 2012

Beta Draft

Oracle Database 2 Day + Data Warehousing Concepts and Design Guide, 12*c* Release 1 (12.1)

E17751-05

Primary Author: Paul Lane

Contributor: The Oracle Database 12*c* documentation is dedicated to Mark Townsend, who was an inspiration to all who worked on this release.

Contributor: Hermann Baer, Charlie Berger, John Haydu, George Lumpkin, Valarie Moore, Ananth Raghavan, Jack Raitto, Sankar Subramanian, Margaret Taft, Mark Townsend, Mark Van de Wiel, Andy Witkowski

Contents

3 Data Warehousing Logical Design

4 Data Warehousing Physical Design

Preface

This preface contains these topics:

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

Audience

Oracle Database 2 Day + Data Warehousing Concepts and Design Guide is for anyone who wants an introduction to common ideas used in an Oracle Database data warehousing environment.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documents

For more information, see these Oracle resources:

- *Oracle Database Data Warehousing Guide*
- *Oracle Database Administrator's Guide*
- *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator*
- *Oracle Data Mining Concepts*
- *Oracle OLAP User's Guide*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Part I

Building Your Data Warehouse

Part I discusses building a data warehouse and includes:

- Chapter 1, "Introduction to Data Warehousing Concepts"

1 Introduction to Data Warehousing Concepts

As the person responsible for administering, designing, and implementing a data warehouse, you also oversee the overall operation of Oracle data warehousing and maintenance of its efficient performance within your organization.

This section contains the following topics:

- About This Guide
- What Is a Data Warehouse?
- Concepts Illustrated in This Guide

About This Guide

Oracle Database 2 Day + Data Warehousing Concepts and Design Guide teaches you how to perform common day-to-day tasks necessary to implement and administer a data warehouse. The goal of this guide is to introduce you to the data warehousing solutions available in Oracle Database. In addition, this guide introduces you to a broad range of the data warehousing solutions available from Oracle.

What This Guide Is Not

Oracle Database 2 Day + Data Warehousing Concepts and Design Guide is not an exhaustive discussion of implementing a data warehouse on Oracle. The objective for this guide is to describe big picture issues in data warehousing, not step-by-step tasks. As the Guide is concept-focused, it does not present code examples.

For complete conceptual information about these features and detailed instructions for using them, see the appropriate Oracle documentation as follows:

- *Oracle Database Data Warehousing Guide*
- *Oracle Database Administrator's Guide* for a discussion of administrative tasks
- *Oracle Data Mining Concepts* for a discussion of data mining
- *Oracle OLAP User's Guide* for a discussion of OLAP
- *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator* for a discussion of ODI

What Is a Data Warehouse?

A **data warehouse** is a database designed to enable business intelligence activities: it exists to help users understand and enhance their organization's performance. Given that every organization faces pressure to improve performance, data warehouses are found everywhere. They can be seen in every domain from business to government to nonprofit. Just as they span the spectrum of domains, data warehouses range immensely in their data size. You can find giant multinational firms storing hundreds of terabytes of data, and you can also see local school systems with just gigabytes. Similarly, the user base for data warehouses varies tremendously in size. A large enterprise may have tens of thousands of users, while a small firm has a handful.

The modern world of the internet means that data is shared ever more widely, so enterprises today will often give carefully controlled data warehouse access to customers, partners and suppliers. Therefore the users of a data warehouse may be internal or external, local or on the other side of the globe, and far more numerous than the enterprise's own employees. It is a certainty that data warehouse audiences will grow in size.

To achieve their goal of enhanced business intelligence, the data warehouse works with data collected from multiple sources. The source data may come from internally developed systems, purchased applications, third-party data syndicators and other sources. It may involve transactions, production, marketing, human resources and more. In today's world of big data, the data may be many billions of individual clicks on web sites or the massive data streams from sensors built into complex machinery.

Data warehouses are distinct from online transaction processing (OLTP) systems. With a data warehouse you separate analysis workload from transaction workload. Thus data warehouses are very much read-oriented systems. They have a far higher amount of data reading versus writing and updating. This enables far better analytical performance and avoids impacting your transaction systems. A data warehouse system can be optimized to consolidate data from many sources to achieve a key goal: it becomes your organization's "single source of truth". There is great value in having a consistent source of data that all users can look to; it prevents many disputes and enhances decision-making efficiency.

A data warehouse usually stores many months or years of data to support historical analysis. The data in a data warehouse is typically loaded through an extraction, transformation, and loading (ETL) process from multiple data sources. Modern data warehouses are moving toward an extract, load, transformation (ELT) architecture in which all or most data transformation is performed on the database that hosts the data warehouse. It is important to note that defining the ETL process is a very large part of the design effort of a data warehouse. Similarly, the speed and reliability of ETL operations are the foundation of the data warehouse once it is up and running.

Users of the data warehouse perform data analyses that are often time-related. Examples include consolidation of last year's sales figures, inventory analysis, and profit by product and by customer. But time-focused or not, users want to "slice and dice" their data however they see fit and a well-designed data warehouse will be flexible enough to meet those demands. Users will sometimes need highly aggregated data, and other times they will need to drill down to details. More sophisticated analyses include trend analyses and data mining, which use existing data to forecast trends or predict futures. The data warehouse acts as the underlying engine used by middleware business intelligence environments that serve reports, dashboards and other interfaces to end users.

Although the discussion above has focused on the term "data warehouse", there are two other important terms that need to be mentioned. These are the data mart and the operation data store (ODS).

A data mart serves the same role as a data warehouse, but it is intentionally limited in scope. It may serve one particular department or line of business. The advantage of a data mart versus a data warehouse is that it can be created much faster due to its limited coverage. However, data marts also create problems with inconsistency. It takes tight discipline to keep data and calculation definitions consistent across data mart. This problem has been widely recognized, so data marts exist in two styles. Independent data marts are those which are fed directly from source data. They can turn into islands of inconsistent information. Dependent data marts are fed from an existing data warehouse. Dependent data marts can avoid the problems of inconsistency, but they require that an enterprise-level data warehouse already exist.

Operational data stores exist to support daily operations. The ODS data is cleaned and validated, but it is not historically deep: it may be just the data for the current day. Rather than support the historically rich queries that a data warehouse can handle, the ODS gives data warehouses a place to get access to the most current data, which has not yet been loaded into the data warehouse. The ODS may also be used as a source to load the data warehouse. As data warehousing loading techniques have become more advanced, data warehouses may have less need for ODS as a source for loading data. Instead, constant trickle-feed systems can load the data warehouse in near real time.

The Key Characteristics of a Data Warehouse

The key characteristics of a data warehouse are as follows:

- Data is structured for simplicity of access and high-speed query performance
- End users are time-sensitive and desire speed-of-thought response times
- Large amounts of historical data are used
- Queries often retrieve large amounts of data, perhaps many thousands of rows
- Both predefined and ad hoc queries are common
- The data load involves multiple sources and transformations

In general, fast query performance with high data throughput is the key to a successful data warehouse.

Common Oracle Data Warehousing Tasks

As an Oracle data warehousing administrator or designer, you can expect to be involved in the following tasks:

- Configuring an Oracle database for use as a data warehouse
- Designing data warehouses
- Performing upgrades of the database and data warehousing software to new releases
- Managing schema objects, such as tables, indexes, and materialized views
- Managing users and security
- Developing routines used for the extraction, transformation, and loading (ETL) processes
- Creating reports based on the data in the data warehouse
- Backing up the data warehouse and performing recovery when necessary
- Monitoring the data warehouse's performance and taking preventive or corrective action as required

In a small-to-midsize data warehouse environment, you might be the sole person performing these tasks. In large, enterprise environments, the job is often divided among several DBAs and designers, each with their own specialty, such as database security or database tuning.

These tasks are illustrated in:

- *Oracle Database Data Warehousing Guide*
- *Oracle Database Administrator's Guide*
- *Oracle Database SQL Tuning Guide*

Concepts Illustrated in This Guide

This guide illustrates the following concepts:

1. Describes the Oracle Data Warehouse Reference Architecture.

   See Chapter 2, "Data Warehouse Reference Architecture".

2. Describes the ideas behind logical design.

   See Chapter 3, "Data Warehousing Logical Design".

3. Describes the ideas behind physical design.

   See Chapter 4, "Data Warehousing Physical Design".

4. Define the ideas behind using Exadata.

   See Chapter 5, "Introduction to Exadata".

5. Describes the ideas behind ODI.

   See Chapter 6, "ODI".

6. Describes the ideas behind Information Access.

   See Chapter 7, "Information Access".

Part II

Oracle's Data Warehouse Architecture

Part II discusses loading data into the data warehouse and includes:

2 Data Warehouse Reference Architecture

Oracle offers a general architecture for working with data warehouse deployments. The guidelines found in this architecture can be used in new deployments as well as in migrating existing ones.

This chapter includes the following topics:

- About Oracle's Data Warehouse Reference Architecture
- About Data Sources
- About Data Warehouse Layers
- About Business Intelligence Tools
- About ETL and the Data Loading Process
- About Security
- About Information Provisioning Process

About Oracle's Data Warehouse Reference Architecture

The goal of Oracle's data warehouse reference architecture is to deliver a high quality integrated system and information at a significantly reduced cost over the long term. It does this through recognizing the differing needs for data management and information access that must both be delivered by the warehouse, applying different types of data modeling to each in a layered and abstracted approach.

This reference architecture is intended as a guide and not an instruction manual. Each layer in the architecture has a role in delivering the analytical platform required to support business execution. This reference architecture gives us something to measure back against so we can understand what we compromise by making specific architectural, technical, and tools choices. It works equally well for new data warehouse deployments as it does for developing a roadmap to migrate any existing ones.

The elements involved in Oracle's reference architecture are:

- How Requirements are Evolving
- Requirements of a Reference Architecture
- Packaged Applications and the Reference Architecture

How Requirements are Evolving

Business needs as well as the database technologies and tools have changed considerably since data warehousing became popular. No longer isolated, data warehouses are increasingly seen as a fundamental element of every business process, and part of the platform that enables business execution by providing insight into operational systems. While efficient business processes must be automated and process-centric, effective ones must be information driven.

The pressing need to deliver additional business value out of the data is leading to a promotion of a number of additional design considerations as organizations look to establish a new foundation for business intelligence and data warehousing. These include:

- Real-Time and Mixed Workload

  The old notion of a single overnight batch load of data is long gone of course, but the operational requirements highlighted already drive the need to be able to load and query data at the same time without complexity around lock-escalations and reading dirty data which can compromise business trust and availability.

- Pervasive Reach

  Business intelligence is becoming pervasive, extending to stakeholders within the business as well as outside to trading partners, regulators, and customers. It is also being consumed by processes through web services as well as people using traditional tools. Importantly, once you view the provision of business intelligence as a service, you must also understand the impact this has on the platform used to deliver it. It must meet the same stringent performance and high availability requirements that must be satisfied by operational applications. Thus, data warehousing systems also become mission critical.

- Changing Tools

  Tools change over time and will probably continue to do so although at a slower rate due to the trend for tools consolidation. Any new data warehouse architecture must allow for such tool changes by separating the way the data is stored from any demands placed on the way the data is presented to the tools themselves.

- Analytical Requirements

  Businesses are much more analysis centric than they used to be, and, in many companies, performing this analysis often involves the use of spreadsheets and other specialist tools. This once again imposes additional platform costs and security risks, isolating users rather than allowing the sharing of insight and introduces latency. What is needed is for the data warehouse to facilitate real depth of analysis, not just provide simple dashboards and reports.

- Realistic Platform

  Data volumes and analysis needs fluctuate along with business cycles and can grow rapidly over time. To manage these demand changes requires a realistic platform - one that can scale to meet these demands, is affordable by the business, and can be managed with the skills available in the market.

- Data Quality

  Decisions based on bad data are inevitably going to be bad decisions. If the intention of the enterprise data warehouse is to support strategic, tactical, and operational decisions by providing access to the right information at the right time using the right tools, then data quality and availability become critical concerns. Even if the data is not perfect, and it rarely is, every effort must be made to profile and improve data quality, as well as publicize quality standards and issues so decisions can be made in full knowledge of any limitations.

- Regulatory Requirements

Growing regulatory requirements are forcing companies to look at their overall corporate governance, risk, and compliance strategies. This is driving more data to be retained, including unstructured data, and for it to be retained for longer.

- Roadmap and Managing Change

  It is one thing to recognize the need to change part of your architecture or tools landscape, and yet another to deliver it while still delivering on today's service-level agreements. Any new architecture must facilitate change through support for modern iterative design techniques as well as isolating changes to prevent them from causing ripple effects that require costly rework throughout the architecture.

- Unstructured Data

  The majority of data held within a business as a whole is unstructured. Companies can reduce costs through rationalizing and consolidating this information, make better decisions through distilling meaning from the data, and use the derived values to further enrich existing data.

Another key area that has changed is the rate at which businesses themselves are changing. In the past, it was probably reasonable to base your IT strategy on just running ERP from a single vendor. In many industries, given the rate of consolidation that is occurring, this is probably no longer a viable position to adopt. The data warehouse must not only be ready for change - it must embrace it as reliable information in those times of change, making it a real differentiator.

One approach, born out of frustration in the business or as a direct result of limitations in specialist data warehouse hardware platforms, has been to push the data downstream into specialized data marts. While delivering better access to the data from the business point of view, this fracturing of the information asset reduces the overall business value, increases data latency, increases costs to the business in the longer term and introduces additional risks with regard to security and compliance. It also further limits analysis requiring the integration of the data and encourages definitional differences of common measures between data marts. Fragmentation of a data warehouse environment should be avoided as it impacts decision accuracy and makes integration into the decision centric business more problematic.

Requirements of a Reference Architecture

To deliver BI pervasively across the business in a practical and cost-effective way, a data warehouse design should satisfy the following criteria:

- Balance the demands of information management and information access in a single design concept.
- Allow the information asset to be preserved and enriched over the long term without having to unnecessarily re-architect the data as the business changes, including changes to hierarchy and product attributions or styles, depth and tools used for analysis.
- Provide layers of abstraction so that changes to tools and the type of analysis undertaken, as well as the inclusion of new data sources through business acquisition, do not necessitate in themselves a ripple effect of changes through the rest of the architecture.
- Ensure security of the information asset and be able to prove the provenance of the data and who has queried it in order to satisfy regulatory requirements. This must be the case even for privileged database users and when the database has been lost through hardware and other failures.

- Facilitate the use of future hardware technologies that might further reduce the cost and complexity of managing huge volumes of data.
- Ensure that packaged applications are effectively integrated into the design.

Packaged Applications and the Reference Architecture

Developing a complete enterprise data warehouse from scratch may guarantee you 100% fit to requirements, but there is always a price to pay regarding costs, effort, elapsed time, and organizational focus. Building an enterprise data warehouse can be a major undertaking, especially for smaller organizations or ones with a more dynamic approach who may lack patience or executive support for such an undertaking.

While the core areas of any business that serve to differentiate the business in the market place will undoubtedly benefit from such a customized solution, other areas may not require such depth of analysis or flexibility.

About Data Sources

Data sources are all potential sources of raw data for the data warehouse, which are required by the business to address its information requirements.

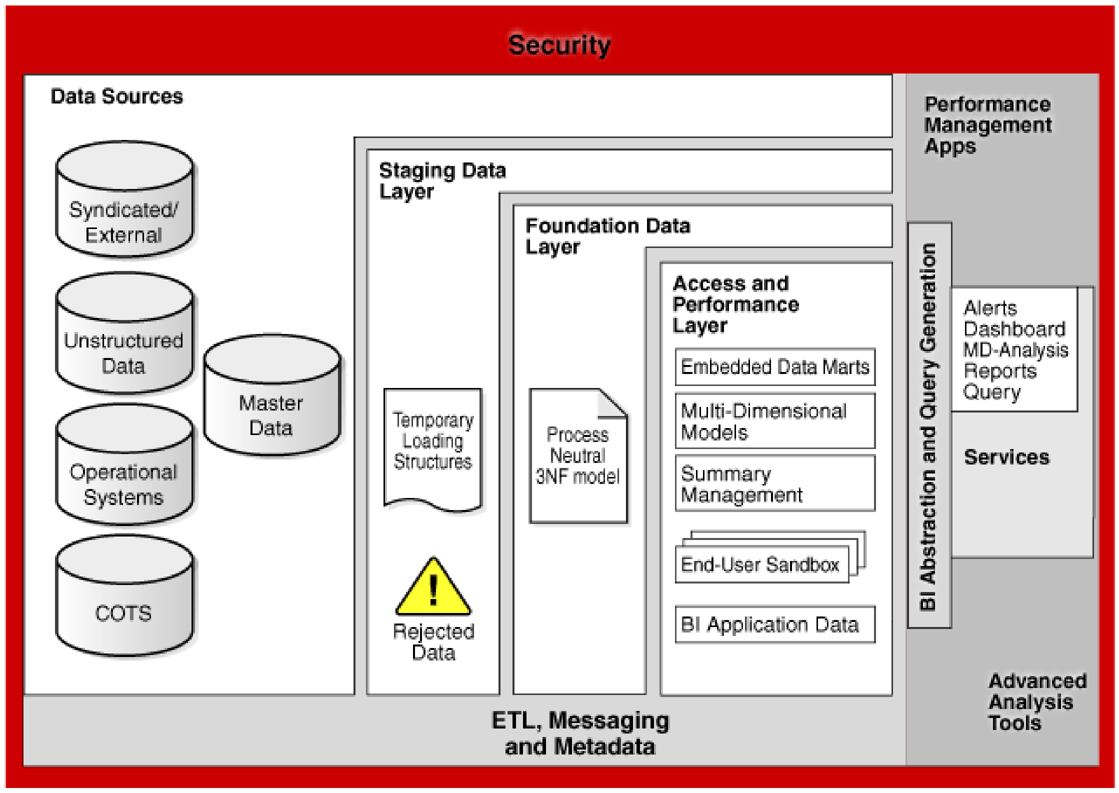
Sources include both internal and external systems and the data may be provided through a range of mechanisms such as real-time provisioning through messaging systems, change tracking through log mining, database replication, or simple file transfer. The approach taken will vary depending on source capability, capacity, regulatory compliance and access requirements.

While the majority of the data sources for a data warehouse are highly structured in nature, increasingly, there is also a need to supplement this with unstructured data. This data is typically either used for simple reference purposes, for example, a marketing analyst may want to see the marketing collateral used for a highly responsive segment of customers, or to enrich the structured data through further processing using text mining or classification, clustering, or feature-extraction techniques.

Master Data Management (MDM) solutions are considered to hold the master definition for any given business entity. The role of the data warehouse in respect to MDM is to preserve a record of changes across time, enable an analysis of these changes and provide the data as context to other analyses. Further data quality related checking may occur in the data warehouse as a separate process (not as part of the ETL), which may result in changes to master data. These changes are pushed back to the MDM solution and the new update received back into the warehouse through the standard flow.

Figure 2–1 illustrates Oracle's data warehouse reference architecture, and the data sources that are part of it.

Figure 2–1 Data Sources in the Reference Architecture



About Data Warehouse Layers

A data warehouse can be subdivided into three conceptual layers. One for staging the data. A second one, the foundation layer, for holding data at its lowest level of granularity. And a third, the access and performance layer. These are illustrated in Figure 2–1 and discussed in:

- Staging Data Layer
- Foundation Data Layer
- Access and Performance Layer

Staging Data Layer

The first destination of data that has been extracted from sources is the staging layer. This layer acts as a temporary storage area for data manipulation before it enters the data warehouse and serves to isolate the rate at which data is received into the data warehouse from the frequency at which data is refreshed and made available to end users. For example, in mobile telephony, universal mass storage devices will typically create files containing call details every 100,000 records or 10-minute interval, whichever is sooner. These will be loaded into the staging data layer as they are received, but only made available for querying as dictated by business requirements. That is, driven by business requirements and not the whim of the originating switch.

While many of the old rules regarding the static qualities of a data warehouse have now gone, it is still true that the data warehouse must contain data that is clean, consistent, and complete, as far as is practical. The staging data layer is where business rules are applied to achieve these objectives. Rejected data is retained in this layer for manual or automatic correction. As with all other layers in the design, the staging data layer is exposed to the BI tools layers so loading performance, including data quality information, can be made available to end users if appropriate.

Foundation Data Layer

The foundation data layer is sometimes referred to as the atomic data layer. As the name implies, this layer records data at the lowest possible level of granularity. It represents the heart of the data warehouse and is the layer responsible for managing the data over the long term.

