

Sound Theory Good Practice

***Designing and Building Data
Warehouses with Oracle 10g,
11g and 11gR2***

Oracle Data Warehousing Fundamentals

Objective: Understand sound theory and use good practice in the design and creation of a Data Warehouse using features provided by Oracle 10g and 11g and 11g R2

Agenda

- Share a Common Set of Data Warehousing Terms
- Examine Business Intelligence, OLAP, and Data Mining
- Learn to Distinguish the Components of a Data Warehouse
- Discuss Data Modeling
- Understand Oracle's Key Data Warehousing Technologies

Agenda

- Dimensions
- Facts
- Storage, Performance
- Compression
- Query Rewrite
- Materialized Views
- Partitioning
- SQL Analytics
- Aggregate Functions
- SQLDeveloper
- Data Modeler
- OBIEE

Analysis, Design and Architecture

Data Warehousing Defined

Two Main Schools of Thought

William H. Inmon (Bill), the “Father of Data Warehousing”

A warehouse is a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process



Kimball's Definition

Ralph Kimball, "The Star Schema"

A copy of transaction data specifically structured for query and analysis

Business Processes are identified and organized into Data Marts

Synthesis

What do these two have in common?

What they can agree upon must be particularly important.

First area of agreement: It is transaction data that has been restructured.

Second area of agreement: The purpose is to support analysis of business data.

Oracle's Recommendation

Reflects elements from both, Oracle's definition leans toward the Star Schema

A data warehouse is a relational database that is designed for query and analysis rather than for transaction processing.

It usually contains historical data derived from transaction data, but it can include data from other sources.

It separates analysis workload from transaction workload and enables an organization to consolidate data from several sources.

***Oracle recommends that
you choose a star schema
unless you have a clear
reason not to***



Three Data Warehouse Architectures

The architecture of your data warehouse often reflects the stage of your organization's DW evolution

Basic

The metadata and raw data of a traditional OLTP system is present. A basic data warehouse also can include summary data. Summaries pre-compute long operations in advance. A summary in an Oracle database is called a materialized view.

With a Staging Area

To clean and process your operational data programmatically, most data warehouses use a staging area instead. This simplifies building summaries and management.

With a Staging Area and Data Marts

Adding data marts, which are systems designed for a particular line of business. For example, purchasing, sales, and inventories might be separated.



The OLTP Query Plan

- In the transactional database, the interface is designed to supply access to a single order or transaction
- When interacting with these interfaces the user's interactions with interface are creating a series of queries designed to retrieve a single transaction
- The entry point into the database is usually the index of the unique values of the transaction
- Even if the user's initial queries are based upon the "LIKE" type searches on text information, these return a short list of items from which an individual item is selected
- In this type of situation a 3NF schema supports the most efficient query path to a single order
- Only recent information is maintained online

Query Plans and the Star Schema

- Database programmers have a unique perspective on the impact of these two approaches have on data retrieval
- The query plans for a transactional database will differ from those of data warehouse, the differences reflecting the different purposes
- The design used in each schema supports the different purposes

The Data Warehouse Query Path

- Questions to be addressed involve sums and counts
- The data is organized to provide aggregations and utilize a “grain” that reflects the way the data is used by business analysts
- The query path into the data begins with a dimension
- The dimension includes indexed foreign keys
- Using the foreign key values select a much smaller, possibly pre-aggregated, portion of the larger fact table
- This is the most efficient path when collecting “clumps” of data from a much larger set

Identify the Dimensions

- Identify those characteristics of the business process that are likely to be of most interest to business users
- Keep in mind the dimensions will tie the data marts together into a data warehouse
- Define the hierarchies that will make up the dimensions themselves
- Almost all data warehouses have a date dimension and the date dimension is a hierarchy
- Every row in every dimension uses a surrogate key as its primary key. Natural keys are never used in dimensions.

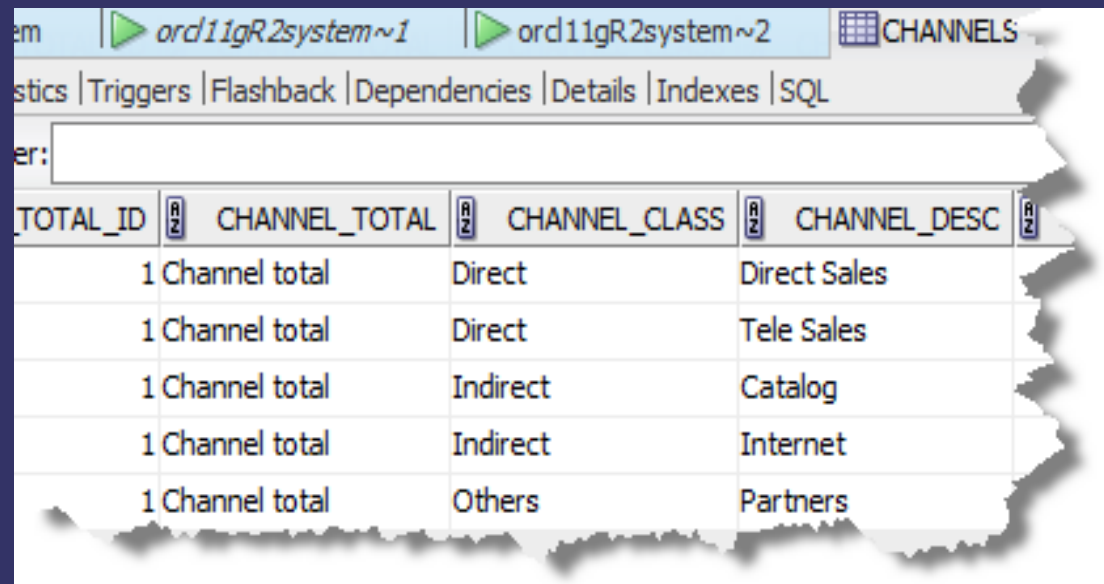
Hierarchies in Dimensions

Many Dimensions have hierarchies or roll ups as part of their structure

Repeating values are used – Sorry 3NF folks!

The repeating text values are used, as these are understandable to business users

The Total Column contains a single value



TOTAL_ID	CHANNEL_TOTAL	CHANNEL_CLASS	CHANNEL_DESC
1	Channel total	Direct	Direct Sales
1	Channel total	Direct	Tele Sales
1	Channel total	Indirect	Catalog
1	Channel total	Indirect	Internet
1	Channel total	Others	Partners



Just the Facts, Ma'am

- Three basic types of Fact tables
- Transaction grain based additive fact table
- Periodic Snapshot table to describe processes with regular balances
 - Semi additive (Day, Month, Year)
 - General Ledger an example
- Accumulating Snapshot to track a series of events
- Often two or three of these types are used in conjunction with each other

Transaction Fact Table

- Series of additive facts
 - Numbers can be rolled or accumulated
 - Even when aggregated accurate
- May include non-additive numbers used for counts or distinct counts
 - Member ID
 - Transaction ID
- Counted items called “Degenerate Dimensions”

Periodic Snapshot

- When periodic balances are measured
 - Inventory Counts
 - Daily count of Accounts
- Semi-Additive numbers
 - It may make sense to add up inventory values to determine the balance at the end of the month, adding the monthly balances does not yield a yearly balance

Accumulating Snapshot

- When a process is a series of events
 - Mortgage loan is a complicated series of events
 - Fact table would be a set of foreign keys to the date dimension
 - As each set is completed, the date reference is added to the appropriate column
- Must be a know series of sets
- Only type of fact table that is regularly updated
- Often a “factless” fact table, as it includes no additive measures

Dimension Details

- When it is okay to build “snowflakes”, that is dimensions with parent and child tables?
- Snowflakes are almost always a bad idea
- If you find yourself with a potential snowflake, as yourself, do you really have two dimensions, each of which should have direct relationship to the fact table
- Layers of snowflakes can – in the worst case – confuse the optimizer to develop a query plan that begins at the fact table
- The only exception – low cardinality columns in already complex dimensions
 - These are dimension outriggers
 - Too many outriggers is a sign of trouble

Slowly Changing Dimensions (SCD)

- Data changes – so how do you handle changes in a data warehouse?
- In fact tables, changing data generally means new data to be added
- In dimensions the options are more complex
- The most common example are the addresses of customers
 - If you change the zip code in an address table, the counts on your geographic analysis might change
 - What if you want to track these changes over time?
- Kimball has developed three strategies for SCDs

Changing Dimensions – Types

- The Type 1 change in a dimension is a straight change
- Do not confuse this type of change with error correction. If your dimension data has an error, correct it
- With a Type Two change, a new row is added to the dimension with the new value
- The old value remains, tied to existing values in the fact tables -- all new data added to the fact tables
- In Type 3 columns added to hold previous and current values
- Example might be 1st quarter, 2nd quarter, 3rd quarter and 4th quarter columns

rrrrrr...Ragged Hierarchies

What about hierarchies that have a variable number of levels

Example: Organization Chart

A recursive structure leads to complex queries

Connect by Start with only allows for one table

In this case we use an Associative table – called in dimensional modeling a bridge table

Add columns for Top and Bottom level flags

Or, multiple hierarchies within a single dimension



Junk Dimensions

Fact tables will sometimes contain a flags, low cardinality codes sets or other odd data fields

Move these into a dimension and build a series of bit mapped indexes on these columns

This is a *Junk Dimension*

Each unique set of values is a row in the dimension

Scanning a row for each of large number of possible combinations is better than a full table scan of a fact table

A junk dimension with 16 fields only has a few thousand possible combinations,

Accessing the junk dimension becomes a more efficient entry point for the query than going directly to the fact table



Role Playing Dimensions

- The same set of values may be used for a number of purposes in a data warehouse
 - Dates: order date, back order date, ship date...
 - Addresses: customer address, vendor address...
- Rather than create multiple tables with the same value duplicated, create views with appropriate names
- These views are Role Playing dimensions