The foundation data layer is modeled in a normalized fashion close to Third Normal Form (3NF) for storage efficiency. As the world of business is changing to rapidly, the data is also recorded in a business neutral fashion. This eliminates the impact of business changes and avoids any unnecessary re-structuring of data. For instance, a logical model may represent each level in a hierarchy in its own table (for example, sub-department, department, division, and group) with a fixed one-to-many relationship between each level. What if the number of levels changes? What if the organization changes to a network management structure or skips levels for some departments? Using some simple design patterns, it is possible to design around these sorts of issues.

The starting point for the logical model design may be a blank sheet of paper and map of existing source systems. More typical is to leverage an enterprise information model source from one of the industry bodies or an enterprise data warehousing model from database vendors such as Oracle or a specialist model vendor.

Some elements of adaptation of the logical model to meet local needs is typically required and any process oriented representations included in the model must be removed before it can be transformed into a final physical model. As the key relationships and entities are all identified in an enterprise model, it is safe to implement incrementally using any combination of bespoke development and COTS (commercial, off the shelf) packages and BI application implementations.

Access and Performance Layer

The normalized 3NF model used to improve data management in the Foundation Data Layer is not necessarily the best way to provide users access to the data as it is more difficult to navigate. Thus, an Access and Performance Layer is used to improve information access in the architecture. Most critical, though, is the realization that the tools used to access the data warehouse will change over time. You may have very little control over which tools are adopted by a particular business unit - we perhaps all wish this were not the case. As new tools may impose different requirements in terms of the way that data is structured, it is critically important that we must be able to create (or re-create) these structures from the underlying pool of data. That ability to re-constitute the data into different representation, either logically or physically is the fundamental purpose of the Access and Performance Layer.

Conceptually, this is the subject-oriented representation of a subset of data to simplify the analysis of the business while preserving the common dimensionality - nothing more, nothing less. Physical implementation in views or second tier aggregate structures is typically an implementation detail. They are created as required to facilitate access by a particular module or toolset. All aggregate structures available from Oracle are accessible from standard SQL, and can be added or required, leading to transparent performance improvements for end users and applications.

Population and update of access data layer structures can be highly automated, either using materialized views which keep track of data changes in sources or by extending the scope of ETL into an intra-ETL process.

The subject-oriented representations in this layer are referred to as embedded dependent data marts. Several advantages accrue through embedding these structures rather than pushing the data out into separate downstream data marts. These include:

- Reduced platform costs

  There is no need to buy a completely separate hardware platform on which to run the data mart. Additional network bandwidth, power, and cooling requirements are also reduced.

- Reduced data and analysis latency

  Data does not need to be off-loaded, transported, re-loaded and aggregated on a second platform. This reduces latency and can improve accuracy as well as a result because the data is less stale.

- Reduced development latency

  Development is often very rapid. With Oracle's ETL tools you can create a logical data mart design and then decide to either implement it relationally or multi-dimensionally by making a small metadata change.

- Improved security and management

  Embedding data marts eliminates the need administering a second platform. Furthermore, the Oracle database is robust, secure, and easily managed, which is often not the case for other platforms.

Specialized areas of analysis, such as data mining and forecasting, often necessitate data to be presented in very specific ways and may also involve data being modified and new data being written back. The notion of an analysis sandpit is also included in the design. Sandpits can be created on a project, user, or group basis. Data is provided and the analysis performed, reading and writing to the project area only. Once complete, any results flow back to the target platform or into the warehouse via the Staging Layer as normal. For example, data may be sampled from other structures in the Access and Performance Layer and restructured into a project schema. A data mining tool is then used to create a range of new customer segmentations. The best segmentation scheme is then selected through analysis and the new segment identifiers written back to the customer master data solution for each customer.

About Business Intelligence Tools

Many business tools can help design and administer a data warehousing environment.

There is a trend towards tools standardization to deliver a more pervasive reach, drive up development productivity and drive down overall cost of ownership. An important aspect of this is BI abstraction and query generation.

Most data warehouse solutions implement a range of end-user tools and applications to meet a series of business reporting and analysis requirements. Productivity issues are often associated with the wide range of such tools and the task of synchronizing any changes to data structures in the data warehouse with end-user reporting environments. If multiple tools are used to access the data warehouse, it is also very common for each tool to encode its own definition of common key performance indicators, so that even though the warehouse provides a single version of the truth, this truth is fractured and potentially changed within the BI tools.

Oracle's BI Server provides a further level of abstraction between the data warehouse and reporting tools so that the data warehouse data model and the tools can change at different rates. The BI Server is included in the architecture to provide a single enterprise view of the information regardless of the tools used to access the enterprise data warehouse. The BI Server provides a single consistent metadata model, including derived and complex KPIs for consumption by any BI tool with an ODBC interface.

Iterative design methodologies are typically based around the development of a conference room pilot in conjunction with key project stakeholders. For data warehousing, this can often be problematic as the data required will often not reside within the warehouse in a suitable form when the project begins. This either means the data has to be artificially generated based on the enterprise model or source system map (a challenge in itself) or a more waterfall approach adopted so the data elements can be put in place first so an understanding of the data can be built before the requirements are explored. This imposed order makes little sense and often results in rework.

The query federation capability of the BI Server offers an alternative development approach by allowing the reporting tools to attach directly to sources while the design is developed. Once requirements are understood, a more rigorous approach to the logical model is taken and the data provisioned through the data warehouse in the standard fashion. From the BI tools perspective, this only necessitates a change to the BI Server metadata physical mappings. This kind of query federation capability can be exceptionally useful for relatively modest data volumes and in a development context as described, but it clearly does not address the broader data integration, data quality, and production data volume challenges typically experienced in data warehousing. This second step of professionally managing the data in a warehouse is therefore essential and should be included in the project plan form the outset.

From a physical implementation perspective, the BI Server also offers a robust architectural infrastructure in the form of clustering for high availability, load balancing and data caching. This makes it a sound choice when considering a more pervasive BI capability.

About ETL and the Data Loading Process

You have to load data into the data warehouse so that you can query it. This is the data loading process, and is illustrated in Figure 2–2.

You can see that data is received through a variety of mechanisms into the initial staging data layer, both synchronously and asynchronously. It is processed through cleansing, enrichment, validation, and integration steps and made ready for loading into the foundation data layer. As previously outlined, the rate at which data is received onto the warehouse platform and the frequency at which data is refreshed into the warehouse, is driven by business needs.

Data is typically partitioned using a suitable schema such as date range and region. This allows for a more granular management throughout the life cycle. For example, indexes can be built on each partition rather than the complete data set and partitions of data loaded and unloaded from the foundation data layer as they enter and leave the data lifecycle.

Once data has been prepared in the Staging Data Layer, it can be moved into the Foundation Data Layer, as determined by the business requirement for the given flow of data. This requires a metadata change, but not data movement, which is more costly. The majority of the access and performance layer is made up of objects that will refresh automatically. This is true for views, materialized views, and cube-organized materialized views for example. In the case of materialized views and cube-organized materialized views, their definition will also define how they are refreshed, either when the data becomes stale or when a user queries the view, thus deferring the aggregation until some time later. For objects requiring load scripts to run, such as for

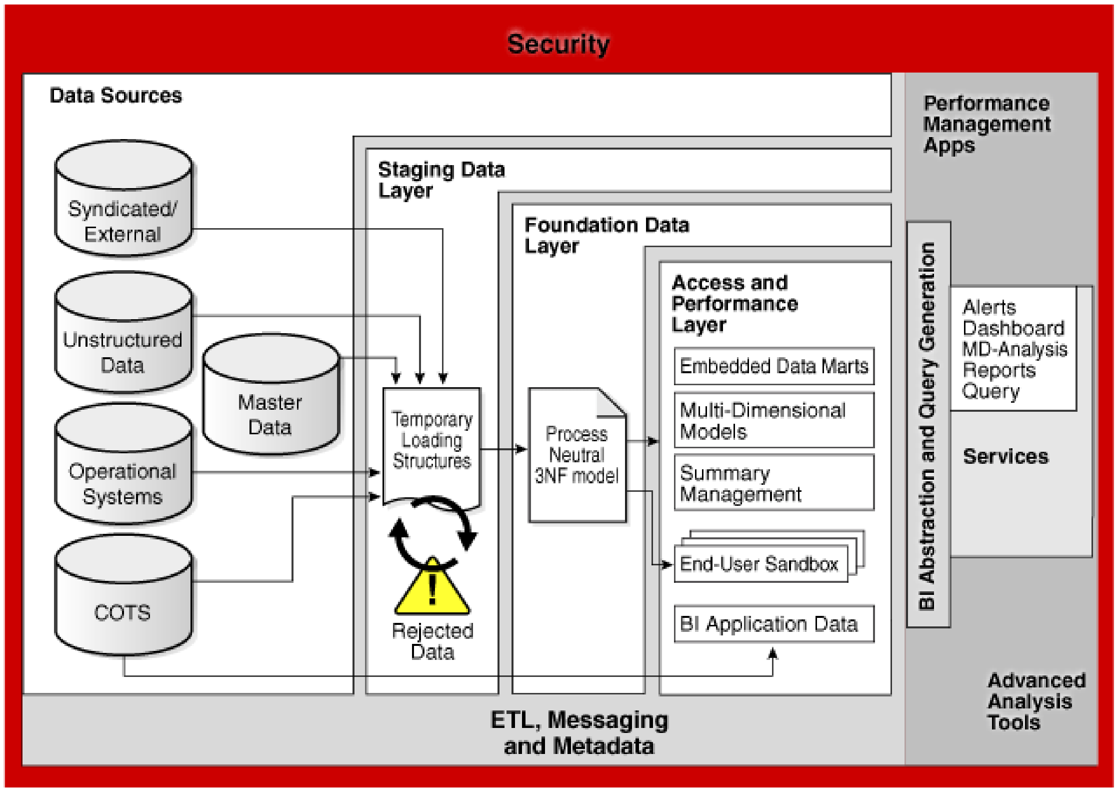
`CREATE TABLE AS SELECT` statement, an intra-ETL job will follow the loading of the Staging Data Layer.

In order for business users to have confidence in the data warehouse and for it to serve as the basis for regulatory reporting, the quality of the data and accuracy of queries are paramount. Oracle guarantees it will not read or write dirty data through the multi-version read consistency mechanism, which is unique in the industry.

There are two streams from COTS packages into the data warehouse. One, required for pre-packaged BI applications, will have pre-defined ETL as part of the application. The other flow, from COTS to Staging and Foundation Data Layers is processed in the standard fashion as for any other source flow. Any potential data conflict between the two flows is resolved in the BI abstraction layer to preserve a single version of the truth for the user.

Figure 2–2 illustrates the data loading process within the data warehouse reference architecture.

Figure 2–2 Data Loading Process



About Security

Data security is of increasing concern to every organization. Managing security is greatly simplified in Oracle's reference architecture by using a single data store with embedded data marts as opposed to multiple data marts standing alone.

Security extends throughout all layers in the reference architecture and is multi-faceted. The data warehouse and BI tools are subject to any corporate-wide security implementations such as LDAP and Active Directory. In addition, data should be protected in the database using role-based security at the row level as a minimum. The use of features such as Virtual Private Databases and fine grained label security may also be required depending on the nature of the data and the threat imposed. The use of additional database features, such as read-only tablespaces to prevent accidental changes to data can also be useful.

One best practice is to separate the duties of DBAs so that one may manage the database but not have access to the data it contains. This can be accomplished with Oracle Database Vault.

About Information Provisioning Process

Data can be referenced by any BI tool and includes any DW layer as well as ETL and data quality process metadata. This allows for a broader analytical capability to be offered, allowing depth of analysis as well as width of business process coverage. That said, the majority of queries will of course be against the Access and Performance Layer, as its entire purpose is to simplify user access to the data.

Performance management applications are also able to query the underlying source systems directly but this is effectively out of scope with respect to the data warehouse and is not covered in more detail in this chapter.

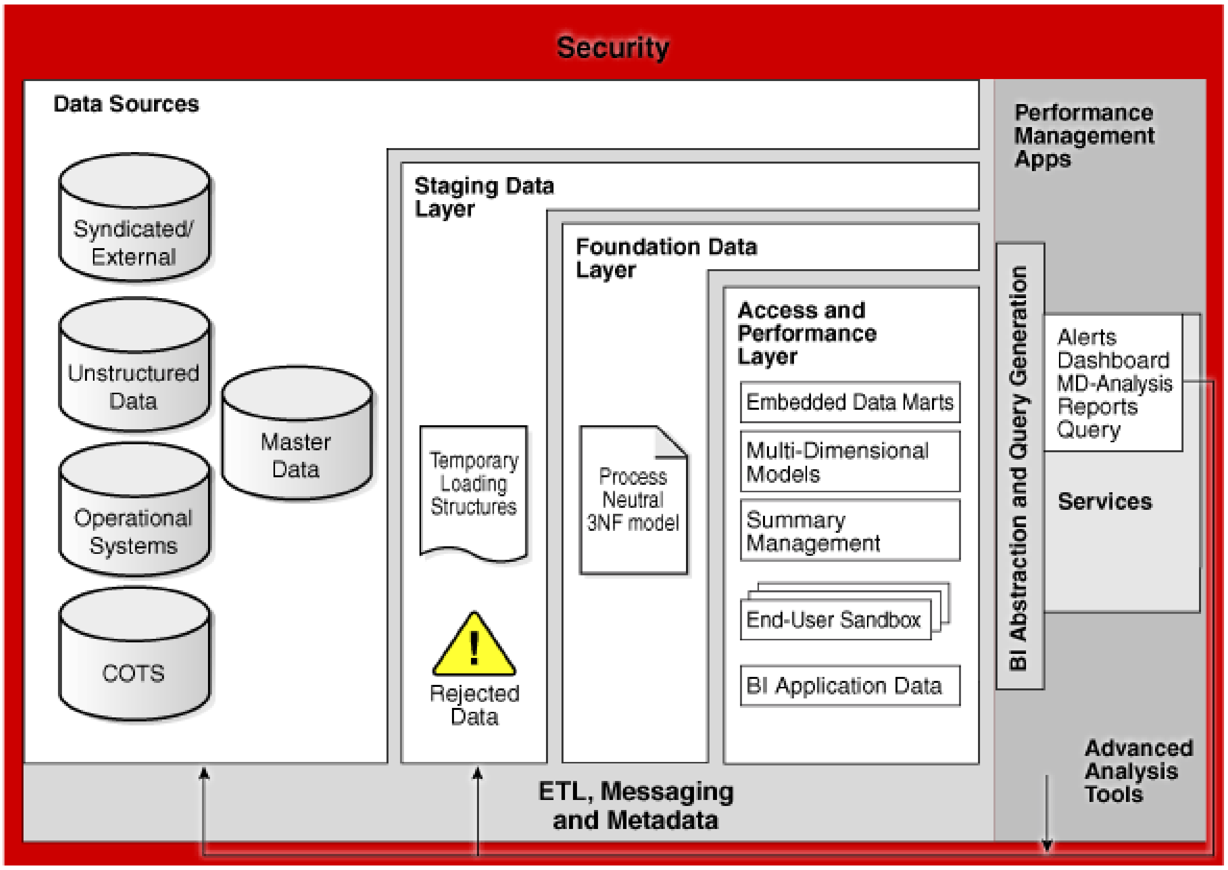
The BI server can also dynamically map a logical value to multiple sources based on metadata and make this available for querying. For instance, a real-time picture of intra-day sales may be generated by joining the data in the Access and Performance Layer with data in the Staging Data Layer that is yet to be made available.

The additional web service capability enables seamless operationalization of the data within the organization and to the wider trading community for solutions such as master data management and technologies, such as Business Process Execution Language (BPEL).

Advanced analysis tools and applications such as forecasting and data mining may through the analysis process create new data. Under the control of the tool or application, data can be read and written to and from the analysis sandpit area of the Access and Performance Layer. When a final set of data has been determined (such as a scored list of customers) a formal flow of data is executed that takes the data and moves it to the operational system, MDM solution or back into the data warehouse via the Staging Data Layer. Data is never loaded directly back from the Access and Performance Layer into the Foundation Data Layer.

Figure 2–3 illustrates the information provisioning process within the data warehouse reference architecture.

Figure 2–3 Data Provisioning Into BI Tools



Part III

Modeling

Part III discusses managing the data warehouse and includes:

3 Data Warehousing Logical Design

This chapter explains how to create a logical design for a data warehousing environment and includes the following topics:

- Logical Versus Physical Design in Data Warehouses
- Creating a Logical Design
- About Third Normal Form Schemas
- About Star Schemas
- About Data Warehousing Objects

Logical Versus Physical Design in Data Warehouses

Your organization has decided to build a data warehouse. You have defined the business requirements and agreed upon the scope of your application, and created a conceptual design. Now you need to translate your requirements into a system deliverable. To do so, you create the logical and physical design for the data warehouse. You then define:

- The specific data content
- Relationships within and between groups of data
- The system environment supporting your data warehouse
- The data transformations required
- The frequency with which data is refreshed

The logical design is more conceptual and abstract than the physical design. In the logical design, you look at the logical relationships among the objects. In the physical design, you look at the most effective way of storing and retrieving the objects as well as handling them from a transportation and backup/recovery perspective.

Orient your design toward the needs of the end users. End users typically want to perform analysis and look at aggregated data, rather than at individual transactions. However, end users might not know what they need until they see it. In addition, a well-planned design allows for growth and changes as the needs of users change and evolve.

By beginning with the logical design, you focus on the information requirements and save the implementation details for later.

Creating a Logical Design

A logical design is conceptual and abstract. You do not deal with the physical implementation details yet. You deal only with defining the types of information that you need.

One technique you can use to model your organization's logical information requirements is entity-relationship modeling. Entity-relationship modeling involves identifying the things of importance (entities), the properties of these things (attributes), and how they are related to one another (relationships).

The process of logical design involves arranging data into a series of logical relationships called entities and attributes. An entity represents a chunk of information. In relational databases, an entity often maps to a table. An attribute is a component of an entity that helps define the uniqueness of the entity. In relational databases, an attribute maps to a column.

To ensure that your data is consistent, you must use unique identifiers. A unique identifier is something you add to tables so that you can differentiate between the same item when it appears in different places. In a physical design, this is usually a primary key.

While entity-relationship diagramming has traditionally been associated with highly normalized models such as OLTP applications, the technique is still useful for data warehouse design in the form of dimensional modeling. In dimensional modeling, instead of seeking to discover atomic units of information (such as entities and attributes) and all of the relationships between them, you identify which information belongs to a central fact table and which information belongs to its associated dimension tables. You identify business subjects or fields of data, define relationships between business subjects, and name the attributes for each subject.

> **See Also:** *Oracle Database Data Warehousing Guide* for further information regarding dimensions

Your logical design should result in (1) a set of entities and attributes corresponding to fact tables and dimension tables and (2) a model of operational data from your source into subject-oriented information in your target data warehouse schema.

You can create the logical design using a pen and paper, or you can use a design tool.

> **See Also:** *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator* for a discussion of ODI

A schema is a collection of database objects, including tables, views, indexes, and synonyms. You can arrange schema objects in the schema models designed for data warehousing in a variety of ways. Most data warehouses use a dimensional model.

The model of your source data and the requirements of your users help you design the data warehouse schema. You can sometimes get the source model from your company's enterprise data model and reverse-engineer the logical data model for the data warehouse from this. The physical implementation of the logical data warehouse model may require some changes to adapt it to your system parameters—size of computer, number of users, storage capacity, type of network, and software.

Comparing Schema Design Approaches

This section describes concepts of Third Normal Form Schemas and Star Schemas. Before jumping into the concepts, it is helpful to have some historical background.

There is a long and contentious history between two main camps of schema designers working in data warehousing. On one side there have been proponents of third normal form schemas who discouraged star schemas for data warehousing. On the other side have been designers favoring star schemas (more generally referred to as dimensional modeling) who discouraged 3NF approaches to data warehousing. After decades of experience with data warehousing, the wrangling of 3NF versus star schemas has been much reduced. Today, practitioners recognize that 3NF schemas and star schemas each have their place and should be leveraged as appropriate. As Chapter 2, "Data Warehouse Reference Architecture", makes clear, your system will work best with a 3NF Foundation Layer for your data warehouse that supports an Access and Performance Layer built with star schemas. Data that is first placed in the 3NF foundation layer for flexible access is sent to the star schemas in the Access layer for excellent query performance. The two approaches are complementary, not competitive.

About Third Normal Form Schemas

Third Normal Form design seeks to minimize data redundancy and avoid anomalies in data insertion, updates and deletion. 3NF design has a long heritage in online transaction processing (OLTP) systems. OLTP systems must maximize performance and accuracy when inserting, updating and deleting data. Transactions must be handled as quickly as possible or the business may be unable to handle the flow of events, perhaps losing sales or incurring other costs. Therefore 3NF designs avoid redundant data manipulation and minimize table locks, both of which can slow inserts, updates and deletes. 3NF designs also works well to abstract the data from specific application needs. If new types of data are added to the environment, you can extend the data model with relative ease and minimal impact to existing applications. Likewise, if you have completely new types of analyses to perform in your data warehouse, a well-designed 3NF schema will be able to handle them without requiring redesigned data structures.