Extract, Transform and Load

Daily ETL Operations

- Loads and transformations must be scheduled and processed in a specific order
- Oracle is not an ETL tool and does not provide a complete solution for ETL and Daily Operations
- But, Oracle provides a rich set of capabilities used by ETL tools and customized solutions
 - Success or failure of the operation or parts of it, must be tracked and subsequent ETL tools such as Oracle Scheduler
 - Transporting data between Oracle databases
 - Transforming large volumes of data
 - Quickly loading new data into a data warehouse
 -

The Evolution of ETL

- The data warehouse is a living system -- sources and targets might change
- Changes must be maintained and tracked through the lifespan of the system without overwriting or deleting the old ETL process flow information.
- To build and keep a level of trust about the information in the warehouse, in an ideal case, the process flow of each individual record in the warehouse can be reconstructed at any point in time in the future.
- compromises occur
- Carefully decide where the compromises will occur

Oracle's Staging and ETL Recommendations

- The staging layer enables the speedy extraction, transformation and loading (ETL) of data from your operational systems into the data warehouse without distributing business users.
- This layer with the complex data transformation and data-quality processing
- It is segregated from the “live” data warehouse
 - One approach for the staging layer is to have it be an identical schema to the operational system(s)
 - Typically some structural changes, such as range partitioning
 - Another philosophy is to do all data transformations “on the fly” as data is extracted
- This is ETL vs. ELT argument

A Stage Layer and Direct Loads Share the Same Goal...

...Load data into the warehouse in the most efficient and expedient manner

- Oracle offers several data loading options
 - External table or SQL*Loader
 - Oracle Data Pump (import & export)
 - Change Data Capture or Oracle streams for trickle feeds
 - Oracle Transparent Gateways
 - Home grown SQL and PL/SQL
 - Transportable Tablespaces



Which approach for External Data?

- Consider your source data and the format of data you receive
- If you are loading from flat files into Oracle you have two options,
 - SQL*Loader
 - External Tables

Oracle STRONGLY recommends using external tables

- When SQL*Loader is used to load data in parallel, the data is loaded into temporary extents.
- Only when the transaction is committed are the temporary extents merged into the actual table
- Any existing space in partially full extents in the table will be skipped
- For highly partitioned tables this could lead to wasted space



External Tables

- An external table provides access data in external sources (flat file) as if it were in a table in the database
- This means that external files can be queried directly and in parallel using the full power of SQL, PL/SQL, and Java
- An external table is created using the standard create table syntax except for the addition of the ORGANIZATION EXTERNAL clause

Transportable Tablespaces

- Oracle transportable tablespaces are the fastest way for moving large volumes of data between two Oracle databases
- Transportable tablespaces entirely bypass the unload and reload of data from files
- Transportable tablespaces, contain table data, indexes, and most other Oracle database objects
- Like import and export, transportable tablespaces transport metadata, too
- Limitations of transportable tablespaces: Source and target systems must be running Oracle8i (or higher); must use the same character set; and prior to Oracle Database 10g, both must run on the same operating system
- The most common applications of transportable tablespaces in data warehouses are in moving data from a staging database to a data warehouse, or in moving data from a data warehouse to a data mart

SQL and PL/SQL

- All of the usual recommendations for use of packages and procedures apply
- Use CTAS (Create Table as Select) or IAS (Insert Append Select)
- Use CTAS and IAS in conjunction with Parallel Processing (Tables and Hints)
- No stale statistics – “The Optimizer is a stupid head”

```
DBMS_STATS.COPY_TABLE_STATS (ownname      => 'DW_XXX',  
                             tabname      => 'XXXXXX_FACT',  
                             srcpartname  => source_part_name,  
                             dstpartname  => dest_part_name,  
                             force        => TRUE);
```