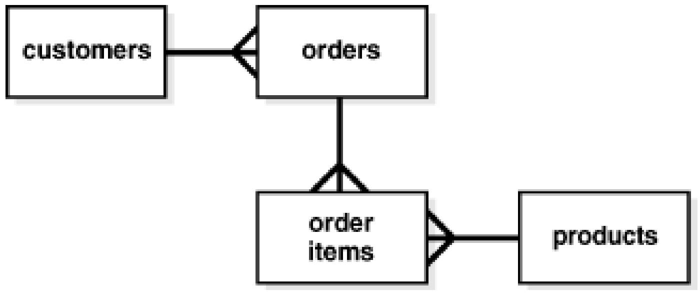
3NF designs have great flexibility, but it comes at a cost. 3NF databases use very many tables and this requires complex queries with many joins. For full scale enterprise models built in 3NF form, over one thousand tables are commonly encountered in the schema. With the kinds of queries involved in data warehousing, which will often need access to many rows from many tables, this design imposes understanding and performance penalties. It can be complex for query builders, whether they are humans or business intelligence tools and applications, to choose and join the tables needed for a given piece of data when there are very large numbers of tables available. Even when the tables are readily chosen by the query generator, the 3NF schema often requires that a large number of tables be used in a single query. More tables in a query mean more potential data access paths, which makes the database query optimizer's job harder. The end result can be slow query performance.

The issue of slow query performance in a 3NF system is not necessarily limited to the core queries used to create reports and analyses. It can also show up in the simpler task of users browsing subsets of data to understand the contents. Similarly the complexity of a 3NF schema may impact generating the pick-lists of data used to constrain queries and reports. Although these may seem relatively minor issues, speedy response time for such processes makes a big impact on user satisfaction.

Figure 3–1 presents a tiny fragment of a 3NF Schema. Note how order information is broken into order and order items to avoid redundant data storage. The "crow's feet" markings on the relationship between tables indicate one-to-many relationships among the entities. Thus one order may have multiple order items, a single customer may have many orders, and a single product may be found in many order items.

Although this diagram shows a very small case, you can see that minimizing data redundancy can lead to many tables in the schema.

Figure 3–1 Fragment of a Third Normal Form Schema



About Normalization

Normalization is a data design process that has a high level goal of keeping each fact in just one place to avoid data redundancy and insert/update/delete anomalies. There are multiple levels of normalization, and the material below describes the first three of them in informal terms, using a simple scenario involving sales orders. Considering how fundamental the term 3NF term is, it only makes sense to see how 3NF is reached.

Consider a situation where we are tracking sales. The core entity we track is sales orders, where each sales order contains details about each item purchased (referred to as a line item): its name, price, quantity, etc. The order also holds the name and address of the customer and more. Some orders have many different line items, and some orders have just one. An original sales order might look like this in terms of its paper form or screen display. For simplicity in this discussion, we ignore taxes and order totals.

Levels of Normalization

Below we show how the logical table design would look at different levels of normalization.

Logical table design is based on normalization, which has different levels. In first normal form, there are no repeating groups of data and no duplicate rows. Every intersection of a row and column (a field) contains just one value, and there are no groups of columns which contain the same facts. To avoid duplicate rows, there is a primary key. For our sales orders example, in first normal form we will not show multiple line items of each sales order in a single field of the table. Also, there will not be multiple columns showing line items.

Next is second normal form, where the design is in first normal form and that every non-key column is dependent on the complete primary key. Thus, we break the line items out into a table of sales order line items where each row represents one line item of one order. We can look at the line item table and see that the names of the items sold are not dependent on the primary key of the line items table: the sales item is its own entity. Therefore, we move the sales item to its own table showing the item name. Prices charged for each item can vary by order (for instance, due to discounts) so these remain in the line items table. In our sales order case, the name and address of the customer is not dependent on the primary key of the sales order: customer is its own entity. Thus, we move the customer name and address columns out into their own table of customer information.

Then is third normal form, where the design is in second normal form and that all columns are dependent on the whole key and nothing but the key. Another way of saying this is that there are no non-key columns which are facts about other non-key columns. For our `Customers` table, note that `City` and `State` are facts about `PostalCode`, because `PostalCode` determines the city and state. Therefore, we create a `PostalCode` table.

There are higher levels of normalization, all the way to sixth normal form, which can be useful, but they are outside the scope of this guide.

What if you have data at different levels of normalization across your database? Commonly, the database as a whole is considered to be normalized at the least normalized level of its data.

About Star Schemas

Star schemas are widely found in data warehousing systems. The term star schema is another way of referring to a "dimensional modeling" approach to defining your data warehouse. Most descriptions of dimensional modeling, including this, uses terminology drawn from the work of Ralph Kimball, the pioneering consultant and writer in this field. Dimensional modeling creates multiple star schemas, each based on a business process such as sales tracking or shipments. Each star schema can be considered a data mart, and perhaps as few as 20 data marts will cover the business intelligence needs of an enterprise. Compared to 3NF designs, the number of tables involved in dimensional modeling is a tiny fraction. Many star schemas will have under a dozen tables. The star schemas are knit together through conformed dimensions and conformed facts, described below. Thus users are able to get data from multiple star schemas with minimal effort.

The goal for star schemas is structural simplicity and high performance data retrieval. Since most queries in the modern era are generated by reporting tools and applications, it's vital to make the query generation convenient and reliable for the tools and application. Star schemas, through their great simplicity, offer a very major advantage for tool and application-based query generation: there are fewer ways to go astray and generate overly complex queries that perform poorly. In fact, many business intelligence tools and applications are designed with the expectation that a star schema representation will be available to them.

Discussions of star schemas are less abstracted from the physical database than 3NF descriptions. This is due to the very pragmatic emphasis of dimensional modeling on the needs of business intelligence users. Thus, this section will make much more direct reference to tables and how they are structured than the 3NF section did. This may seem like a move into the territory of the physical design chapter which follows, but it is not: the concepts discussed here do not focus on the physical implementation covered in the next chapter. Rather, they are part of the high-level planning of star schemas which will later be optimized through specific database objects.

Note how different the dimensional modeling style is from the 3NF approach that minimizes data redundancy and the risks of update/inset/delete anomalies. The star schema accepts data redundancy (denormalization) in its dimension tables for the sake of easy user understanding and better data retrieval performance. A common criticism of star schemas is that they limit analysis flexibility compared to 3NF designs. However, a well designed dimensional model can be extended to enable new types of analysis, and star schemas have been successful for many years at the largest enterprises. Oracle offers powerful optimization techniques for processing queries against star schemas, and these are discussed in *Oracle Database Data Warehousing Guide*.

See Also: *Oracle Database Data Warehousing Guide* for further information regarding dimensions

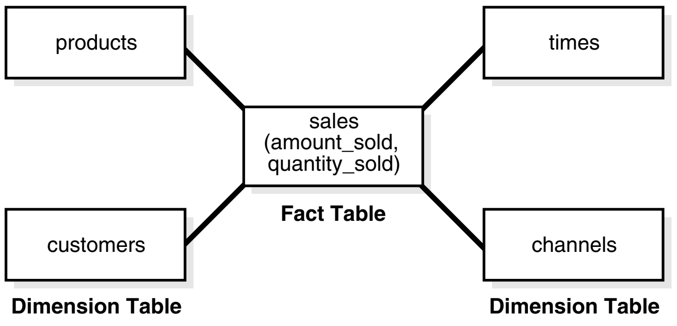
As noted earlier, the modern approach to data warehousing does not pit star schemas and 3NF against each other. Rather, both techniques are used, with a foundation layer of 3NF acting as the bedrock data, and star schemas as a central part of an access and performance optimization layer.

Facts and Dimensions

Star schemas divide data into facts and dimensions. Facts are the measurements of some event such as a sale and are typically numbers. Dimensions are the categories we use to identify facts, such as date, location and product.

The name "star schema" comes from the fact that the diagrams of the schemas typically show a central fact table with lines joining it to the dimension tables, so the graphic impression is similar to a twinkling star. Figure 3–2 is a simple example with sales as the fact table and products, times, customers, and channels as the dimension table.

Figure 3–2 Star Schema



A star schema optimizes performance by keeping queries simple and providing fast response time. All the information about each level is stored in one row.

Fact Tables

Fact tables have measurement data. They have many rows but typically not many columns. Fact tables for a large enterprise can easily hold billions of rows. For many star schemas, the fact table will represent well over 90 percent of the total storage space. A fact table has a composite key made up of the primary keys of the dimension tables of the schema.

A fact table contains either detail-level facts or facts that have been aggregated. Fact tables that contain aggregated facts are often called summary tables. A fact table usually contains facts with the same level of aggregation. Though most facts are additive, they can also be semi-additive or non-additive. Additive facts can be aggregated by simple arithmetical addition. A common example of this is sales. Non-additive facts cannot be added at all. An example of this is averages. Semi-additive facts can be aggregated along some of the dimensions and not along others. An example of this is inventory levels stored in physical warehouses, where you may be able to add across a dimension of warehouse sites, but you cannot aggregate across time.

In terms of adding rows to data in a fact table, there are three main approaches:

- Transaction based - Shows a row for the finest level detail in a transaction. A row is entered only if a transaction has occurred for a given combination of dimension values. This is the most common type of fact table.
- Periodic Snapshot - Shows data as of the end of a regular time interval, such as daily or weekly. If a row for the snapshot exists in a prior period, a row is entered for it in the new period even if no activity related to it has occurred in the latest interval. This type of fact table is useful in complex business processes where it is difficult to compute snapshot values from individual transaction rows.
- Accumulating snapshot - Shows one row for each occurrence of a short-lived process. The rows contain multiple dates tracking major milestones of a short-lived process. Unlike the other two types of fact tables, rows in an accumulating snapshot are updated multiple times as the tracked process moves forward.

Dimension Tables

Dimension tables provide category data to give context to the fact data. For instance, a star schema for sales data will have dimension tables for product, date, sales location, promotion and more. Dimension tables act as lookup or reference tables since their information lets you choose the values used to constrain your queries. The values in many dimension tables may change infrequently. As an example, a dimension of geographies showing cities may be fairly static. But when dimension values do change, it is vital to update them fast and reliably. Of course, there are situations where data warehouse dimension values change frequently. The customer dimension for an enterprise will certainly be subject to a frequent stream of updates and deletions.

A key aspect of dimension tables is the hierarchy information they provide. Dimension data typically has rows for the lowest level of detail plus rows for aggregated dimension values. These natural rollups or aggregations within a dimension table are called hierarchies and add great value for analyses. For instance, if you want to calculate the share of sales that a specific product represents within its specific product category, it is far easier and more reliable to have a predefined hierarchy for product aggregation than to specify all the elements of the product category in each query. Since hierarchy information is so valuable, it is common to find multiple hierarchies reflected in a dimension table.

Dimension tables are usually textual and descriptive, and you will use their values as the row headers, column headers and page headers of the reports generated by your queries. While dimension tables have far fewer rows than fact tables, they can be quite wide, with dozens of columns. A location dimension table might have columns indicating every level of its rollup hierarchy, and may show multiple hierarchies reflected in the table. The location dimension table could have columns for its geographic rollup, such as street address, postal code, city, state/province, and country. The same table could include a rollup hierarchy set up for the sales organization, with columns for sales district, sales territory, sales region, and characteristics.

The community of 3NF database designers has long had concern about dimension tables being highly denormalized. This concern is reasonable given the 3NF proponents' background in OLTP systems. However, the proven success of dimensional modeling across multiple decades in all types of domains has justified dimension tables' denormalized form.

Design Concepts in Star Schemas

Here we touch on some of the key terms used in star schemas. This is by no means a full set, but is intended to highlight some of the areas worth your consideration.

Data Grain

One of the most important tasks when designing your model is to consider the level of detail it will provide, referred to as the grain of the data. Consider a sales schema: will the grain be very fine, storing every single item purchased by each customer? Or will it be a coarse grain, storing only the daily totals of sales for each product at each store? In modern data warehousing there is a strong emphasis on providing the finest grain data possible, since this allows for maximum analytic power. Dimensional modeling experts generally recommend that each fact table store just one grain level. Presenting fact data in single-grain tables supports more reliable querying and table maintenance, since there is no ambiguity about the scope of any row in a fact table.

Working with Multiple Star Schemas

Since the star schema design approach is intended to chunk data into distinct processes, we need reliable and performant ways to traverse the schemas when queries span multiple schemas. One term for this ability is a data warehouse bus architecture. A data warehouse bus architecture can be achieved with conformed dimensions and conformed facts, described below.

Conformed Dimensions

Conformed dimensions means that dimensions are designed identically across the various star schemas. Conformed dimensions use the same values, column names and data types consistently across multiple stars. The conformed dimensions do not have to contain the same number of rows in each schema's copy of the dimension table, as long as the rows in the shorter tables are a true subset of the larger tables.

Conformed Facts

If the fact columns in multiple fact tables have exactly the same meaning, then they are considered conformed facts. Such facts can be used together reliably in calculations even though they are from different tables. Conformed facts should have the same column names to indicate their conformed status. Facts which are not conformed should always have different names to highlight their different meanings.

Surrogate Keys

Surrogate or artificial keys, usually sequential integers, are recommended for dimension tables. By using surrogate keys, the data is insulated from operational changes. Also, compact integer keys may allow for better performance than large and complex alphanumeric keys.

Degenerate Dimensions

Degenerate dimensions are dimension columns in fact tables that do not join to a dimension table. They are typically items such as order numbers and invoice numbers. You will see them when the grain of a fact table is at the level of an order line-item or a single transaction.

Junk Dimension

Junk dimensions are abstract dimension tables used to hold text lookup values for flags and codes in fact tables. These dimensions are referred to as junk, not because they have low value, but because they hold an assortment of columns for convenience, analogous to the idea of a "junk drawer" in your home. The number of distinct values (cardinality) of each column in a junk dimension table is typically small.

Embedded Hierarchy

Classic dimensional modeling with star schemas advocates that each table contain data at a single grain. However, there are situations where designers choose to have multiple grains in a table, and these commonly represent a rollup hierarchy. A single sales fact table, for instance, might contain both transaction-level data, then a day-level rollup by product, then a month-level rollup by product. In such cases, the fact table will need to contain a level column indicating the hierarchy level applying to each row, and queries against the table will need to include a level predicate.

Factless Fact Tables

Factless fact tables do not contain measures such as sales price or quantity sold. Instead, the rows of a factless fact table are used to show events not represented by other fact tables. Another use for factless tables is as a "coverage table" which holds all the possible events that could have occurred in a given situation, such as all the products that were part of a sales promotion and might have been sold at the promotional price.

Slowly Changing Dimensions

One of the certainties of data warehousing is that the way data is categorized will change. Product names and category names will change. Characteristics of a store will change. The areas included in sales territories will change. The timing and extent of these changes will not always be predictable. How can these slowly changing dimensions be handled? Star schemas treat these in three main ways:

- Type 1 - The dimension values that change are simply overwritten, with no history kept. This creates a problem for time-based analyses. Also, it invalidates any existing aggregates that depended on the old value of the dimension.
- Type 2 - When a dimension value changes, a new dimension row showing the new value and having a new surrogate key is created. We may choose to include date columns in our dimension showing when the new row is valid and when it is expired. No changes need be made to the fact table.
- Type 3 - When a dimension value is changed, the prior value is stored in a different column of the same row. This enables easy query generation if we want to compare results using the current and prior value of the column.

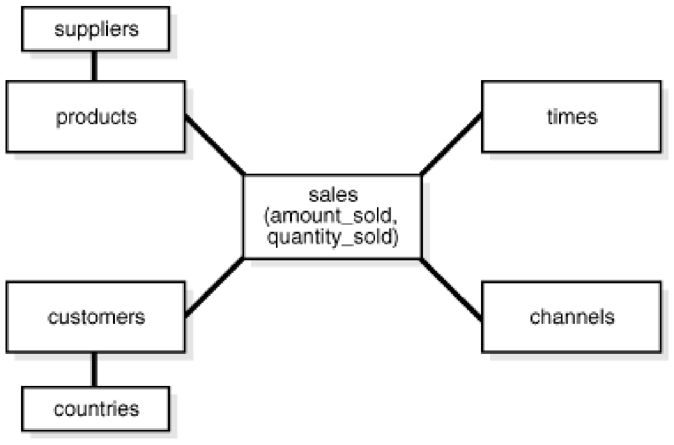
In practice, type 2 is the most common treatment for slowly changing dimensions.

About Snowflake Schemas

The snowflake schema is a more complex data warehouse model than a star schema, and is a type of star schema. It is called a snowflake schema because the diagram of the schema resembles a snowflake.

Snowflake schemas normalize dimensions to eliminate redundancy. That is, the dimension data has been grouped into multiple tables instead of one large table. For example, a product dimension table in a star schema might be normalized into a products table, a `product_category` table, and a `product_manufacturer` table in a snowflake schema. While this saves space, it increases the number of dimension tables and requires more foreign key joins. The result is more complex queries and reduced query performance. Figure 3–3 presents a graphical representation of a snowflake schema.

Figure 3–3 Snowflake Schema



Note: Oracle recommends you choose a star schema over a snowflake schema unless you have a clear reason not to.

About Data Warehousing Objects

Fact tables and dimension tables are the two types of objects commonly used in dimensional data warehouse schemas.

Fact tables are the large tables in your data warehouse schema that store business measurements. Fact tables typically contain facts and foreign keys to the dimension tables. Fact tables represent data, usually numeric and additive, that can be analyzed and examined. Examples include `sales`, `cost`, and `profit`.

Dimension tables, also known as lookup or reference tables, contain the relatively static data in the data warehouse. Dimension tables store the information you normally use to contain queries. Dimension tables are usually textual and descriptive and you can use them as the row headers of the result set. Examples are `customers` or `products`.

Data Warehousing Objects: Fact Tables

A fact table typically has two types of columns: those that contain numeric facts (often called measurements), and those that are foreign keys to dimension tables. A fact table contains either detail-level facts or facts that have been aggregated. Fact tables that contain aggregated facts are often called summary tables. A fact table usually contains facts with the same level of aggregation. Though most facts are additive, they can also be semi-additive or non-additive. Additive facts can be aggregated by simple arithmetical addition. A common example of this is sales. Non-additive facts cannot be added at all. An example of this is averages. Semi-additive facts can be aggregated along some of the dimensions and not along others. An example of this is inventory levels, where you cannot tell what a level means simply by looking at it.

Requirements of Fact Tables

You must define a fact table for each star schema. From a modeling standpoint, the primary key of the fact table is usually a composite key that is made up of all of its foreign keys.

Data Warehousing Objects: Unique Identifiers

Unique identifiers should be indicated as such in schema diagrams and design documentation. In Figure 3–4, they are represented with the # character.

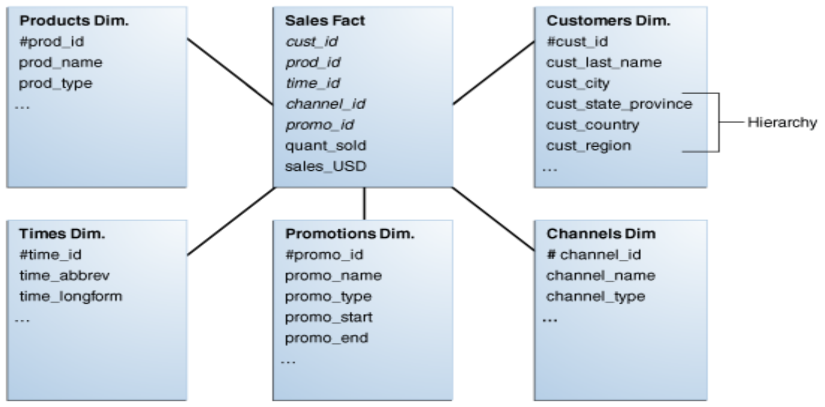
Data Warehousing Objects: Relationships

As you design your logical model, it is vital to consider issues of data integrity and to define the rules that will enforce them. For example, a sale transaction record must refer to a valid product ID and customer ID. For details on one of the ways this can be implemented, see "Integrity Constraints" on page 4-6.