- Use partition swapping when ever possible in loading partitioned tables



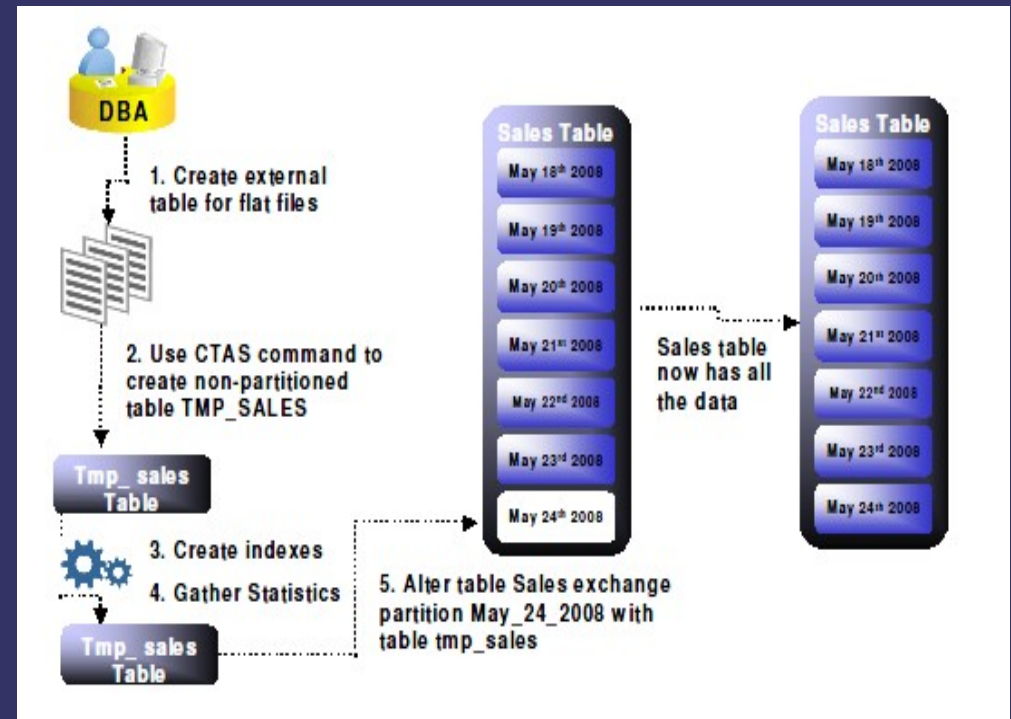
Partition Exchange Loads

- Oracle recommended that the larger tables in a data warehouse be partitioned
- A feature of partitioning is the ability to load data using the exchange partition command
- The exchange partition command allows you to swap the data in a non-partitioned table into a particular partition in your partitioned table
- The command does not physically move data it simply updates the data dictionary to reset a pointer from the partition to the table and vice versa
- Because there is no physical movement of data, this exchange does not generate redo and undo
- This makes for a sub-second operation
- Far less impact on performance



Partition Exchange Load Steps

1. Create external table for the flat file data coming from the online system
2. Using a CTAS (Create Table As Select) statement, create a non-partitioned table called tmp_sales that has the same column structure as Sales table
3. Build any indexes that are on the Sales table on the tmp_sales table
4. Gather optimizer statistics on the tmp_sales table
5. Issue the exchange partition command



```
Alter table Sales exchange partition p2 with  
table tmp_sales including indexes without valid-  
ation;
```

Compress During Loads?

- In 11g onwards the new feature, OLTP Table Compression allows data to be compressed during all types of data manipulation operations
- includes conventional DML such as INSERT and UPDATE
- Use of compression will add some additional CPU overhead to compress when loading and decompress during query execution
- Normally, performance gain will easily outweigh the cost of compression

Oracle strongly recommends compressing your data

Tips for the Staging Layer

- Use external tables
- Load using parallel DML statements
CTAS or IAS
- Use data compression
- Considering range partitioning fact
table to enable partition exchange
loads

The Data Warehouse in a Relational Database

A significant portion of Oracle's support of Data Warehousing is in the form of special types of physical objects and supporting functionality built into the Oracle database

- Dimensions
- Partitioning
- Materialized Views and Query Rewrite
- SQL for Aggregation in Data Warehouses
- OLAP Functionality including Cubes and Multidimensional Objects
- SQL Modeling Clause
- SQL for Analysis and Reporting
- Parallel Execution



What is in the Dimension Object?

- Data analysis typically starts at higher levels in the dimensional hierarchy and gradually drills down if the situation warrants such analysis.
- Dimensional modeling, is worth the time creating because they help query rewrite perform more complex types of rewrite
- Dimensions are also beneficial to certain types of materialized view refresh operations
- Dimensions also inform the SQL Access Advisor.

Creating Dimensions

- Do not create dimensions in any schema that does not fully satisfy dimensional relationships
- Begin by creating the dimension tables containing the dimension data
- Identify the hierarchies of dimensions – this hierarchical information will be stored in the database object dimension.
- The dimension is created using either the CREATE DIMENSION statement or the Dimension Wizard in Oracle Enterprise Manager

The Dimension Object

- Dimensions are all about Business User data navigation
- The database object dimension helps to organize and group dimensional information into hierarchies
- This represents natural 1:n relationships between columns or column groups (the levels of a hierarchy) that cannot be represented with constraint conditions.
- Going up a level in the hierarchy is called rolling up the data and going down a level in the hierarchy is called drilling down the data

Examples:

Within the time dimension, months roll up to quarters, quarters roll up to years, and years roll up to all years.

Within the product dimension, products roll up to subcategories, subcategories roll up to categories, and categories roll up to all products.