Example of Data Warehousing Objects and Their Relationships

Figure 3–4 illustrates a simple example of a sales fact table and dimension tables customers, products, promotions, times, and channels. Because dimensions can be very wide, each dimension table has "…" at the bottom to indicate there would likely be many more attributes. The first attribute in each dimension table, preceded by the "#" sign, is the unique identifier for the dimension, and it has a defined relation to the sales fact table. The italicized elements of the sales table represent a composite unique identifier for each row in that table. In the customers dimension, there is an embedded hierarchy for location. Hierarchies could be expected in each of the dimensions, and likely multiple hierarchies in a single dimension. For simplicity, only one hierarchy is shown in the diagram.

Figure 3–4 Typical Data Warehousing Objects



4 Data Warehousing Physical Design

This chapter describes the physical design of a data warehousing environment, and includes the following topics:

- Moving from Logical to Physical Design
- About Physical Design
- About Compression

Moving from Logical to Physical Design

Logical design is what you draw with a pen and paper or design with Oracle Warehouse Builder or Oracle Designer before building your data warehouse. Physical design is the creation of the database with SQL statements.

During the physical design process, you convert the data gathered during the logical design phase into a description of the physical database structure. Physical design decisions are mainly driven by query performance and database maintenance aspects. For example, choosing a partitioning strategy that meets common query requirements enables Oracle Database to take advantage of partition pruning, a way of narrowing a search before performing it.

See Also:

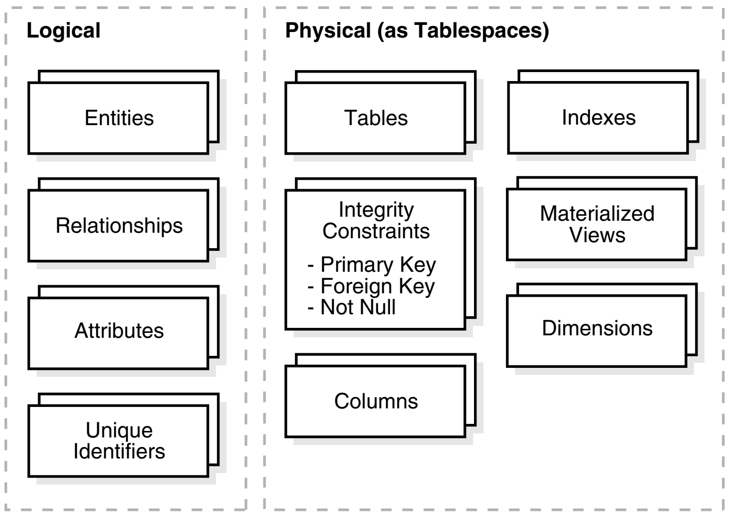
- *Oracle Database VLDB and Partitioning Guide* for further information regarding partitioning
- *Oracle Database Concepts* for further conceptual material regarding all design matters

About Physical Design

During the logical design phase, you defined a model for your data warehouse consisting of entities, attributes, and relationships. The entities are linked together using relationships. Attributes are used to describe the entities. The unique identifier (UID) distinguishes between one instance of an entity and another.

Figure 4–1 illustrates a graphical way of distinguishing between logical and physical designs.

Figure 4–1 Logical Design Compared with Physical Design



During the physical design process, you translate the expected schemas into actual database structures. At this time, you must map:

- Entities to tables
- Relationships to foreign key constraints
- Attributes to columns
- Primary unique identifiers to primary key constraints
- Unique identifiers to unique key constraints

Physical Design Structures

Once you have converted your logical design to a physical one, you must create some or all of the following structures:

- Tablespaces
- About Partitioning
- Views
- Integrity Constraints
- Indexes and Partitioned Indexes

Some of these structures require disk space. Others exist only in the data dictionary. Additionally, the following structures may be created for performance improvement:

- Materialized Views
- Dimensions

Tablespaces

A tablespace consists of one or more datafiles, which are physical structures within the operating system you are using. A datafile is associated with only one tablespace. From a design perspective, tablespaces are containers for physical design structures.

Tablespaces need to be separated by differences. For example, tables should be separated from their indexes and small tables should be separated from large tables.

Tablespaces should also represent logical business units if possible. Because a tablespace is the coarsest granularity for backup and recovery or the transportable tablespaces mechanism, the logical business design affects availability and maintenance operations.

You can now use ultralarge data files, a significant improvement in very large databases.

About Partitioning

Oracle Partitioning is an extremely important functionality for data warehousing, improving manageability, performance and availability. This section presents the key concepts and benefits of partitioning noting special value for data warehousing.

Partitioning allows a table, index or index-organized table to be subdivided into smaller pieces. Each piece of the database object is called a partition. Each partition has its own name, and may optionally have 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. This gives the administrator considerable flexibility in managing a partitioned object. However, from the perspective of the user, a partitioned table is identical to a non-partitioned table; no modifications are necessary when accessing a partitioned table using SQL DML commands.

Database objects - tables, indexes, and index-organized tables - are partitioned using a partitioning key, a set of columns that determine in which partition a given row will reside. For example a sales table partitioned on sales date, using a monthly partitioning strategy; the table appears to any application as a single, normal table. However, the DBA can manage and store each monthly partition individually, potentially using different storage tiers, applying table compression to the older data, or store complete ranges of older data in read only tablespaces.

Basic Partitioning Strategies

Oracle Partitioning offers three fundamental data distribution methods that control how the data is actually placed into the various individual partitions, namely:

- Range

  The data is distributed based on a range of values of the partitioning key (for a date column as the partitioning key, the 'January-2012' partition contains rows with the partitioning-key values between '01-JAN-2012' and '31-JAN-2012'). The data distribution is a continuum without any holes and the lower boundary of a range is automatically defined by the upper boundary of the preceding range.

- List

  The data distribution is defined by a list of values of the partitioning key (for a region column as the partitioning key, the 'North America' partition may contain values 'Canada', 'USA', and 'Mexico'). A special 'DEFAULT' partition can be defined to catch all values for a partition key that are not explicitly defined by any of the lists.

- Hash

  A hash algorithm is applied to the partitioning key to determine the partition for a given row. Unlike the other two data distribution methods, hash does not provide any logical mapping between the data and any partition.

Along with these fundamental approaches Oracle provides several more:

- Interval Partitioning

  An extension to range partitioning that enhances manageability. Partitions are defined by an interval, providing equi-width ranges. With the exception of the first partition all partitions are automatically created on-demand when matching data arrives.

- Partitioning by Reference

  Partitioning for a child table is inherited from the parent table through a primary key - foreign key relationship. Partition maintenance is simplified and partition-wise joins enabled.

- Virtual column based Partitioning

  Defined by one of the above mentioned partition techniques and the partitioning key is based on a virtual column. Virtual columns are not stored on disk and only exist as metadata. This approach enables a more flexible and comprehensive match of the business requirements.

Using the above-mentioned data distribution methods, a table can be partitioned either as single or composite partitioned table:

- Single (one-level) Partitioning

  A table is defined by specifying one of the data distribution methodologies, using one or more columns as the partitioning key. For example consider a table with a number column as the partitioning key and two partitions `less_than_five_hundred` and `less_than_thousand`, the `less_than_thousand` partition contains rows where the following condition is true: `500 <= Partitioning key <1000`.

  You can specify range, list, and hash partitioned tables.

- Composite Partitioning
- Combinations of two data distribution methods are used to define a composite partitioned table. First, the table is partitioned by data distribution method one and then each partition is further subdivided into subpartitions using a second data distribution method. All sub-partitions for a given partition together represent a logical subset of the data. For example, a range-hash composite partitioned table is first range-partitioned, and then each individual range-partition is further subpartitioned using the hash partitioning technique.

See Also:

- *Oracle Database VLDB and Partitioning Guide*
- *Oracle Database Administrator's Guide*
- *Oracle Database Concepts* for more information about Hybrid Columnar Compression

Index Partitioning

Irrespective of the chosen index partitioning strategy, an index is either coupled or uncoupled with the underlying partitioning strategy of the underlying table. The appropriate index partitioning strategy is chosen based on the business requirements, making partitioning well suited to support any kind of application. Oracle Database 12*c* differentiates between three types of partitioned indexes.

- Local Indexes

A local index is an index on a partitioned table that is coupled with the underlying partitioned table, 'inheriting' the partitioning strategy from the table. Consequently, each partition of a local index corresponds to one - and only one - partition of the underlying table. The coupling enables optimized partition maintenance; for example, when a table partition is dropped, Oracle simply has to drop the corresponding index partition as well. No costly index maintenance is required. Local indexes are most common in data warehousing environments.

- Global Partitioned Indexes

  A global partitioned index is an index on a partitioned or nonpartitioned table that is partitioned using a different partitioning-key or partitioning strategy than the table. Global-partitioned indexes can be partitioned using range or hash partitioning and are uncoupled from the underlying table. For example, a table could be range-partitioned by month and have twelve partitions, while an index on that table could be hash-partitioned using a different partitioning key and have a different number of partitions. Global partitioned indexes are more common for OLTP than for data warehousing environments.

- Global Non-Partitioned Indexes

  A global non-partitioned index is essentially identical to an index on a non-partitioned table. The index structure is not partitioned and uncoupled from the underlying table. In data warehousing environments, the most common usage of global non-partitioned indexes is to enforce primary key constraints.

Partitioning for Manageability

A typical usage of partitioning for manageability is to support a 'rolling window' load process in a data warehouse. Suppose that a DBA loads new data into a table on a daily basis. That table could be range partitioned so that each partition contains one day of data. The load process is simply the addition of a new partition. Adding a single partition is much more efficient than modifying the entire table, since the DBA does not need to modify any other partitions. Another advantage of using partitioning is when it is time to remove data. In this situation, an entire partition can be dropped, which is very efficient and fast, compared to deleting each row individually.

Partitioning for Performance

By limiting the amount of data to be examined or operated on, partitioning provides a number of performance benefits. Two features specially worth noting are:

- Partitioning Pruning: Partitioning pruning is the simplest and also the most substantial means to improve performance using partitioning. Partition pruning can often improve query performance by several orders of magnitude. For example, suppose an application contains an `ORDERS` table containing an historical record of orders, and that this table has been partitioned by day. A query requesting orders for a single week would only access seven partitions of the `ORDERS` table. If the table had two years of historical data, this query would access seven partitions instead of 730 partitions. This query could potentially execute 100x faster simply because of partition pruning. Partition pruning works with all of Oracle's other performance features. Oracle Database will utilize partition pruning in conjunction with any indexing technique, join technique, or parallel access method.

- Partition-wise Joins: Partitioning can also improve the performance of multi-table joins, by using a technique known as partition-wise joins. Partition-wise joins can be applied when two tables are being joined together, and at least one of these tables is partitioned on the join key. Partition-wise joins break a large join into smaller joins of 'identical' data sets for the joined tables. 'Identical' here is defined as covering exactly the same set of partitioning key values on both sides of the join, thus ensuring that only a join of these 'identical' data sets will produce a result and that other data sets do not have to be considered. Oracle is using either the fact of already (physical) equi-partitioned tables for the join or is transparently redistributing ("repartitioning") one table at runtime to create equipartitioned data sets matching the partitioning of the other table, completing the overall join in less time. This offers significant performance benefits both for serial and parallel execution.

Partitioning for Availability

Partitioned database objects provide partition independence. This characteristic of partition independence can be an important part of a high-availability strategy. For example, if one partition of a partitioned table is unavailable, all of the other partitions of the table remain online and available. The application can continue to execute queries and transactions against this partitioned table, and these database operations will run successfully if they do not need to access the unavailable partition. The database administrator can specify that each partition be stored in a separate tablespace; this would allow the administrator to do backup and recovery operations on an individual partition or sets of partitions (by virtue of the partition-to-tablespace mapping), independent of the other partitions in the table. Therefore in the event of a disaster, the database could be recovered with just the partitions comprising the active data, and then the inactive data in the other partitions could be recovered at a convenient time, thus decreasing the system down-time.

In light of the manageability, performance and availability benefits, it should be part of every data warehouse.

See Also: *Oracle Database VLDB and Partitioning Guide*

Views

A view is a tailored presentation of the data contained in one or more tables or other views. A view takes the output of a query and treats it as a table. Views do not require any space in the database.

See Also: *Oracle Database Concepts*

Integrity Constraints

Integrity constraints are used to enforce business rules associated with your database and to prevent having invalid information in the tables. Integrity constraints in data warehousing differ from constraints in OLTP environments. In OLTP environments, they primarily prevent the insertion of invalid data into a record, which is not a big problem in data warehousing environments because accuracy has already been guaranteed. In data warehousing environments, constraints are only used for query rewrite. `NOT NULL` constraints are particularly common in data warehouses. Under some specific circumstances, constraints need space in the database. These constraints are in the form of the underlying unique index.

See Also: *Oracle Database Concepts*

Indexes and Partitioned Indexes

Indexes are optional structures associated with tables or clusters. In addition to the classical B-tree indexes, bitmap indexes are very common in data warehousing environments. Bitmap indexes are optimized index structures for set-oriented operations. Additionally, they are necessary for some optimized data access methods such as star transformations.

Indexes are just like tables in that you can partition them, although the partitioning strategy is not dependent upon the table structure. Partitioning indexes makes it easier to manage the data warehouse during refresh and improves query performance.

See Also: *Oracle Database Concepts*

Materialized Views

Materialized views are query results that have been stored in advance so long-running calculations are not necessary when you actually execute your SQL statements. From a physical design point of view, materialized views resemble tables or partitioned tables and behave like indexes in that they are used transparently and improve performance.

See Also: *Oracle Database Data Warehousing Guide*

Dimensions

A dimension is a schema object that defines hierarchical relationships between columns or column sets. A hierarchical relationship is a functional dependency from one level of a hierarchy to the next one. A dimension is a container of logical relationships and does not require any space in the database. A typical dimension is city, state (or province), region, and country.

A dimension is a structure, often composed of one or more hierarchies, that categorizes data. Dimensional attributes help to describe the dimensional value. They are normally descriptive, textual values. Several distinct dimensions, combined with facts, enable you to answer business questions. Commonly used dimensions are customers, products, and time.

Dimension data is typically collected at the lowest level of detail and then aggregated into higher level totals that are more useful for analysis. These natural rollups or aggregations within a dimension table are called hierarchies.

Hierarchies

Hierarchies are logical structures that use ordered levels to organize data. A hierarchy can be used to define data aggregation. For example, in a time dimension, a hierarchy might aggregate data from the month level to the quarter level to the year level. A hierarchy can also be used to define a navigational drill path and to establish a family structure.

Within a hierarchy, each level is logically connected to the levels above and below it. Data values at lower levels aggregate into the data values at higher levels. A dimension can be composed of more than one hierarchy. For example, in the product dimension, there might be two hierarchies—one for product categories and one for product suppliers.

Dimension hierarchies also group levels from general to granular. Query tools use hierarchies to enable you to drill down into your data to view different levels of granularity. This is one of the key benefits of a data warehouse.

When designing hierarchies, you must consider the relationships in business structures. For example, a divisional multilevel sales organization.

Hierarchies impose a family structure on dimension values. For a particular level value, a value at the next higher level is its parent, and values at the next lower level are its children. These familial relationships enable analysts to access data quickly.

Levels A level represents a position in a hierarchy. For example, a time dimension might have a hierarchy that represents data at the month, quarter, and year levels. Levels range from general to specific, with the root level as the highest or most general level. The levels in a dimension are organized into one or more hierarchies.

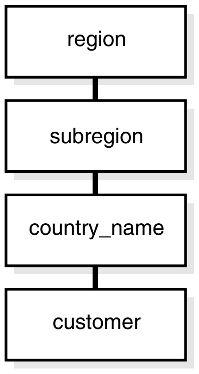
Level Relationships Level relationships specify top-to-bottom ordering of levels from most general (the root) to most specific information. They define the parent-child relationship between the levels in a hierarchy.

Hierarchies are also essential components in enabling more complex rewrites. For example, the database can aggregate an existing sales revenue on a quarterly base to a yearly aggregation when the dimensional dependencies between quarter and year are known.

Typical Dimension Hierarchy

Figure 4–2 illustrates a dimension hierarchy based on `customers`.

Figure 4–2 Typical Levels in a Dimension Hierarchy



See Also: *Oracle Database Data Warehousing Guide* for further information regarding hierarchies

About Compression

The benefits of compression are reduced disk space, better I/O throughput, and tiered storage. However, you must balance the benefits against the costs, which include CPU and processing overhead.

Part IV

Data Warehouse Deployment Considerations

Part IV discusses maintaining the data warehouse and includes:

5 Introduction to Exadata

The topics discussed in this chapter include:

Balanced Hardware Configuration

Regardless of the design or implementation of a data warehouse the initial key to good performance lies in the hardware configuration used. Many data warehouse operations are based upon large table scans and other I/O-intensive operations, which perform vast quantities of random I/Os. In order to achieve optimal performance the hardware configuration must be sized end to end to sustain this level of throughput. This type of hardware configuration is called a balanced system. In a balanced system all components - from the CPU to the disks - are orchestrated to work together to guarantee the maximum possible I/O throughput. This chapter begins by describing the key concepts of balanced systems and then presents information on the Oracle Exadata Database Machine, an engineered system available in multiple models, that has been designed from the start to achieve that balance. The material on the Oracle Exadata Database Machine covers much more than the balanced system topics: it presents the full range of software and hardware features that Exadata provides to bring maximum value to the enterprise.

To create a balanced system you must first understand how much throughput capacity is required for your system. There is no general answer to the question of how much I/O a system requires since it heavily depends on the workload running on the system. Third normal form data warehouses require high throughput since their dominant query type rely on good table scan performance such as hash joins. On the other side, star schema data warehouses require a high rate of random I/O operations because their operations rely more on index access such as bitmap access. Exadata provides both the I/O throughput and the random I/O operations to handle the most demanding third normal form and star schema data warehouses.

Two approaches to sizing a system can help you determine the I/O demand of the application. One is to define a lower and upper bound of the I/O demand. The other is to size the I/O according to the number and speed of cores used in the system.

Begin by considering the lower and upper I/O demand boundaries. A query operation can generally be divided into two parts: reading rows and processing rows. Reading rows is I/O intensive (or buffer cache intensive) while processing rows is CPU intensive. There are queries that are more I/O intensive and some that are more CPU intensive. It is necessary to determine which type of queries dominates the intended workload. I/O intensive queries define the upper boundary of the I/O throughput, while CPU intensive queries define the lower boundary. A typical system's workload will fall anywhere between the two extreme cases. To determine the throughput requirements of a system, it is necessary to analyze whether the workload contains more I/O or CPU intensive queries. This will be influenced by query complexity in terms of join operations. The calculations will also need to take into account the number of concurrent users. Separately from user queries, you need to understand workload created by ETL operations. During these operations write performance is also to be considered.

When the workload of a data warehouse is understood, then it is possible to make a rough estimate of the amount of I/O bandwidth which will be required. However, data warehouses often by their very nature have unpredictable and ad hoc workloads. Thus, there is often a need to have a simpler method of estimating I/O requirements. An alternative approach is to size the throughput requirements solely on the number CPU cores in your system and how much throughput each individual CPU or core in your configuration can drive. Both pieces of information can be determined from an existing system. If you are sizing a potential system, you should research the value of MB/second I/O throughput per core. This is an essential planning number for designing a balanced system. All subsequent critical components on the I/O path - the Host Bus Adapters (HBA), network infrastructure, the switches, the disk controllers, and the disks - have to be sized appropriately, and the starting point is knowing MB/sec I/O throughput per core. Enterprise data warehousing on Oracle uses Real Application Cluster (RAC) systems, so our discussion will be in terms of multi-node systems where each node can access all disks.

All the following need to be in balance to prevent bottlenecks:

- CPU cores - quantity and speed impact all the other calculations
- HBAs - number and speed of HBAs depend on cores
- Node Interconnect - I/O capacity depends on cores
- Switches - depend on HBA quantity and speed
- Disk Controllers - quantity and speed influenced by HBAs
- Disk Arrays - number and speed influenced by controllers

As your starting point, multiply the number of cores per node by the MB/second I/O per core. That result is the amount of I/O each node can drive. Therefore, each node will need enough HBA resources to support the calculated I/O level. The HBA quantity multiplied by HBA throughput must meet the I/O level the cores can drive.

Memory per core is another important consideration for system configuration. Memory per core needs to be adequate to handle memory-intensive operations such as large sorts. In turn, the memory must be allocated appropriately among the memory pool of the Oracle Database. Exadata systems provide generous amounts of memory, with multiple gigabytes per core, to handle the most demanding tasks.

> **See Also:** *Oracle Database Performance Tuning Guide* for further information

Now consider the network switches. An important way to size switches is to consider the I/O required to perform a full table scan using all nodes of your Oracle RAC system. Since each node has one HBA connected to each switch, multiply the number of nodes by the throughput of each HBA to find the total I/O demand on each switch. Every switch should be capable of handling the calculated I/O level from the nodes. On the disk-facing side of the switch, the same throughput is needed. Exadata optimizes performance by connecting servers to storage with InfiniBand technology switches.