Creating Dimensions

- Within the CREATE DIMENSION statement, the LEVEL clause identifies the names of the dimension levels
- To skip NULL levels in a dimension, use the SKIP WHEN NULL clause
- Hierarchical integrity is necessary for the correct operation of management functions for materialized views that include aggregates
- Dimensions may be based on multiple normalized tables, but this is not recommended

Creating Levels

- Create the dimension products_dim, which contains levels
 - product, subcategory, and category
- Each level in the dimension must correspond to one or more columns in a table in the database.
- Level product is identified by the column prod_id in the products table
- Level subcategory is identified by a column called prod_subcategory the same table.

```
CREATE DIMENSION products_dim
    LEVEL product IS (products.prod_id)
    LEVEL subcategory IS (products.prod_subcategory)
    LEVEL category IS (products.prod_category) ...
```



The Hierarchy Clause

- The next step is to declare the relationship between the levels with the HIERARCHY statement and give that hierarchy a name
- A hierarchical relationship is a functional dependency from one level of a hierarchy to the next level in the hierarchy
- Using the level names defined previously, the CHILD OF relationship denotes that each child's level value is associated with one and only one parent level value. The following statement declares a hierarchy prod_rollup and defines the relationship between products, subcategory, and category:
- the HIERARCHY clause adds the The 1:n hierarchical relationships

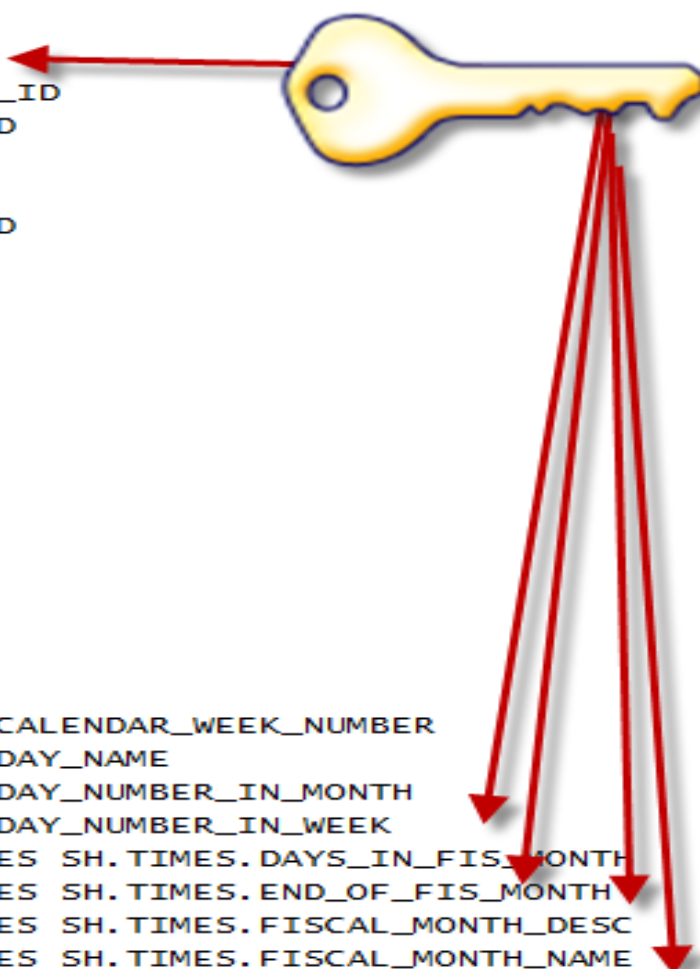
```
HIERARCHY prod_rollup (product      CHILD OF
                        subcategory  CHILD OF
                        category)
```



Attributes of Dimensions

- Dimensions include 1:1 attribute relationships between the hierarchy levels and their dependent, determined attributes

```
DIMENSION SH.TIMES_DIM
LEVEL DAY IS SH.TIMES.TIME_ID
LEVEL FIS_MONTH IS SH.TIMES.FISCAL_MONTH_ID
LEVEL FIS_QUARTER IS SH.TIMES.FISCAL_QUARTER_ID
LEVEL FIS_WEEK IS SH.TIMES.WEEK_ENDING_DAY_ID
LEVEL FIS_YEAR IS SH.TIMES.FISCAL_YEAR_ID
LEVEL MONTH IS SH.TIMES.CALENDAR_MONTH_ID
LEVEL QUARTER IS SH.TIMES.CALENDAR_QUARTER_ID
LEVEL YEAR IS SH.TIMES.CALENDAR_YEAR_ID
HIERARCHY CAL_ROLLUP (
    DAY CHILD OF
    MONTH CHILD OF
    QUARTER CHILD OF
    YEAR
)
HIERARCHY FIS_ROLLUP (
    DAY CHILD OF
    FIS_WEEK CHILD OF
    FIS_MONTH CHILD OF
    FIS_QUARTER CHILD OF
    FIS_YEAR
)
ATTRIBUTE DAY LEVEL DAY DETERMINES SH.TIMES.CALENDAR_WEEK_NUMBER
ATTRIBUTE DAY LEVEL DAY DETERMINES SH.TIMES.DAY_NAME
ATTRIBUTE DAY LEVEL DAY DETERMINES SH.TIMES.DAY_NUMBER_IN_MONTH
ATTRIBUTE DAY LEVEL DAY DETERMINES SH.TIMES.DAY_NUMBER_IN_WEEK
ATTRIBUTE FIS_MONTH LEVEL FIS_MONTH DETERMINES SH.TIMES.DAYS_IN_FIS_MONTH
ATTRIBUTE FIS_MONTH LEVEL FIS_MONTH DETERMINES SH.TIMES.END_OF_FIS_MONTH
ATTRIBUTE FIS_MONTH LEVEL FIS_MONTH DETERMINES SH.TIMES.FISCAL_MONTH_DESC
ATTRIBUTE FIS_MONTH LEVEL FIS_MONTH DETERMINES SH.TIMES.FISCAL_MONTH_NAME
ATTRIBUTE FIS_MONTH LEVEL FIS_MONTH DETERMINES SH.TIMES.FISCAL_MONTH_NUMBER
ATTRIBUTE FIS_QUARTER LEVEL FIS_QUARTER DETERMINES SH.TIMES.DAYS_IN_FIS_QUARTER
```



Materialized Views and Fact Tables

- Data warehouses commonly range in size from tens of gigabytes to terabytes – most of the data is stored in a few very large fact tables.
- Summaries are special types of aggregate views that improve query execution times by pre-calculating expensive joins and aggregation operations.
- Summaries can require a significant amount of time and effort in identification, creation, indexing, updating, and advising their users on which to use.

Oracle recommends building summaries or aggregates using a schema object called a materialized view



MVs and Query Rewrite

- A materialized view is a database object that contains the results of a query
- They are local copies of data located remotely, or are used to create summary tables based on aggregations of a table's data
- Materialized views can perform a number of roles, such as improving query performance or providing replicated data
- The use of Materialized Views is called summary management



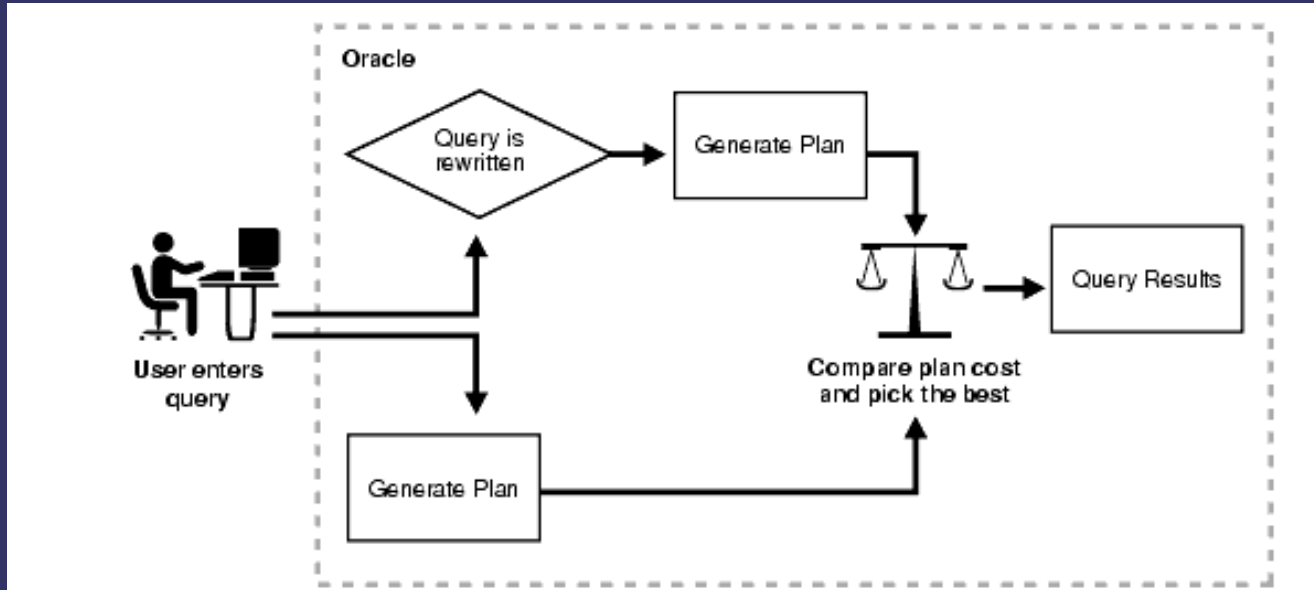
more...

- Materialized views within the data warehouse are transparent to the end user or to the database application
- Through the query rewrite mechanism the Oracle server automatically rewrites the SQL query to use the materialized view
- Query results come back *FAST*
- An end user or application can construct queries that directly access the materialized views, but this should be restricted so as to not alter the MV

When to use MVs

- Use materialized views to precompute and store aggregated data, referred to as summaries
- Materialized views also can be used to precompute expensive joins to eliminates the overhead
- In distributed environments, use materialized views to replicate data at distributed sites and to synchronize updates
 - These replicas provide local access to data that
 - MVs are also useful in remote data marts
- Use materialized views to download a subset of data from central servers to mobile clients, with periodic refreshes

HUH?



OHHH!

- The query optimizer automatically recognizes when an existing materialized view should be used to satisfy a request
- It transparently rewrites the request to use the materialized view going directly to the view and not to the underlying detail tables

Some Rules

- Materialized view used by query rewrite must be stored in the same database as the detail tables on which it relies
- Materialized view can be partitioned, and you can define a materialized view on a partitioned table
- Materialized views are allowed one or more indexes
- Materialized views can be accessed directly using a SELECT statement, however, it is recommended that you try to avoid writing SQL statements that directly reference the materialized view. Changing the query may affect your application. Let query rewrite transparently rewrite your query to use the materialized view.