The next stage of I/O sizing is the disk arrays: the disk controllers and disks must also be in line with our calculated throughput requirement. There is an important issue of disk size versus I/O throughput. It is tempting to have fewer, larger disks to reduce storage costs. However, fewer disks means fewer disk controllers and less I/O throughput for the disk array. Inadequate disk array throughput is a guarantee of performance bottlenecks. Do not be misled by large values for I/O operations per second (IOPS) or disk capacity (TB). The IOPS and TB figures cannot substitute for the essential value of I/O throughput.

Along with the I/O subsystem throughput, the cluster interconnect is another key area to size for Oracle RAC systems. The interconnect must be capable of I/O throughput equal to the number of cores in the cluster (the sum of cores across all nodes) times the MB/second per core. This large I/O requirement, has important justifications: to avoid bottleneck and to scale linearly for operations involving inter-node parallel execution. Here is the scenario for inter-node parallel execution.

- Once data is read off of the disks for a given query it will reside in process memory on one of the nodes in the cluster.
- Should another process on a different node require some or all of that data to complete this query, it will request the data to be passed over the interconnect rather than being read again from disk.
- If the interconnect bandwidth is not equal to the disk I/O bandwidth it will become a major bottleneck for the query and scalability will be compromised.

The sizing requirement for the interconnect highlights the need to use the highest throughput technology. Currently, InfiniBand technology is a wise choice for the interconnect; other approaches such as Gigabit Ethernet will require many more network interface cards per node to achieve the bandwidth necessary. InfiniBand provides a better solution for large scale systems as it consumes less CPU per message sent/received. Exadata is built on an optimized InfiniBand architecture that is used for both the interconnect and the storage system.

About Orion

Note that I/O validation should occur before an Oracle database is installed and data is loaded. In too many cases, poor I/O performance is only noticed after the database has been created, and often the only way to reconfigure the storage is to rebuild the database. Thus, much time and energy can be saved with a simple I/O performance test when the I/O system is first configured. Orion is a tool that Oracle provides to mimic a typical workload on a database system to calibrate the throughput. Use Orion to verify the maximum achievable throughput, even if a database has already been installed.

The types of supported I/O workloads are as follows:

- Small and random
- Large and sequential
- Large and random
- Mixed workloads

For each type of workload, Orion can run tests at different levels of I/O load to measure performance metrics such as MB per second, I/O per second, and I/O latency. You can run different I/O simulations depending upon which type of system you plan to build. Examples are the following:

- Daily workloads when users or applications query the system
- The data load when users may or may not access the system
- Index and materialized view builds
- Backup operations

See Also: *Oracle Database Performance Tuning Guide* for further information

About Disk Layout

Once you have confirmed the hardware configuration has been set up as a balanced system that can sustain your required throughput you need to focus on your disk layout.

In effect, the disk layout must also be balanced for the data warehouse needs. One of the key problems seen with existing data warehouse implementations is poor disk design. Often a large Enterprise Data Warehouse (EDW) can be found residing on the same disk array as one or more other applications. This design choice often occurs because the EDW does not generate the number of IOPS needed to saturate the disk array. However, there is a more important consideration than IOPS: the EDW will do fewer, larger I/Os than other applications, and the EDW I/Os will easily exceed the disk array's throughput capabilities in terms of gigabytes per second. Ideally you want your data warehouse to reside on its own storage array(s).

When configuring the storage subsystem for a data warehouse it should be simple, efficient, highly available and very scalable. It should not be complicated or hard to scale out. One of the easiest ways to achieve this is to apply the S.A.M.E. methodology (Stripe and Mirror Everything), which spreads data across the full storage subsystem. S.A.M.E. can be implemented at the hardware level or by using Oracle ASM (Automatic Storage Management). ASM provides file system and volume manager capabilities built into the Oracle database.

The rest of this chapter describes the Oracle Exadata Database Machine, an engineered system available in various models, that reflects these balanced configuration concepts across its design. The material that follows covers much more than balanced configuration since the Oracle Exadata Database Machine offers a very large set of features - both hardware and software - that support outstanding performance. The features make Oracle Exadata Database Machine an excellent platform for all kinds of tasks in the realms of both OLTP and data warehousing.

About Oracle Exadata

Oracle Exadata Database Machine provides an optimal solution for all database workloads, ranging from scan-intensive data warehouse applications to highly concurrent online transaction processing (OLTP) applications. With its combination of smart Oracle Exadata Storage Server Software, complete and intelligent Oracle Database software, and the latest industry-standard hardware components, Oracle Exadata Database Machine delivers extreme performance in a highly-available, highly-secure environment. Oracle provides unique clustering and workload management capabilities so Oracle Exadata Database Machine is well-suited for consolidating multiple databases into a single grid. Delivered as a complete pre-optimized, and pre-configured package of software, servers, and storage, Oracle

Exadata Database Machine is fast to implement, and it is ready to tackle your large-scale business applications.

The Oracle Exadata Database Machine is an easy to deploy solution for hosting the Oracle Database that delivers the highest levels of database performance available. The Exadata Database Machine is a "cloud in a box" composed of database servers, Oracle Exadata Storage Servers, an InfiniBand fabric for storage networking and all the other components required to host an Oracle Database. It delivers outstanding I/O and SQL processing performance for online transaction processing (OLTP), data warehousing (DW) and consolidation of mixed workloads. Extreme performance is delivered for all types of database applications by leveraging a massively parallel grid architecture using Real Application Clusters and Exadata storage. Database Machine and Exadata storage delivers breakthrough analytic and I/O performance, is simple to use and manage, and delivers mission-critical availability and reliability.

The Exadata Storage Server is an integral component of the Exadata Database Machine. Extreme performance is delivered by several unique features of the product. Exadata storage provides database aware storage services, such as the ability to offload database processing from the database server to storage, and provides this while being transparent to SQL processing and database applications. Hence just the data requested by the application is returned rather than all the data in the queried tables. Exadata Smart Flash Cache dramatically accelerates Oracle Database processing by speeding I/O operations. The Flash provides intelligent caching of database objects to avoid physical I/O operations and speeds database logging. The Oracle Database on the Database Machine is the first Flash enabled database. Exadata storage provides an advanced compression technology, Hybrid Columnar Compression, that typically provides 10x, and higher, levels of data compression. Exadata compression boosts the effective data transfer by an order of magnitude. The Oracle Exadata Database Machine is the world's most secure database machine. Building on the superior security capabilities of the Oracle Database, the Exadata storage provides the ability to query fully encrypted databases with near zero overhead at hundreds of gigabytes per second. The combination of these, and many other, features of the product are the basis of the outstanding performance of the Exadata Database Machine.

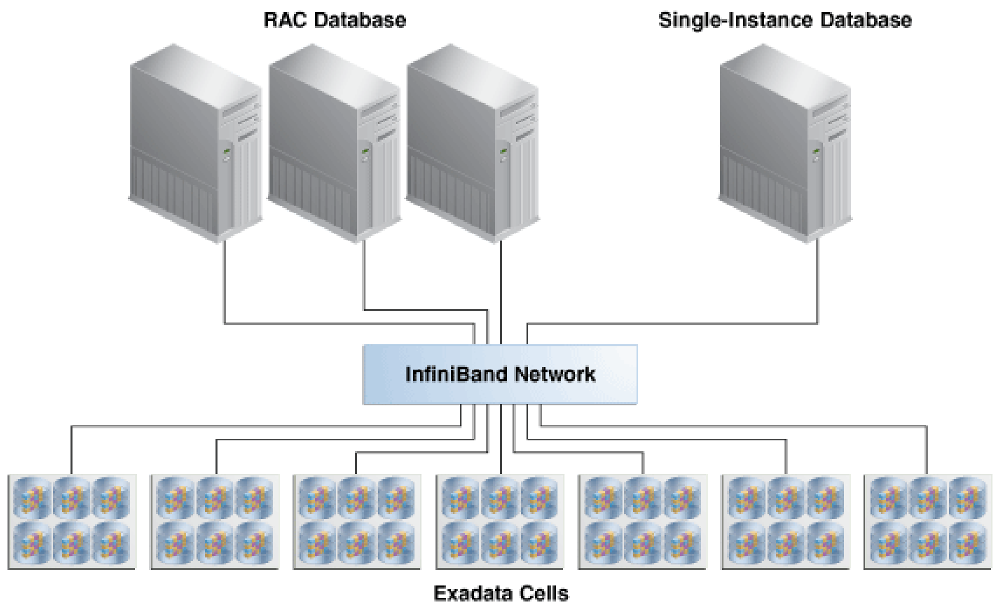
The Exadata Storage Expansion Rack enables the growth of Exadata storage capacity and bandwidth for X2-2 and X2-8 Exadata Database Machines. It is designed for database deployments that require very large amounts of data beyond what is included in an Exadata Database Machine and when additional database analytical processing power is not required. Standard Exadata Storage Servers, and supporting infrastructure, are packaged together in the Exadata Storage Expansion Rack to allow an easy to deploy extension of the Exadata storage configuration in an Exadata Database Machine. All the benefits and capabilities of Exadata storage are available and realized when using an Exadata Storage Expansion Rack.

The Exadata Database Machine has also been designed to work with, or independently of, the Oracle Exalogic Elastic Cloud. The Exalogic Elastic Cloud provides the best platform to run Oracle’s Fusion Middleware and Oracle’s Fusion applications. The combination of Exadata and Exalogic is a complete hardware and software engineered solution that delivers high-performance for all enterprise applications including Oracle EBusiness Suite, Siebel, and PeopleSoft applications.

Exadata Database Machine Architecture

Figure 5–1 illustrates a simplified schematic of a typical Database Machine Half Rack deployment. Two Oracle Databases, one Real Application Clusters (RAC) database deployed across three database servers and one single-instance database deployed on the remaining database server in the Half Rack, are shown. (Of course all four database servers could be used for a single four node Oracle RAC cluster.) The Oracle RAC database might be a production database and the single-instance database might be for test and development. Both databases are sharing the seven Exadata cells in the Half Rack but they would have separate Oracle homes to maintain software independence. All the components for this configuration – database servers, Exadata cells (storage servers), InfiniBand switches and other support hardware are housed in the Database Machine.

Figure 5–1 Exadata Machine Architecture



The Database Machine uses a state of the art InfiniBand interconnect between the servers and storage. Each database server and Exadata cell has dual port Quad Data Rate InfiniBand connectivity for high availability. Each InfiniBand link provides 40 Gigabits of bandwidth – many times higher than traditional storage or server networks. Further, Oracle's interconnect protocol uses direct data placement (DMA – direct memory access) to ensure very low CPU overhead by directly moving data from the wire to database buffers with no extra data copies being made. The InfiniBand network has the flexibility of a LAN network, with the efficiency of a SAN. By using an InfiniBand network, Oracle ensures that the network will not bottleneck performance. The same InfiniBand network also provides a high performance cluster interconnect for the Oracle Database Real Application Cluster (RAC) nodes.

Oracle Exadata is architected to scale-out to any level of performance. To achieve higher performance and greater storage capacity, additional database servers and Exadata cells are added to the configuration – for example, Half Rack to Full Rack upgrade. As more Exadata cells are added to the configuration, storage capacity and I/O performance increases near linearly. No cell-to-cell communication is ever done or required in an Exadata configuration.

When using Exadata, much SQL processing is offloaded from the database server to the Exadata cells. Exadata enables function shipping from the database instance to the underlying storage in addition to providing traditional block serving services to the database. One of the unique things the Exadata storage does compared to traditional storage is return only the rows and columns that satisfy the database query rather than the entire table being queried. Exadata pushes SQL processing as close to the data (or disks) as possible and gets all the disks operating in parallel. This reduces CPU consumption on the database server, consumes much less bandwidth moving data between database servers and storage servers, and returns a query result set rather than entire tables. Eliminating data transfers and database server workload can greatly benefit data warehousing queries that traditionally become bandwidth and CPU constrained. Eliminating data transfers can also have a significant benefit on online transaction processing (OLTP) systems that often include large batch and report processing operations.

Exadata is totally transparent to the application using the database. The exact same Oracle Database 12*c* Release 1 that runs on traditional systems runs on the Database Machine – but on Database Machine it runs faster. Existing SQL statements, whether ad hoc or in packaged or custom applications, are unaffected and do not require any modification when Exadata storage is used. The offload processing and bandwidth advantages of the solution are delivered without any modification to the application. All features of the Oracle Database are fully supported with Exadata. Exadata works equally well with single-instance or Real Application Cluster deployments of the Oracle Database. Functionality like Oracle Data Guard, Oracle Recovery Manager (RMAN), Oracle GoldenGate, and other database tools are administered the same, with or without Exadata. Users and database administrators leverage the same tools and knowledge they are familiar with today because they work just as they do with traditional non-Exadata storage.

Database Server Software

Oracle Database 12*c* Release 1 has been significantly enhanced to take advantage of Exadata storage. The Exadata software is optimally divided between the database servers and Exadata cells. The database servers and Exadata Storage Server Software communicate using the iDB – the Intelligent Database protocol. iDB is implemented in the database kernel and transparently maps database operations to Exadata-enhanced operations. iDB implements a function shipping architecture in addition to the traditional data block shipping provided by the database. iDB is used to ship SQL operations down to the Exadata cells for execution and to return query result sets to the database kernel. Instead of returning database blocks, Exadata cells return only the rows and columns that satisfy the SQL query. Like existing I/O protocols, iDB can also directly read and write ranges of bytes to and from disk so when offload processing is not possible Exadata operates like a traditional storage device for the Oracle Database. But when feasible, the intelligence in the database kernel enables, for example, table scans to be passed down to execute on the Exadata Storage Server so only requested data is returned to the database server.

iDB is built on the industry standard Reliable Datagram Sockets (RDSv3) protocol and runs over InfiniBand. ZDP (Zero-loss Zero-copy Datagram Protocol), a zero-copy implementation of RDS, is used to eliminate unnecessary copying of blocks. Multiple network interfaces can be used on the database servers and Exadata cells. This is an extremely fast low-latency protocol that minimizes the number of data copies required to service I/O operations.

Oracle Automatic Storage Management (ASM) is used as the file system and volume manager for Exadata. ASM virtualizes the storage resources and provides the advanced volume management and file system capabilities of Exadata. Striping database files evenly across the available Exadata cells and disks results in uniform I/O load across all the storage hardware. The ability of ASM to perform non-intrusive resource allocation, and reallocation, is a key enabler of the shared grid storage capabilities of Exadata environments. The disk mirroring provided by ASM, combined with hot swappable Exadata disks, ensure the database can tolerate the failure of individual disk drives. Data is mirrored across cells to ensure that the failure of a cell will not result in loss of data, or inhibit data accessibility. This massively parallel architecture delivers unbounded scalability and high availability.

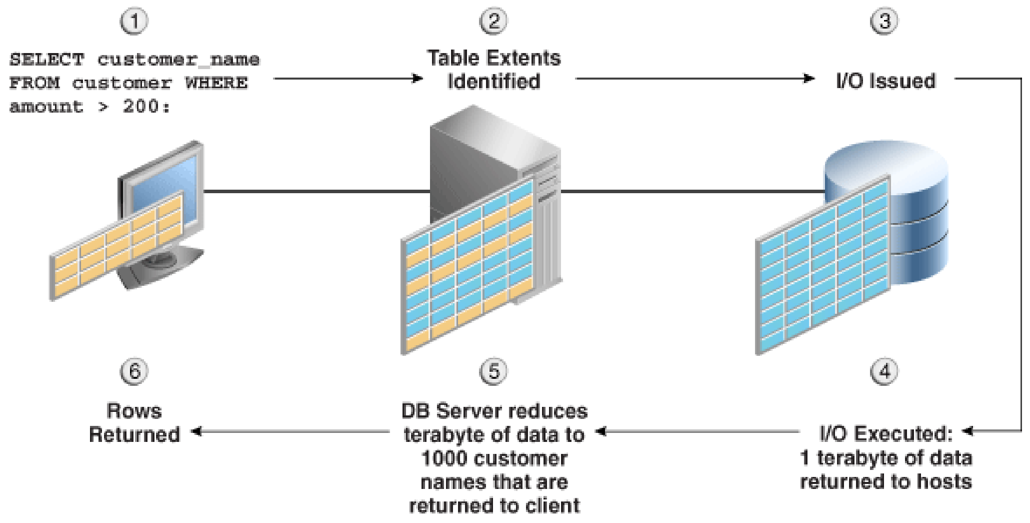
The Database Resource Manager (DBRM) feature in Oracle Database 11g has been enhanced for use with Exadata. DBRM lets the user define and manage intra- and inter-database I/O bandwidth in addition to CPU, undo, degree of parallelism, active sessions, and the other resources it manages. This allows the sharing of storage between databases without fear of one database monopolizing the I/O bandwidth and impacting the performance of the other databases sharing the storage. Consumer groups are allocated a percent of the available I/O bandwidth and the DBRM ensures these targets are delivered. This is implemented by the database tagging I/O with the associated database and consumer group. This provides the database with a complete view of the I/O priorities through the entire I/O stack. The intra-database consumer group I/O allocations are defined and managed at the database server. The inter-database I/O allocations are defined within the software in the Exadata cell and managed by the I/O Resource Manager. The Exadata cell software ensures that inter-database I/O resources are managed and properly allocated within, and between, databases. Overall, DBRM ensures each database receives its specified amount of I/O resources and user defined SLAs are met.

Two features of the Oracle Database that are offered exclusively on the Exadata Database Machine are the Oracle Database Quality of Service (QoS) Management and the QoS Management Memory Guard features. QoS Management allows system administrators to directly manage application service levels hosted on Oracle Exadata Database Machines. Using a policy-based architecture, QoS Management correlates accurate run-time performance and resource metrics, analyzes this data with its expert system to identify bottlenecks, and produces recommended resource adjustments to meet and maintain performance objectives under dynamic load conditions. Should sufficient resources not be available, QoS will preserve the more business critical objectives at the expense of the less critical ones. In conjunction with Cluster Health Monitor, QoS Management's Memory Guard detects nodes that are at risk of failure due to memory over-commitment. It responds by automatically preventing new connections thus preserving existing workloads and restores connectivity once the sufficient memory is again available.

Exadata Smart Scan Processing

With traditional, non-iDB aware storage, all database intelligence resides in the database software on the server. To illustrate how SQL processing is performed in this architecture, an example of a table scan is shown in Figure 5–2.

Figure 5–2 Exadata Scan Processing


1
SELECT customer_name FROM customer WHERE amount > 200;
2
Table Extents Identified
3
I/O Issued
4
I/O Executed: 1 terabyte of data returned to hosts
5
DB Server reduces terabyte of data to 1000 customer names that are returned to client
6
Rows Returned

The client issues a `SELECT` statement with a predicate to filter and return only rows of interest. The database kernel maps this request to the file and extents containing the table Oracle White Paper— A Technical Overview of the Oracle Exadata Database Machine and Exadata Storage Server being scanned. The database kernel issues the I/O to read the blocks. All the blocks of the table being queried are read into memory. Then SQL processing is done against the raw blocks searching for the rows that satisfy the predicate. Lastly the rows are returned to the client.

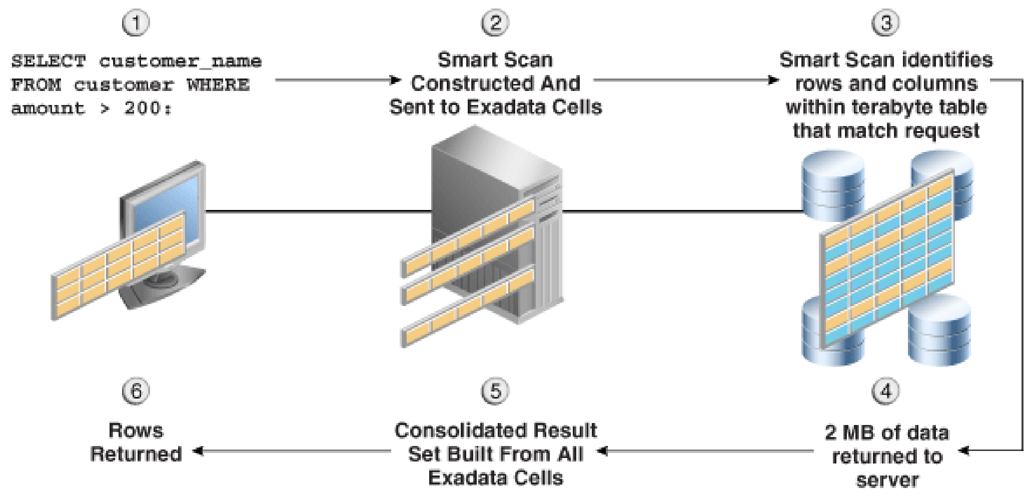
As is often the case with the large queries, the predicate filters out most of the rows read. Yet all the blocks from the table need to be read, transferred across the storage network and copied into memory. Many more rows are read into memory than required to complete the requested SQL operation. This generates a large number of data transfers which consume bandwidth and impact application throughput and response time.