DBMS_MVIEW

For Developers

- *EXPLAIN_MVIEW Procedure*
- Explains what is possible with a materialized view or potential materialized view
- *EXPLAIN_REWRITE Procedure*
- Explains why a query failed to rewrite or why the optimizer chose to rewrite a query with a particular materialized view or materialized views
- *REFRESH Procedures*
- Refreshes one or more materialized views that are not members of the same refresh group
- *REFRESH_ALL_MVIEWS Procedure*
- Refreshes all materialized views that do not reflect changes to their master table or master materialized view
- *REFRESH_DEPENDENT Procedures*
- Refreshes all table-based materialized views that depend on a specified master table or master materialized view, or list of master tables or master materialized views



And More

The MV_CAPABILITIES_TABLE

```
SELECT capability_name, possible, SUBSTR(related_text,1,8)
  AS rel_text, SUBSTR(msgtxt,1,60) AS msgtxt
FROM MV_CAPABILITIES_TABLE
ORDER BY seq;
```

CAPABILITY_NAME	P	REL_TEXT	MSGTXT
-----	-	-----	-----
PCT	N		
REFRESH_COMPLETE	Y		
REFRESH_FAST	N		
REWRITE	Y		
PCT_TABLE	N	SALES	no partition key or PMARKER in select list
PCT_TABLE	N	TIMES	relation is not a partitioned table
REFRESH_FAST_AFTER_INSERT	N	SH.TIMES	mv log must have new values
REFRESH_FAST_AFTER_INSERT	N	SH.TIMES	mv log must have ROWID
REFRESH_FAST_AFTER_INSERT	N	SH.TIMES	mv log does not have all necessary columns
REFRESH_FAST_AFTER_INSERT	N	SH.SALES	mv log must have new values
REFRESH_FAST_AFTER_INSERT	N	SH.SALES	mv log must have ROWID
REFRESH_FAST_AFTER_INSERT	N	SH.SALES	mv log does not have all necessary columns
REFRESH_FAST_AFTER_ONETAB_DML	N	DOLLARS	SUM(expr) without COUNT(expr)
REFRESH_FAST_AFTER_ONETAB_DML	N		see the reason why
REFRESH_FAST_AFTER_ONETAB_DML	N		REFRESH_FAST_AFTER_INSERT is disabled
REFRESH_FAST_AFTER_ONETAB_DML	N		COUNT(*) is not present in the select list
REFRESH_FAST_AFTER_ONETAB_DML	N		SUM(expr) without COUNT(expr)
REFRESH_FAST_AFTER_ANY_DML	N		see the reason why
REFRESH_FAST_AFTER_ANY_DML	N		REFRESH_FAST_AFTER_ONETAB_DML is disabled
REFRESH_FAST_AFTER_ANY_DML	N	SH.TIMES	mv log must have sequence
REFRESH_FAST_AFTER_ANY_DML	N	SH.SALES	mv log must have sequence
REFRESH_PCT	N		PCT is not possible on any of the detail tables in the materialized view
REWRITE_FULL_TEXT_MATCH	Y		
REWRITE_PARTIAL_TEXT_MATCH	Y		
REWRITE_GENERAL	Y		
REWRITE_PCT	N		PCT is not possible on any detail tables

Detour: Prebuilt Materialized Views

- Register a user-defined materialized view with the CREATE MATERIALIZED VIEW ... ON PREBUILT TABLE statement
- Once registered, the materialized view can be used for query rewrites or maintained by one of the refresh methods, or both.
- In the case where a, user-defined materialized view is refreshed on a schedule that is longer than the update cycle (Example, a monthly materialized view may be updated only at the end of each month, and the materialized view values always refer to complete time periods.)

Overview of Query Rewrite

- Query rewrite transforms a SQL statement expressed in terms of tables or views into a statement accessing one or more materialized views that are defined on the detail tables
- The transformation is transparent to the end user or application, requiring no intervention and no reference to the materialized view in the SQL statement
- Because query rewrite is transparent, materialized views can be added or dropped just like indexes without invalidating the SQL in the application code

How?

- The optimizer uses two different methods to recognize when to rewrite a query in terms of a materialized view.
- The first method is based on matching the SQL text of the query with the SQL text of the materialized view definition.
- If the first method fails, the optimizer uses the more general method in which it compares joins, selections, data columns, grouping columns, and aggregate functions between the query and materialized views.

Will the Query Rewrite?

The purpose behind the creation of most materialized view is to use query rewrite against them to speed performance

A defined Materialized will not automatically be used by the query rewrite facility

- Before creating a materialized view, verify what types of query rewrite are possible by calling the procedure `DBMS_MVIEW.EXPLAIN_MVIEW`
- Alternatively use, `DBMS_ADVISOR.TUNE_MVIEW` to optimize the materialized view to allow many types of query rewrite
- With existing MVs use `DBMS_MVIEW.EXPLAIN_REWRITE` to find out why or why not it will rewrite a specific query.