Integrating database functionality within the storage layer of the database stack allows queries, and other database operations, to be executed much more efficiently. Implementing database functionality as close to the hardware as possible, in the case of Exadata at the disk level, can dramatically speed database operations and increase system throughput.

With Exadata storage, database operations are handled much more efficiently. Queries that perform table scans can be processed within Exadata storage with only the required subset of data returned to the database server. Row filtering, column filtering and some join processing (among other functions) are performed within the Exadata storage cells. When this takes place only the relevant and required data is returned to the database server.

Figure 5–3 below illustrates how a table scan operates with Exadata storage.

Figure 5–3 Exadata Scan and Storage


1
SELECT customer_name FROM customer WHERE amount > 200;
2
Smart Scan Constructed And Sent to Exadata Cells
3
Smart Scan identifies rows and columns within terabyte table that match request
4
2 MB of data returned to server
5
Consolidated Result Set Built From All Exadata Cells
6
Rows Returned

The client issues a `SELECT` statement with a predicate to filter and return only rows of interest. The database kernel determines that Exadata storage is available and constructs an iDB command representing the SQL command issued and sends it the Exadata storage. The `CELLSRV` component of the Exadata software scans the data blocks to identify those rows and columns that satisfy the SQL issued. Only the rows satisfying the predicate and the requested columns are read into memory. The database kernel consolidates the result sets from across the Exadata cells. Lastly, the rows are returned to the client.

Smart scans are transparent to the application and no application or SQL changes are required.

The SQL `EXPLAIN PLAN` statement shows when Exadata smart scan is used. Returned data is fully consistent and transactional and rigorously adheres to the Oracle Database consistent read functionality and behavior. If a cell dies during a smart scan, the uncompleted portions of the smart scan are transparently routed to another cell for completion. Smart scans properly handle the complex internal mechanisms of the Oracle Database including: uncommitted data and locked rows, chained rows, compressed tables, national language processing, date arithmetic, regular expression searches, materialized views and partitioned tables.

The Oracle Database and Exadata server cooperatively execute various SQL statements. Moving SQL processing off the database server frees server CPU cycles and eliminates a massive amount of bandwidth consumption which is then available to better service other requests. SQL operations run faster, and more of them can run concurrently because of less contention for the I/O bandwidth. We will now look at the various SQL operations that benefit from the use of Exadata.

Smart Scan Predicate Filtering

Exadata enables predicate filtering for table scans. Only the rows requested are returned to the database server rather than all rows in a table.

Smart Scan Column Filtering

Exadata provides column filtering, also called column projection, for table scans. Only the columns requested are returned to the database server rather than all columns in a table.

Smart Scan Join Processing

Exadata performs joins between large tables and small lookup tables, a very common scenario for data warehouses with star schemas. This is implemented using Bloom Filters, which are a very efficient probabilistic method to determine whether a row is a member of the desired result set.

Smart Scan Processing of Encrypted Tablespaces and Columns

Smart Scan offload processing of Encrypted Tablespaces (TSE) and Encrypted Columns is supported in Exadata storage. This enables increased performance when accessing the most confidential data in the enterprise.

Storage Indexing

Storage Indexes are a very powerful capability provided in Exadata storage that helps avoid I/O operations. The Exadata Storage Server Software creates and maintains a Storage Index (that is, metadata about the database objects) in the Exadata cell. The Storage Index keeps track of minimum and maximum values of columns for tables stored on that cell. When a query specifies a `WHERE` clause, but before any I/O is done, the Exadata software examines the Storage Index to determine if rows with the specified column value exist in the cell by comparing the column value to the minimum and maximum values maintained in the Storage Index. If the column value is outside the minimum and maximum range, scan I/O for that query is avoided. Many SQL Operations will run dramatically faster because large numbers of I/O operations are automatically replaced by a few lookups. To minimize operational overhead, Storage Indexes are created and maintained transparently and automatically by the Exadata Storage Server Software.

Offload of Data Mining Model Scoring

Data Mining model scoring is offloaded to Exadata. This makes the deployment of data warehouses on Database Machine an even better and more performant data analysis platform. All data mining scoring functions (for example, `prediction_probability`) are offloaded to Exadata for processing. This will not only speed warehouse analysis but reduce database server CPU consumption and the I/O load between the database server and Exadata storage.

Other Exadata Smart Scan Processing

Two other database operations that are offloaded to Exadata are incremental database backups and tablespace creation. The speed and efficiency of incremental database backups has been significantly enhanced with Exadata. The granularity of change tracking in the database is much finer when Exadata storage is used. Changes are tracked at the individual Oracle block level with Exadata rather than at the level of a large group of blocks. This results in less I/O bandwidth being consumed for backups and faster running backups.

With Exadata, the create file operation is also executed much more efficiently. For example, when issuing a `CREATE TABLESPACE` statement, instead of operating synchronously with each block of the new tablespace being formatted in server memory and written to storage, an iDB command is sent to Exadata instructing it to create the tablespace and format the blocks. Host memory usage is reduced and I/O associated with the creation and formatting of the tablespace blocks is offloaded. The I/O bandwidth saved with these operations means more bandwidth is available for other business critical work.

Exadata Smart Memory Scans

Exadata can provide all the performance benefits of in-memory databases, flash, and high performance storage in a single integrated solution. Using Oracle Database 11.2.0.3, one can execute in-memory parallel queries against table data in the buffer cache while simultaneously offloading the query to the Exadata storage, if necessary, to get additional additive performance. While the data throughput of Exadata from a combination of disk and flash is more than sufficient for most applications, applications that can leverage even more data throughput can run at over 200GB/sec using this in-memory query capability. With Hybrid Columnar Compression it is possible to store more table data in-memory and thus get higher effective scan bandwidths. This combination of in-memory parallel query and smart Exadata storage gives Exadata all the benefits of an in-memory solution but without losing the cost and capacity benefits of disk and flash.

About Hybrid Columnar Compression

Compressing data can provide dramatic reduction in the storage consumed for large databases. Exadata provides a very advanced compression capability called Hybrid Columnar Compression (HCC). Hybrid Columnar Compression enables the highest levels of data compression and provides enterprises with tremendous cost-savings and performance improvements due to reduced I/O. Average storage savings can range from 10x to 15x depending on how HCC is used. With average savings of 10x, IT managers can drastically reduce and often eliminate their need to purchase new storage for several years. For example, a 100 terabyte database achieving 10x storage savings would utilize only 10 terabytes of physical storage. With 90 terabytes of storage now available, IT organizations can delay storage purchases for a significant amount of time.

HCC is a new method for organizing data within a database block. As the name implies, this technology utilizes a combination of both row and columnar methods for storing data. This hybrid, or best of both worlds, 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 are detached from the set of rows, ordered and grouped together and then compressed. After the column data for a set of rows has been compressed, it is fit into the compression unit.

Smart Scan processing of HCC data is provided and column projection and filtering are performed within Exadata. Queries run directly on Hybrid Columnar Compressed data and do not require the data to be decompressed. Data that is required to satisfy a query predicate does not need to be decompressed, only the columns and rows being returned to the client are decompressed in memory. The decompression process takes place on the Exadata cell in order to maximize performance and offload processing from the database server. Given the typical tenfold compression of Hybrid Columnar Compressed Tables, this effectively increases the I/O rate ten-fold compared to uncompressed data.

Exadata Smart Flash Cache Features

Oracle has implemented a smart flash cache directly in the Oracle Exadata Storage Server. The Exadata Smart Flash Cache holds frequently accessed data in very fast flash storage while most of the data is kept in very cost effective disk storage. This happens automatically without the user having to take any action. The Oracle Flash Cache is smart because it knows when to avoid trying to cache data that will never be reused or will not fit in the cache. The Oracle Database and Exadata storage allow the user to provide directives at the database table, index and segment level to ensure that specific data is retained in flash. Tables can be moved in and out of flash with a simple command, without the need to move the table to different tablespaces, files or LUNs like you would have to do with traditional storage with flash disks.

The Exadata Smart Flash Cache is also used to reduce the latency of log write I/O eliminating performance bottlenecks that might occur due to database logging. The time to commit user transactions is very sensitive to the latency of log writes. Also, many performance critical database algorithms such as space management and index splits are also very sensitive to log write latency. Today Exadata storage speeds up log writes using the battery backed DRAM cache in the disk controller. Writes to the disk controller cache are normally very fast, but they can become slower during periods of high disk I/O. Smart Flash Logging takes advantage of the flash memory in Exadata storage to speed up log writes.

Flash memory has very good average write latency, but it has occasional slow outliers that can be one or two orders of magnitude slower than the average. The idea of the Exadata Smart Logging is to perform redo writes simultaneously to both flash memory and the disk controller cache, and complete the write when the first of the two completes. This literally gives Exadata the best of both worlds. The Smart Flash Logging both improves user transaction response time, and increases overall database throughput for I/O intensive workloads by accelerating performance critical database algorithms.

Smart Flash Logging handles all crash and recovery scenarios without requiring any additional or special administrator intervention beyond what would normally be needed for recovery of the database from redo logs. From a DBA perspective, the system behaves in a completely transparent manner and the DBA need not be concern themselves with the fact that flash is being used as a temporary store for redo. The only behavioral difference will be consistently low latencies for redo log writes.

I/O Resource Management with Exadata

With traditional storage, creating a shared storage grid is hampered by the inability to prioritize the work of the various jobs and users consuming I/O bandwidth from the storage subsystem. The same occurs when multiple databases share the storage subsystem. The DBRM and I/O resource management capabilities of Exadata storage can prevent one class of work, or one database, from monopolizing disk resources and bandwidth and ensures user defined SLAs are met when using Exadata storage. The DBRM enables the coordination and prioritization of I/O bandwidth consumed between databases, and between different users and classes of work. By tightly integrating the database with the storage environment, Exadata is aware of what types of work and how much I/O bandwidth is consumed. Users can therefore have the Exadata system identify various types of workloads, assign priority to these workloads, and ensure the most critical workloads get priority.

In data warehousing, or mixed workload environments, you may want to ensure different users and tasks within a database are allocated the correct relative amount of I/O resources. For example, you may want to allocate 70% of I/O resources to interactive users on the system and 30% of I/O resources to batch reporting jobs. This is simple to enforce using the DBRM and I/O resource management capabilities of Exadata storage.

An Exadata administrator can create a resource plan that specifies how I/O requests should be prioritized. This is accomplished by putting the different types of work into service groupings called Consumer Groups. Consumer groups can be defined by a number of attributes including the username, client program name, function, or length of time the query has been running. Once these consumer groups are defined, the user can set a hierarchy of which consumer group gets precedence in I/O resources and how much of the I/O resource is given to each consumer group. This hierarchy determining I/O resource prioritization can be applied simultaneously to both intra-database operations (that is, operations occurring within a database) and inter-database operations (that is, operations occurring among various databases).

When Exadata storage is shared between multiple databases you can also prioritize the I/O resources allocated to each database, preventing one database from monopolizing disk resources and bandwidth to ensure user defined SLAs are met.

Consolidating multiple databases on to a single Exadata Database Machine is a cost saving solution for customers. With Exadata Storage Server Software 11.2.2.3 and above, the Exadata I/O Resource Manager (IORM) can be used to enable or disable use of flash for the different databases running on the Database Machine. This empowers customers to reserve flash for the most performance critical databases.

In essence, Exadata I/O Resource Manager has solved one of the challenges traditional storage technology does not address: creating a shared grid storage environment with the ability to balance and prioritize the work of multiple databases and users sharing the storage subsystem. Exadata I/O resource management ensures user defined SLAs are met for multiple databases sharing Exadata storage. This ensures that each database or user gets the correct share of disk bandwidth to meet business objectives.

Quality of Service (QoS) Management with Exadata

Oracle Exadata QoS Management is an automated, policy-based product that monitors the workload requests for an entire system. It manages the resources that are shared across applications and adjusts the system configuration to keep the applications running at the performance levels needed by your business. It responds gracefully to changes in system configuration and demand, thus avoiding additional oscillations in the performance levels of your applications.

Oracle Exadata QoS Management monitors the performance of each work request on a target system. It starts to track a work request from the time a work request requests a connection to the database using a database service. The amount of time required to complete a work request, or the response time (also known as the end-to-end response time, or round-trip time), is the time from when the request for data was initiated and when the data request is completed. By accurately measuring the two components of response time (the time spent using resources and the time spent waiting to use resources), QoS Management can quickly detect bottlenecks in the system. It then makes recommendations to reallocate resources to relieve a bottleneck, thus preserving or restoring service levels. System administrators are alerted to the need for this reallocation and it is implemented with a simple button click on the QoS Management dashboard. Full details as to the entire cluster’s projected performance impact to this action are also provided. Finally, an audit log of all actions and policy changes is maintained along with historical system performance graphs.

Oracle Exadata QoS Management manages the resources on your system so that:

- When sufficient resources are available to meet the demand, business-level performance requirements for your applications are met, even if the workload changes.
- When sufficient resources are not available to meet the demand, Oracle Exadata QoS Management attempts to satisfy the more critical business performance requirements at the expense of less critical performance requirements.
- When load conditions severely exceed capacity, resources remain available.

Conclusion

Businesses today increasingly need to leverage a unified database platform to enable the deployment and consolidation of all applications onto one common infrastructure. Whether OLTP, DW, or mixed workload, a common infrastructure delivers the efficiencies and reusability the datacenter needs – and provides the reality of grid computing in-house. Building or using custom special purpose systems for different applications is wasteful and expensive. The need to process more data increases every day while corporations are also finding their IT budgets being squeezed. Examining the total cost of ownership (TCO) for IT software and hardware leads to choosing a common high performance infrastructure for deployments of all applications. By incorporating the Exadata based Database Machine into the IT infrastructure, companies will:

- Accelerate database performance and be able to do much more in the same amount of time.
- Handle change and growth in scalable and incremental steps by consolidating deployments on to a common infrastructure.
- Deliver mission-critical data availability and protection.

6 ODI

The topics discussed in this chapter include:

About ODI

This chapter provides a high-level introduction to Oracle Data Integrator Enterprise Edition. Because the material is intended for readers new to ODI, it covers more than just the data warehousing features of the product and it emphasizes business value rather than technical details.

Oracle is a leader in the data integration market, with the industry's most comprehensive fully-integrated offering in data integration, including Oracle Data Integrator Enterprise Edition, Oracle GoldenGate, Oracle Enterprise Data Quality. Oracle's data integration solutions provide continuous access to timely, trusted, and heterogeneous data across the enterprise to support both analytical and operational data integration.

Oracle Data Integrator provides a complete set of components for designing, deploying, and managing data integration processes. Data integration processes move and transform data from source data servers to target data servers, using an Extract-Load-Transform approach that eliminates the need for a transformation engine by delegating all transformations to the source and target data servers.

Capabilities included in Oracle Data Integrator Enterprise Edition 11*g*:

- High performance E-LT capabilities integrated with Oracle GoldenGate enable fast and efficient loading and transformation of real-time data into a data warehouse
- Better productivity with mapping wizards. Quick-Editor, generated code simulation and error table management
- Strong usability and manageability through integrations to Oracle JDeveloper and Oracle Enterprise Manager
- Increased performance by expressing commonality between different versions of the same source application with shortcuts and release tags
- Variable tracking feature to determine the actual values of variables and sequences that were used during a session

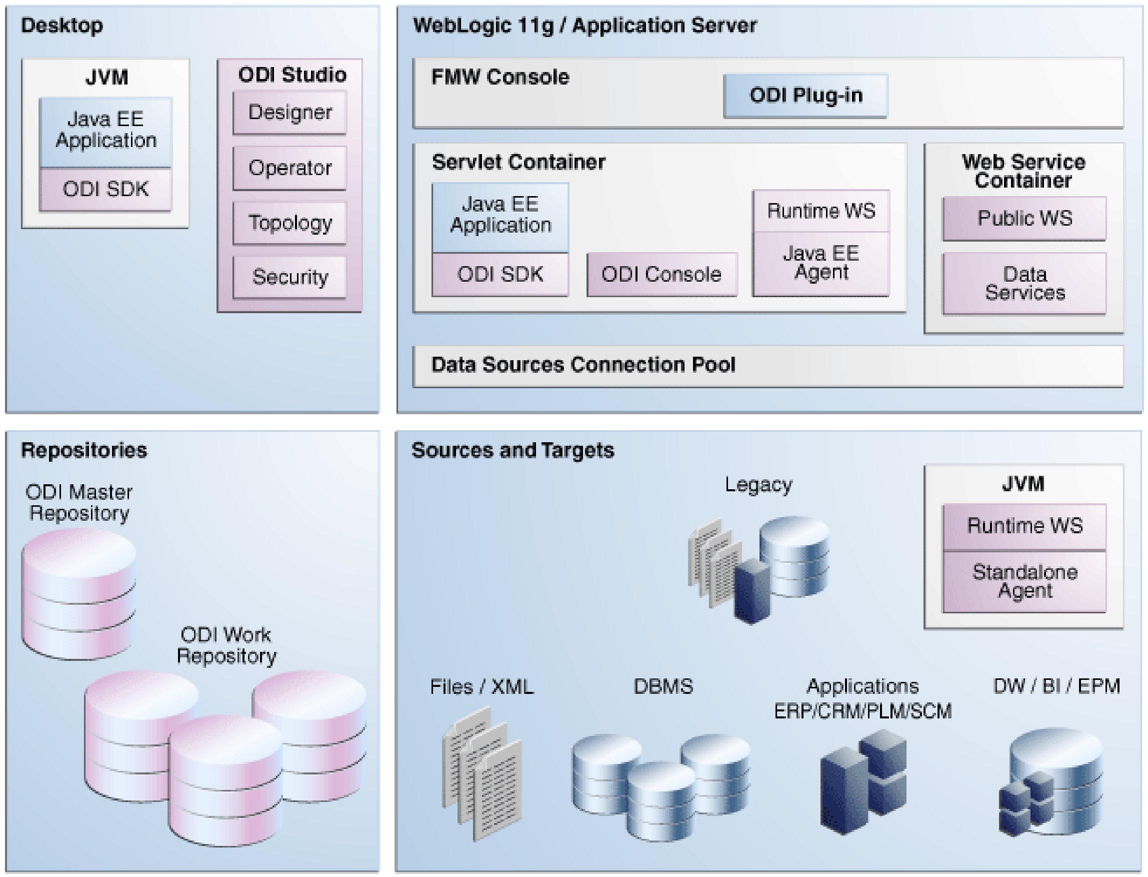
- Enhanced support for online analytical processing (OLAP), SAP and new APIs that can be embedded directly within custom applications
- Direct integration into Oracle WebLogic and Oracle Coherence deployments, for high availability and resilience
- Invocation of an Oracle Enterprise Data Quality (Datanomic) job

> **See Also:** *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator* for a detailed discussion of ODI

ODI Component Architecture

The ODI platform integrates in the broader Fusion Middleware platform and becomes a key component of this stack. ODI provides its run-time components as Java EE applications, enhanced to fully leverage the capabilities of the Oracle WebLogic Application Server or IBM WebSphere Application Server. ODI components include exclusive features for Enterprise-Scale Deployments, high availability, scalability, and hardened security. The run-time architecture is illustrated in Figure 6–1.

Figure 6–1 ODI Component Architecture



Where:

- Repository

  The repository stores the data integration design-time objects, the run-time objects (data integration scenarios to run) and the sessions corresponding to execution instances of these design-time and run-time objects. The architecture of the repository is designed to allow several separated environments that exchange metadata and scenarios (for example, Development, Test, Maintenance, and Production environments). The repository also acts as a version control system where objects are archived and assigned a version number.

The ODI Repository is composed of one Master Repository and several Work Repositories. Objects developed or configured through the user interfaces are stored in one of these repository types.

- Run-time agents

  There are the standalone agent or Java EE agents deployed in an application server, which run the sessions. The agent connects the repository as well as the source and target data servers when processing the data integration sessions. The agent also provides a web service for starting and monitoring scenario executions from third-party SOA-enabled applications.

- User Interfaces:

  - ODI Studio

    Administrators, Developers, and Operators use the ODI Studio to access the repositories. This Fusion Client Platform (FCP) based UI is used for administering the infrastructure (security and topology), reverse-engineering the metadata, developing projects, scheduling, operating, and monitoring executions. ODI Studio provides four Navigators for managing the different aspects and steps of an ODI integration project; Topology Navigator, Designer Navigator, Operator Navigator, and Security Navigator.

  - ODI Console

- Web services and data services: ODI can be integrated seamlessly in a SOA in several ways:

  - A dedicated public web service component provides operations to list the contexts and scenarios available. To use operations from this web service, you must first install and configure this component in a Java EE container.

  - Data Services are specialized

- Data Sources Connection Pool: The Data Services generated by ODI do not contain connection information for sources and targets. Instead, they make use of data sources defined with the Web Services container or on the application server. These JDBC data sources contain connection properties required to access data, and must correspond to data servers already defined within the ODI topology.

The data sources are used by ODI Java EE Agents to connect to a data server defined in the ODI topology. This allows the Java EE agent to benefit from the data sources and connection pooling features available on the application server. Connection pooling allows reusing connections across several sessions.

Smart Export and Import Feature

The Smart Export and Import feature is a lightweight and consistent export and import mechanism to move ODI objects between repositories when working with multiple environments such as Development, Quality Assurance, and Production. An object is automatically exported with all its object dependencies and a set of customizable object matching rules and actions is applied during the import.

Simplifying the Complexity of Data Integration

Oracle Data Integrator Enterprise Edition addresses multiple enterprise data integration needs:

- Data Warehousing and Business Intelligence-by executing high-volume, high performance loading of data warehouses, data marts, On Line Analytical Processing (OLAP) cubes, and analytical applications. It transparently handles incremental loads and slowly changes dimensions, manages data integrity and consistency, and analyzes data lineage.
- Service-Oriented Architecture-by calling on external services for data integration and by deploying data services and transformation services that can be seamlessly integrated within an SOA infrastructure. It adds support for high volume, high-performance bulk data processing to an existing service-oriented architecture.
- Master Data Management (MDM)-by providing a comprehensive data synchronization infrastructure for customers who build their own data hubs, work with packaged MDM solutions, or coordinate hybrid MDM systems with integrated SOA process analytics and Business Process Execution Language (BPEL) compositions.
- Migration-by providing efficient bulk load of historical data (including complex transformations) from existing systems to new ones. It continues to seamlessly synchronize data for as long as the two systems coexist.

E-LT Architecture for High Performance

Oracle Data Integrator Enterprise Edition's Extract, Load, Transform (E-LT) architecture leverages disparate relational database management systems (RDBMS) engines to process and transform the data. This approach optimizes performance and scalability and lowers overall solution costs. Instead of relying on a separate, conventional ETL transformation server, Oracle Data Integrator Enterprise Edition's E-LT architecture generates native code for disparate RDBMS engines (SQL, bulk loader scripts, for example). E-LT architecture extracts data from sources, loads it into a target, and transforms it using the database power. By leveraging existing databases and database expertise, Oracle Data Integrator Enterprise Edition provides unparalleled efficiency and lower cost of ownership. By reducing network traffic and transforming data in the database containing the target tables, E-LT architecture delivers the highest possible performance.

High-Productivity Designer for Data Integration

Oracle Data Integrator Enterprise Edition 11g includes a JDeveloper-based integrated development environment (IDE) called the Oracle Data Integrator Enterprise Edition Studio. This client is designed to dramatically increase the developer's productivity and make it easy to implement advanced features in data loading and transformation. Features like "quick-edit" support mass-updates, and intuitive and accessible keyboard navigation. In addition, Oracle Data Integrator Enterprise Edition follows a declarative design model, which simplifies common data integration design and deployment use cases, shortening implementation times. Data integration designers describe source and target data formats and data integration processes. The business user or the developer can focus on describing what to do, not how to do it. Oracle Data Integrator Enterprise Edition generates, deploys and manages the code required to implement those processes across the various source and target systems.

Enterprise Deployment for High Availability and Scalability

Oracle Data Integrator Enterprise Edition integrates to Oracle Fusion Middleware as a platform. Specifically, Oracle Data Integrator Enterprise Edition provides its run-time components as Java EE applications, enhanced to fully leverage the capabilities of Oracle WebLogic and Oracle Coherence. Oracle Data Integrator Enterprise Edition components include exclusive features for Enterprise-Scale Deployments, high availability, scalability, and hardened security.

High-Availability (HA) and Scalability is fully supported via clustered deployments for Java EE components. Oracle Data Integrator Enterprise Edition components deployed in WebLogic Server benefit from the capabilities of clustering for scalability, including JDBC connection pooling and load balancing. In addition to the cluster-inherited HA capabilities, the run-time agent also supports a connection retry mechanism to transparently recover sessions running in repositories that are stored in HA-capable database engines such as Oracle RAC.

Unified Administration and Management

Oracle Data Integrator Enterprise Edition simplifies complex data-centric deployments by improving visibility and control with a unified set of management interfaces. The Oracle Data Integrator Console leverages the Oracle Application Development Framework (ADF) and Ajax Framework for a rich user experience. Using this console, production users can set up an environment, export and import the repositories, manage run-time operations, monitor the sessions, diagnose the errors, browse design-time artifacts and generate lineage reports. In addition, this interface integrates seamlessly with the Oracle Enterprise Manager Fusion Middleware Control and allows administrators to monitor from a single screen not only their data integration components but their other Fusion Middleware components as well.

Knowledge Modules Provide Flexibility and Extensibility

Knowledge Modules are at the core of the Oracle Data Integrator Enterprise Edition architecture. They make all Oracle Data Integrator Enterprise Edition processes modular, flexible, and extensible. In addition, Oracle Data Integrator Enterprise Edition provides heterogeneous support for third party platforms, data-sources, and data warehousing appliances.

Knowledge Modules implement the actual data flows and define the templates for generating code across the multiple systems involved in each process. Knowledge Modules are generic, because they allow data flows to be generated regardless of the transformation rules. And they are highly specific, because the code they generate and the integration strategy they implement are finely tuned for a given technology.

Oracle Data Integrator Enterprise Edition provides a comprehensive library of Knowledge Modules, which can be tailored to implement existing best practices.

By helping companies capture and reuse technical expertise and best practices, Oracle Data Integrator Enterprise Edition's Knowledge Module framework reduces the cost of ownership. It also enables metadata-driven extensibility of product functionality to meet the most demanding data integration challenges.

Sample of available modules are:

- Oracle Database
- Generic SQL
- Hypersonic SQL

- IBM DB2/400, DB2 UDB
- Informix
- JD Edwards Enterprise One
- JMS
- Microsoft Access
- Microsoft SQL
- Netezza
- Oracle E-Business Suite
- Oracle Enterprise Service Bus
- Oracle GoldenGate
- Oracle Hyperion Essbase, Financial Management, Planning
- Oracle OLAP
- Oracle PeopleSoft
- Oracle Siebel CRM
- SAP ERP & BW
- Sybase ASE
- Sybase IQ
- Teradata

Oracle Data Integrator (ODI) helps facilitate the loading of a Data Warehouse using specialized Knowledge Modules. We note two of these briefly below.

Incremental Updates

Incremental Update can also be referred to as Type 1 Slowly Changing Dimension (see "About Star Schemas" on page 3-5 for information on slowly changing dimensions). Using Incremental Update Knowledge Modules, ODI will automatically perform insert and update operations based on a unique Update key defined by the end users.

Slowly Changing Dimension

Oracle Data Integrator provides Knowledge Modules that implement a Type 2 Slowly Changing Dimension for a Data Warehouse. The Data Integration developers will simply choose this Knowledge Module in a list to implement this complex integration strategy.

These two knowledge modules are just a small example of how ODI helps developers rapidly design data integration processes to load a Data Warehouse without worrying about defining every single step of a complex integration process.

Load Plans

Oracle Data Integrator is often used for populating very large data warehouses. In these use cases, it is common to have thousands of tables being populated using hundreds of ODI scenarios (scenarios are executable objects stored in ODI work repositories). The execution of these scenarios has to be organized in such a way that the data throughput from the sources to the target is the most efficient within the batch window. Load Plans help the user organizing the execution of scenarios in a hierarchy of sequential and parallel steps for these types of use cases.

A Load Plan is an executable object in Oracle Data Integrator that can contain a hierarchy of steps that can be executed conditionally, in parallel or in series. Load Plan allow setting and using variables at multiple levels. They also support exception handling strategies in the event of a scenario ending in error. A Load Plan can be modified in production environments and steps can be enabled or disabled according to the production needs.

Benefits of E-LT Combined with a Business-Rule Driven Approach

Compared to other architectures (manual coding and traditional ETL), ODI mixes the best of both E-LT and Business-rules:

- Productivity/Maintenance
  - The business-rules driven approach delivers greater productivity as developers simply need to concentrate on the "What" without caring about the "How". They define SQL expressions for the business rules, and ODI Knowledge Modules generate the entire set of SQL operations needed to achieve these rules.
  - When a change needs to be made in operational logic (such as "creating a backup copy of every target table before loading the new records"), it is simply applied in the appropriate Knowledge Module and it automatically impacts the hundreds of interfaces already developed. With a traditional ETL approach, such a change would have necessitated opening every job and manually adding the new steps, increasing the risk of mistakes and inconsistency.
  - Flexibility and a shallow learning curve are ensured by leveraging the RDBMS' latest features.
  - With a centralized repository that describes all the metadata of the sources and targets and a single unified and comprehensive graphical interface, maintenance is greatly optimized as cross-references between objects can be queried at any time. This gives the developers and the business users a single entry point for impact analysis and data lineage ("What is used where?", "Which sources populate which targets?" etc.)
  - In the ODI repository, the topology of the infrastructure is defined in detail, and moving objects between different execution contexts (Development, Testing, QA, Production, etc.) is straightforward. With a powerful version control repository, several teams can work on the same project within different release stages, with guaranteed consistency of deliverables.
  - With a centralized framework for Data Quality, developers spend less time on defining technical steps, and more time on the specification of data quality rules. This helps to build a consistent and standardized Data Warehouse.
- High Performance
  - The E-LT architecture leverages the power of all the features of in-place databases engines. ODI generates pure set-oriented SQL optimized for each RDBMS which can take advantage of advanced features such as parallel processing or other advanced features.
  - Native database utilities can be invoked by the ODI Knowledge Modules provided.

  - When data from the target database is referenced - table lookups for example - it does not need to be extracted from the database, into an engine. It remains where it is, and it is processed by database engine.
- Low Cost
  - Oracle Data Integrator doesn't require a dedicated server. The loads and transformations are carried out by the RDBMS.

With its business-rule driven E-LT architecture, Oracle Data Integrator is a powerful solution for moving beyond the worlds of manual coding and traditional ETL to achieve greater productivity, faster performance and lower cost.

7 Information Access

The topics discussed in this chapter include:

About Information Access

Accessing data is a key element of creating a successful database architecture.

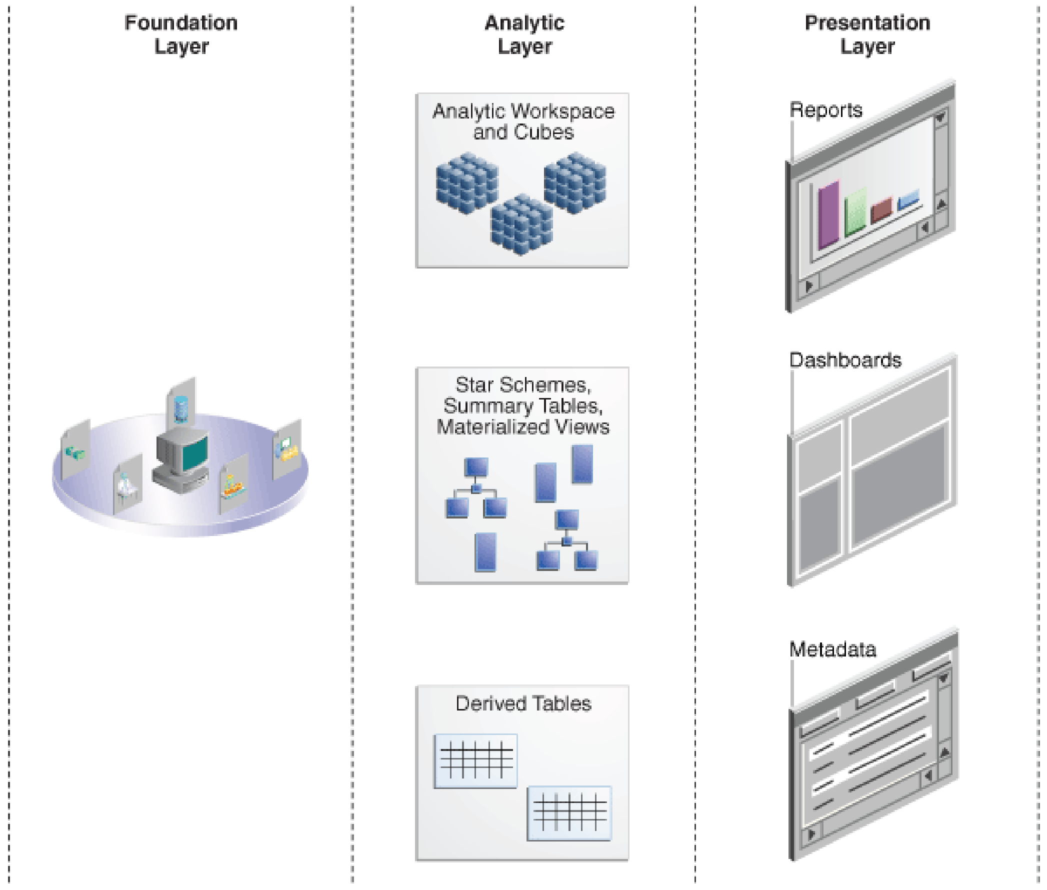
Industry Data Models

Oracle offers industry-specific pre-built data warehouse solutions that leverage the full set of Oracle Data Warehouse, ETL/ELT technologies, Business Intelligence and analytical products. These solutions provide standards-based underlying schema, database objects, metadata, analytical models, role-based dashboards and reports built specifically for a given industry. They create a data warehouse framework that is both adaptable and extensible, allowing you to maximize the capabilities of your data warehouse while accelerating the deployment of a comprehensive business intelligence strategy.

The pre-built, pre-tuned data warehouse solution sets allow you to quickly gain value from your data warehousing initiatives, support diverse analytical requirements, and assist in building future analytical applications. Fast, easy and predictable implementation reduces risks and enables you to achieve strategic value more rapidly by eliminating deployment delays and expenses associated with build-from-scratch or proprietary data warehouse solutions so you can start making better business decisions faster.

The normalized (Third Normal Form) foundation schema serves as a detailed and structured representation of the business, providing an integrated base for business information with fully defined tables and relationships. Automated data movement transforms and aggregates this data into the Analytical Layer and computes performance measures for you to fully understand what is happening in the enterprise. Advanced analysis and Data Mining models analyze and forecast trends, find correlations and commonalities, and predict outcomes. Finally, the Information Access layer provides robust sample reports and dashboards in order to view the measures and analytics. Sample reports are accessible from easy-to-use, role-specific dashboards via any standard Web browser. Role-based dashboards enable customers to tailor the information needed to enhance and improve their decision- making process. Metadata tools allow for lineage and impact analyses.

Figure 7–1 Industry Data Models



Oracle Airline Data Model

An airline's specific, pre-built data warehouse solution set that supports the data warehouse foundation and analytical needs of passenger service systems (includes reservation systems and departure control systems), global distribution system (GDS), loyalty management systems, and customer data warehouses.

This data warehouse solution provides detail transaction storage and advanced analysis for a full range of airline subject areas, including reservations, sales, operations, loyalty, and finance. Using reservation data, the data model enables detailed insight into passenger bookings by time period, fare class, and flight. It enhances understanding of channel performance looking at bookings, cancellations, and revenues through travel agency, OTA, ticket counter, call center, and web channels. The data model allows you to analyze passenger revenues by geography, time period, and flight. Finally it provides insights into loyalty program member activity through a variety of reports. The Oracle Airline data model fits the needs of large network carriers and low-cost carriers.

Oracle Communications Data Model

Conformance certified with the TM Forum's Information Framework (SID), Oracle Communications Data Model provides a Communications and Media specific pre-built data warehouse solution set that supports the data warehouse foundation and analytical needs of Communications Service Providers (CSPs).

With the Oracle Communications Data Model, CSPs can more easily collect and analyze customer data required for making decisions across customer, revenue, marketing, network, product and other business areas. With sophisticated trending and data mining capabilities built into the system, the reports provide greater business insights than ever. The solution also supports TMF Business Metrics, and has been certified with TM Forum's Business Metric Automation (BMA) initiative enabling CSPs to measure their performance vis-à-vis their competition and peers.

Oracle Retail Data Model

Based on the Association for Retail Technology Standards (ARTS), a pre-built retail-specific data warehouse solution set that supports the data warehouse foundation and analytical needs of retailers.

Oracle Retail Data Model provides the basis for operational reporting, detailed analysis, fraud detection and loss prevention, as well as forecasting, trend & and predictive analysis. With pre-built Oracle OLAP and Oracle Data Mining models geared toward specific retail business problems, Oracle Retail Data Model transforms retail data into insightful, actionable information.

Data Mining

Oracle Data Mining provides users SQL access to high performance algorithms in the database. ODM can mine tables, views, star schemas, transactional and unstructured data to represent a complete 360 degree view of the customer for better customer understanding. ODM leverages database parallelism, Oracle Real Application Clusters (RAC) and other database features for fast in-database model building and scoring, thus avoiding time consuming data movement. Exadata smart scans execute ODM models at the storage tier, while parallel query, Exadata and cursors support ODM in Decision Support System (DSS) and OLTP environments (e.g., real-time call centers).

Oracle Data Miner, a graphical user interface extension to SQL Developer, helps data analysts explore their data, build and evaluate data mining models, and share analytical workflows for churn prediction, response modeling segmentation, fraud and anomaly detection, sentiment analysis and market basket analysis problems. Oracle Data Miner generates SQL and PL/SQL code to accelerate model deployment. Developers can leverage the code to build predictive applications that automate knowledge discovery.

What is Data Mining?

Data mining is the practice of automatically searching large stores of data to discover patterns and trends that go beyond simple analysis. Data mining uses sophisticated mathematical algorithms to segment the data and evaluate the probability of future events. Data mining is also known as Knowledge Discovery in Data (KDD).

The key properties of data mining are:

- Automatic discovery of patterns
- Prediction of likely outcomes

- Creation of actionable information
- Focus on large data sets and databases

Data mining can answer questions that cannot be addressed through simple query and reporting techniques.

Data Mining and Data Warehousing

Data can be mined whether it is stored in flat files, spreadsheets, database tables, or some other storage format. The important criteria for the data is not the storage format, but its applicability to the problem to be solved.

Proper data cleansing and preparation are very important for data mining, and a data warehouse can facilitate these activities. However, a data warehouse will be of no use if it does not contain the data you need to solve your problem. With one exception, Oracle Data Mining requires a single-record case data presentation. This means that the data for each record (case) must be specified within a single row. The exception is the data accepted by the a priori algorithm for the calculation of association rules. The data for association rules may be presented as transactions (market-basket data), with the data for each case specified over multiple rows.

Asking the Right Questions

Data mining does not automatically discover information without guidance. The patterns you find through data mining will be very different depending on how you formulate the problem.

To obtain meaningful results, you must learn how to ask the right questions. For example, rather than trying to learn how to "improve the response to a direct mail solicitation," you might try to find the characteristics of people who have responded to your solicitations in the past.

Data Mining in the Database Kernel

Oracle Data Mining provides comprehensive, state-of-the-art data mining functionality within Oracle Database.

Oracle Data Mining is implemented in the Oracle Database kernel, and mining models are first class database objects. Oracle Data Mining processes use built-in features of Oracle Database to maximize scalability and make efficient use of system resources.

The advantages of mining in the kernel are:

- No Data Movement. Some data mining products require that the data be exported from a corporate database and converted to a specialized format for mining. With Oracle Data Mining, no data movement or conversion is needed. This makes the entire mining process less complex, time-consuming, and error-prone.
- Security. Your data is protected by the extensive security mechanisms of Oracle Database. Moreover, specific database privileges are needed for different data mining activities. Only users with the appropriate privileges can score (apply) mining models.
- Data Preparation and Administration. Most data must be cleansed, filtered, normalized, sampled, and transformed in various ways before it can be mined. Up to 80% of the effort in a data mining project is often devoted to data preparation. Oracle Data Mining can automatically manage key steps in the data preparation process. Additionally, Oracle Database provides extensive administrative tools for preparing and managing data.

- Ease of Data Refresh. Mining processes within Oracle Database have ready access to refreshed data. Oracle Data Mining can easily deliver mining results based on current data, thereby maximizing its timeliness and relevance.
- Oracle Database Analytics. Oracle Database offers many features for advanced analytics and business intelligence. Oracle Data Mining can easily be integrated with other analytical features of the database, such as statistical analysis and OLAP.
- Oracle Technology Stack. You can take advantage of all aspects of Oracle's technology stack to integrate data mining within a larger framework for business intelligence or scientific inquiry.
- Domain Environment. Data mining models have to be built, tested, validated, managed, and deployed in their appropriate application domain environments. Data mining results may need to be post-processed as part of domain specific computations (for example, calculating estimated risks and response probabilities) and then stored into permanent repositories or data warehouses. With Oracle Data Mining, the pre- and post-mining activities can all be accomplished within the same environment.
- Application Programming Interfaces. PL/SQL API and SQL language operators provide direct access to Oracle Data Mining functionality in Oracle Database.

OLAP

Oracle OLAP delivers advanced multidimensional analytic capabilities within Oracle Database 12*c*. It is designed to provide excellent query performance, fast incremental updates of data sets, efficient management of summary data and rich analytic content.

Oracle OLAP makes it easy to produce analytic measures, including time-series calculations, financial models, forecasts, allocations, regressions, and more. Hundreds of analytic functions can be easily combined in custom functions to solve nearly any analytic calculation requirement. Oracle OLAP cubes are represented using a star schema design: dimension views form a constellation around the cube (or fact) view. This standard representation of OLAP data makes it easy for any SQL-based tool or application to leverage the power of Oracle OLAP. As an embedded component of Oracle Database 12*c*, Oracle OLAP benefits from the manageability, scalability, high availability and security features that make the Oracle Database the market leading information platform:

- Oracle OLAP is embedded in the Oracle Database kernel and runs in the same database process
- OLAP cubes are secured by standard Oracle Database security features (e.g. Virtual Private Databases)
- Oracle OLAP fully leverages scalability and high availability features such as Real Application Clusters and Automatic Storage Management.
- Queries against OLAP cubes may be combined with other types of data managed by the Oracle Database - including spatial, XML, documents, and so on.

From a total cost of ownership perspective, Oracle OLAP represents an incremental investment in the Oracle Database you already own. Oracle OLAP does not require separate server computers. It allows you to leverage your current capital and intellectual investment and to continue to use your existing SQL-based applications. Using Oracle OLAP enables these applications to simply become smarter and faster.

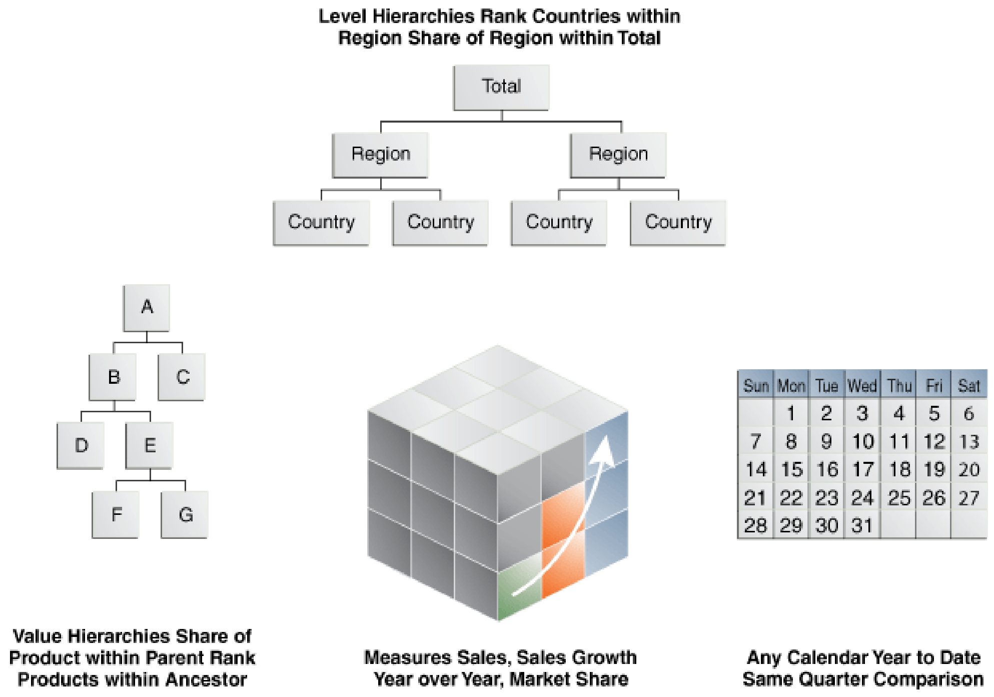
Business Model in the Oracle Database

Oracle dimensions and cubes are based on a business model. That is, the business users view of the data. The business model organizes data, making it easy to extend the data with sophisticated calculations and making it very easy to query. Fact tables are transformed into cubes. The cube defines measures and the relationship between measures and dimensions. It understands how data should be aggregated. Dimensions include full featured hierarchies and attributes.

The model makes it very easy to define new calculated measures. For example, using OLAP expressions that leverage hierarchical relationships within a dimension it is very easy to define measures such as the share of sales for a product within its parent (value hierarchies) or the share of sales for a country within its region (level hierarchies). OLAP understands time, making it very easy to define time series calculations. You can create your own calendars, for example fiscal, retail and manufacturing calendars that reflect how you run your business and create time series calculations based on those calendars.

When calculations are defined in the cube, they do not need to be defined within the query. This makes querying the cube very simple, even when it contains complex calculations. The cube is the perfect foundation for custom BI applications and making simple BI tools smart.

Figure 7–2 OLAP Hierarchies


Level Hierarchies Rank Countries within Region Share of Region within Total
Total
Region
Region
Country
Country
Country
Country
A
B
C
D
E
F
G
Sun Mon Tue Wed Thu Fri Sat
Value Hierarchies Share of Product within Parent Rank Products within Ancestor
Measures Sales, Sales Growth Year over Year, Market Share
Any Calendar Year to Date Same Quarter Comparison

Oracle dimensions and cubes organize data into a business model for easy calculation definition and query.

Oracle OLAP in the Oracle Database

Oracle OLAP is an ideal technology to enhance the performance layer of the data warehouse. A single OLAP cube benefits multiple applications by simultaneously providing several access paths:

- Cubes and dimensions can be queried directly using SQL and MDX (MDX requires third party MDX Provider from Simba Technologies Inc.)
- Cubes can be queried indirectly when deployed as a cube-organized materialized view (Cube materialized view). Cube materialized views augment your summary management strategy. Automatic query rewrite redirects summary queries against a fact table to the cube - transparently improving query performance.
- Cubes can be queried in a multidimensional style using the OLAP API.

Direct SQL Query of the Cube

A cube, including summary data and calculations, can be queried directly by SQL-based applications. Dimension views, hierarchy views and cube views are provided for this purpose. The analytic content embedded in the cube and revealed through views allows even the most basic SQL-based reporting tools to deliver high-end OLAP content.

Cubes present both summary data and analytic measures as fully computed. Calculation rules defined in the cube are used to ensure the correct results. Defining this metadata in the cube greatly simplifies the SQL required to query the cube because applications are not burdened with the need to understand how the data is computed - they simply select the information that they need. For example:

- Analytic calculations are exposed as additional fact columns in the cube view. The application queries sophisticated calculations by simply selecting columns from the cube view.
- All summary data is available in the cube view. This is especially useful when aggregation rules are complex (for example, headcount sums over organization but aggregates using last over time). The application simply selects data at the correct level of summarization; it does not need to include aggregation functions and `GROUP BY` in the query.
- Partitioned outer joins are automatically and efficiently executed within the cube as needed by analytic calculations. For example, time series functions such as leads, lags and parallel periods are automatically "densified" in the cube, thus eliminating the need for complicated outer join syntax in SQL queries.

Summary Management With Cube-Organized Materialized Views

SQL-based applications can also use the cube to enhance a summary management solution. This is especially helpful when users' queries exhibit ad hoc patterns. In this use of the cube, summary data is managed within the cube and revealed to the application as a cube-organized materialized view. The application continues to query the detailed level data in relational tables, expressing queries for summary level data with an aggregation function and `GROUP BY`. The automatic query rewrite feature of the Oracle Database rewrites the query to the cube-organized materialized view. The result is improved query performance without any changes to the application.

The cube-organized materialized view is managed using standard materialized view refresh policies and procedures. Incremental updates to the cube will be performed using materialized view logs or partition change tracking.

Improving the Business Intelligence Solutions You Already Own

The vast majority of BI solutions query relational schemas that have been implemented using a star schema design. Oracle OLAP has been designed to be compatible with these BI solutions. Because OLAP cubes are exposed using a traditional star design, BI solutions can easily query cubes and gain access to their rich analytic content. BI administrators experience tremendous productivity gains from the automatic generation of Oracle Business Intelligence Enterprise Edition metadata over cubes. End users benefit from excellent query performance and enhanced content in business intelligence applications. Using an optional MDX Provider for Oracle OLAP, business users can query OLAP cubes using Excel pivot tables.

Centralized Management of Key Business Intelligence Assets

Metadata (e.g. dimensions, levels, hierarchies) and calculation rules are key assets of a BI solution and the organization that owns it. Oracle OLAP allows organizations to manage these assets within the Oracle Database and share them among any number of BI and reporting tools. Applications simply query the Oracle data dictionary views to discover the properties of the business model. This management strategy allows the Oracle Database to be a single version of the truth.

Managing Cubes

Oracle cubes are designed using Analytic Workspace Manager, a graphical administrative tool that is designed specifically to manage OLAP cubes and dimensions. Using Analytic Workspace Manager, the DBA or application developer designs dimensions, hierarchies, cubes, measures, aggregation rules, and security policies for cubes and dimensions. The OLAP expression language may be used to design custom analytic functions. Application developers have the option of using the OLAP API to create and manage cubes. The API naturally reflects the logical business model - enabling Java-based applications to easily define and query OLAP cubes.

One a day to day basic, the loading and processing of dimensions and cubes is managed by the DBA or application owning using either Analytic Workspace Manager or PL/SQL programs. Cube processing is typically a step in the overall ELT process.

About R Enterprise

Oracle R Enterprise is the most integrated and complete suite of advanced analytics available to R users for enterprise environments. Designed for problems involving large amounts of data, it gives R programmers tight integration with the database for rapid development of sophisticated analytical methods and unprecedented support for big data analytics in Oracle Database and Oracle Big Data Appliance environments. Data analysts can use R to analyze billion row data sets, test a variety of sophisticated numerical techniques in seconds, thus making iterative, speed of thought numerical analysis on big data practical and feasible. R users can develop, refine and deploy R scripts that leverage the parallelism and scalability of the database for data preparation, model development and model deployment. R users can develop and deploy R scripts in one step-without having to learn SQL. Because it runs as an embedded component of the database, ORE can run any CRAN open source R package either by ORE R to SQL function pushdown for native in-database execution or via embedded R mode where the database manages the data flows to multiple R engines. Oracle R Enterprise exposes Oracle Data Mining's high-performance native SQL based algorithms through R. DBAs can productionize R scripts for in-database execution eliminating model translation to other languages such as SQL or C. Oracle also integrates R with Hadoop running on Oracle's Big Data Appliance for complex computations performed on data with low value information density such as social network, sensor and web log data managed in Hadoop servers prior to ingestion into the database for further analysis.

R is an open source statistical programming language and environment. For information about R, see the R Project for Statistical Computing at http://www.r-project.org.

R provides an environment for statistical computing, including:

- An easy-to-use language
- A powerful graphical environment for visualization
- Many out-of-the-box statistical techniques
- R packages (An R package is a set of related functions, help files, and data files; currently, there are more than 3340 available packages.).
- The R Console graphical user interface for analyzing data interactively

R's rapid adoption has earned it a reputation as a new statistical software standard.

Oracle R Enterprise is a component of the Oracle Advanced Analytics Option of Oracle Database Enterprise Edition. Oracle R Enterprise allows users to perform statistical analysis on data stored in tables in an Oracle Database. Oracle R Enterprise has these components:

- The Oracle R Enterprise R transparency layer. The transparency layer is a collection of packages that support mapping of R data types to Oracle Database objects and generate SQL transparently in response to R expressions on mapped data types. The transparency layer allows an R user to directly interact with database-resident data using R language constructs. This enables R users to work with data too large to fit into the memory of a user's desktop system.
- The Oracle statistics engine, a collection of statistical functions and procedures corresponding to commonly-used statistical libraries. The statistics engine packages execute in Oracle Database.
- SQL extensions supporting R engine execution through the database on the database server. These SQL extensions enable productizing R scripts, that is, running R scripts in a lights-out mode.
- Oracle R Connector for Hadoop is an R package executing MapReduce jobs that enables R users to directly work with an Oracle Hadoop cluster executing computations written in the R language and working on data resident in HDFS, Oracle database or local files.

R Enterprise Architecture

Oracle R Enterprise has these three components including the connector for Hadoop.

1. The Client R Engine is installed on Microsoft Windows or Oracle Linux. The client engine is a collection of R packages that allows you to connect to an Oracle Database and to interact with data in that database. The ORCH package allows you to connect to an Oracle Hadoop cluster and interact with data in HDFS files; the package also allows the execution of MapReduce jobs.

   You can use any R commands from the Client. In addition, the client supplies these functions:

   - The R SQL Transparency framework intercepts R functions for scalable in-database execution
   - Functions intercept data transforms, statistical functions, and Oracle R Enterprise-specific functions.
   - Interactive display of graphical results and flow control as in open source R

- Submission of R closures (functions) for execution in the Oracle Database

2. The Server is a collection of PL/SQL procedures and libraries that augment Oracle Database with the capabilities required to support an Oracle R Enterprise client. The R engine is also installed on Oracle Database to supported embedded R execution. Oracle Database spawns R engines, which can provide data parallelism.

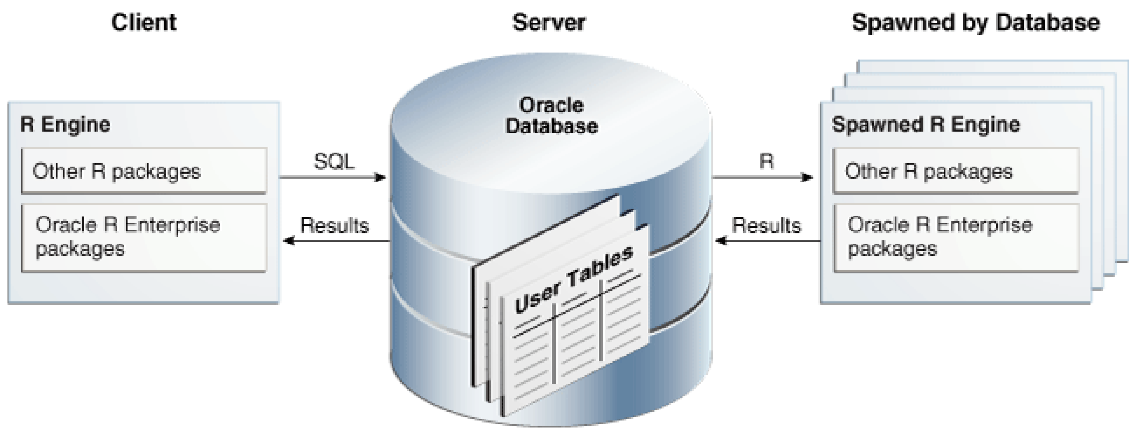
   The Oracle R Enterprise Database engine provides this functionality:

   - Scale to large data sets
   - Access to tables, views, and external tables in the database, as well as those accessible through database links
   - Use SQL query parallel execution
   - Use in-database statistical and data mining functionality

3. R Engines spawned by Oracle Database are spawned to support database-managed parallelism; provide lights-out scheduled execution of R scripts, that is, scheduling or triggering R scripts packaged inside a PL/SQL or SQL query. Oracle R Enterprise provides efficient transfer to and from the spawned engines. Embedded R execution can be used to emulate MapReduce style programming.

4. The Oracle R Connector for Hadoop is an R package that allows an ORE client to interact with and execute MapReduce jobs on the Oracle Hadoop cluster.

Figure 7–3 R Enterprise Architecture



OBI Enterprise Edition

Oracle Business Intelligence Enterprise Edition 11*g* delivers reporting, ad hoc query and analysis, OLAP, dashboard, and scorecard functionality with a rich end user experience that includes visualization, collaboration, alerts and notifications, search and mobile access.

OBIEE 11g integrates with all popular data sources, ETL tools, business applications, application servers, security infrastructure, portal technology as well as any ODBC compliant third party analytical tool. OBIEE 11g accesses data from multiple heterogeneous sources-including popular relational and multidimensional data sources and major ERP and CRM applications from Oracle and other sources.

OBIEE 11*g* is based on an architecturally integrated technology foundation built on an open, standards based service oriented architecture. OBIEE 11*g* features a Common

Enterprise Information model, common security model and a common configuration, deployment and systems management framework.

The rest of this section highlights key functionality of OBIEE, focusing on business value.

- Interactive Dashboards

  OBIEE 11*g* provides a fully interactive collection of dashboards and reports with a rich variety of visualizations. The dashboards provide users with information filtered and personalized for their identity, function, or role based on predefined security rules. The rich, interactive user interface makes the presentation of data intuitive, relevant, and easy to understand. In addition, guided navigation and alerts drive the business user to greater insight and action.

- Ad hoc Analysis and Interactive Reporting

  Providing business users with full ad hoc query and analysis capability, users of Oracle Business Intelligence can create new analyses from scratch or modify existing analyses in dashboard pages. To free business users from data structure complexity, the metadata layer of Oracle Business Intelligence offers a logical view of metrics, hierarchies, and calculations expressed in understandable concepts. Business users do not need to understand physical data storage to combine data from multiple enterprise information sources.

- Enterprise Reporting

  Delivered via Oracle Business Intelligence Publisher, this functionality allows the creation of highly formatted templates, reports, and documents such as flash reports, checks, and more. It is the most efficient, most scalable reporting solution available for complex and distributed environments. Tightly integrated with the OBIEE 11*g* platform; BI Publisher can also be deployed as a separate product.

- Proactive Detection and Alerts

  OBIEE 11*g* features a powerful, near-real-time, multi-step alert engine that can trigger workflows based on business events and notify stakeholders via their preferred medium and channel. This means field sales representatives can receive a short message service alert on their cell phone, logistics managers get a PDF attachment via e-mail, and financial analysts obtain the report as a Microsoft Excel spreadsheet saved to their shared corporate file system.

- Actionable Intelligence

  The OBIEE 11*g* Action Framework turns insights into actions by providing the ability to invoke business processes from within the business intelligence dashboards and reports. This is made possible by the integration of business process management technologies within the business intelligence platform. Invoked actions may include initiating a business process, a web service, or simply calling another dashboard or report.

- Spatial Intelligence via Map-Based Visualizations

  Map Views allows users to visualize their analytics data using maps, thus bringing the intuitiveness of spatial visualizations to the world of business intelligence. The maps are fully interactive and data on the maps can be visualized using numerous formatting options including color fills, variable-sized markers, custom image markers, percentile binning, value binning, and continuous color-fill options.

- Scorecard and Strategy Management

  Scorecard and Strategy Management extends OBIEE 11*g* with capabilities aimed at communicating strategic goals across the organization and monitoring progress over time. Scorecard and Strategy Management provides capabilities to establish specific goals, define how to measure success, and communicate that information down the entire organization.

- Server-Based Query, Reporting, and Analysis

  As the foundation of OBIEE 11*g*, the Oracle Business Intelligence Server generates queries optimized for each data source, appropriately aggregates them, and presents the results to users within a familiar Web browser via easy-to-use dashboards and reports.

A flexible, enterprise metadata layer spans all your underlying data sources-including flat files, databases, packaged applications, and more   Report authors can select the items they want in their report and the Oracle Business Intelligence Server will collect and aggregate the information-even if it exists in disparate data sources. With larger user populations, many queries will have similar content and the Oracle Business Intelligence Server can intelligently reuse previous query results. Queries might also be scheduled to be pre-run so the results are available when the user opens the dashboard.

The Oracle Business Intelligence Server also includes parallel query execution engines, memory management, and high-throughput data connectivity adapters to allow highly efficient data sourcing and aggregation that minimize data retrieval time. This highly scalable platform with clustering and caching capabilities is the heart of what drives the other suite components. Multiple servers can be clustered to provide session replication and automatic failover capabilities. Powered by a centralized, single, IT controlled metadata layer; Oracle BI Server features easy change management-for example, a single-click switch from a test system to production.

Index