The SH Schema Offers Examples

```
CREATE MATERIALIZED VIEW sum_sales_pscat_week_mv
ENABLE QUERY REWRITE AS
SELECT p.prod_subcategory, t.week_ending_day,
SUM(s.amount_sold) AS sum_amount_sold
FROM sales s, products p, times t
WHERE s.time_id=t.time_id AND s.prod_id=p.prod_id
GROUP BY p.prod_subcategory, t.week_ending_day;
```

```
CREATE MATERIALIZED VIEW sum_sales_prod_week_mv
ENABLE QUERY REWRITE AS
SELECT p.prod_id, t.week_ending_day, s.cust_id,
SUM(s.amount_sold) AS sum_amount_sold
FROM sales s, products p, times t
WHERE s.time_id=t.time_id AND s.prod_id=p.prod_id
GROUP BY p.prod_id, t.week_ending_day, s.cust_id;
```

```
CREATE MATERIALIZED VIEW sum_sales_pscat_month_city_mv
ENABLE QUERY REWRITE AS
SELECT p.prod_subcategory, t.calendar_month_desc, c.cust_city,
SUM(s.amount_sold) AS sum_amount_sold,
COUNT(s.amount_sold) AS count_amount_sold
FROM sales s, products p, times t, customers c
WHERE s.time_id=t.time_id AND s.prod_id=p.prod_id AND s.cust_id=c.cust_id
GROUP BY p.prod_subcategory, t.calendar_month_desc, c.cust_city;
```

**Aggregate
Materialized Views**

**Materialized
Views Joins
Only**

```
CREATE MATERIALIZED VIEW join_sales_time_product_mv
ENABLE QUERY REWRITE AS
SELECT p.prod_id, p.prod_name, t.time_id, t.week_ending_day,
s.channel_id, s.promo_id, s.cust_id, s.amount_sold
FROM sales s, products p, times t
WHERE s.time_id=t.time_id AND s.prod_id = p.prod_id;
```

```
CREATE MATERIALIZED VIEW join_sales_time_product_oj_mv
ENABLE QUERY REWRITE AS
SELECT p.prod_id, p.prod_name, t.time_id, t.week_ending_day,
s.channel_id, s.promo_id, s.cust_id, s.amount_sold
FROM sales s, products p, times t
WHERE s.time_id=t.time_id AND s.prod_id=p.prod_id(+);
```

**STATS on a single
object or a schema**

```
EXECUTE DBMS_STATS.GATHER_TABLE_STATS ( -
'SH', 'JOIN_SALES_TIME_PRODUCT_MV', estimate_percent => 20, - block_sample => TRUE,
cascade => TRUE);
```

```
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS ( 'SH', -
options => 'GATHER EMPTY', - estimate_percent => 20, block_sample => TRUE, -
cascade => TRUE);
```

Partitions in Data Warehouses

Large tables require techniques both for management and good query performance and partitioning addresses these needs

The partitioning columns (or subpartitioning columns) of a table or index consist of an ordered list of columns whose values determine how the data is partitioned or subpartitioned.

This list can include up to 16 columns

3 Partitioning Strategies:

Range

Hash

List



Hash Partitioning

Hash partitioning maps data to partitions based on a hashing algorithm that Oracle applies to a partitioning key

- The hashing algorithm evenly distributes rows among partitions, giving partitions approximately the same size.
- Hash partitioning is the ideal method for distributing data evenly across devices
- Hash partitioning is also an easy-to-use alternative to range partitioning, especially when the data to be partitioned is not historical.
- Oracle Database uses a linear hashing algorithm and to prevent data from clustering within specific partitions

```
CREATE TABLE sales_hash  
(salesman_id NUMBER(5),  
salesman_name VARCHAR2(30),  
sales_amount NUMBER(10),  
week_no NUMBER(2))  
PARTITION BY HASH(salesman_id);
```

Define the number of partitions
by a power of two

List Partitioning

- List partitioning explicitly control how rows map to partitions
- Specify a list of discrete values for the partitioning column in the description for each partition
- A range of values is associated with a partition and with hash partitioning, with control of the row-to-partition mapping

```
CREATE TABLE sales_list
(salesman_id NUMBER(5),
 salesman_name VARCHAR2(30),
 sales_state VARCHAR2(20),
 sales_amount NUMBER(10),
 sales_date DATE)
PARTITION BY LIST(sales_state)
(PARTITION sales_west VALUES('California', 'Hawaii') COMPRESS,
 PARTITION sales_east VALUES('New York', 'Virginia', 'Florida'),
 PARTITION sales_central VALUES('Texas', 'Illinois'));
```

Group and organize unordered and unrelated sets of data in a natural way

Partition Pruning Tips

- When using the Oracle DATE data type, if the user or application passes date using a character string, Oracle may not make the conversion to a the DATE data type

```
where date_column = '01-MAR-10';
```

- As a result, Oracle will not use partition pruning on a table partitioned by DATE
- A properly applied TO_DATE function guarantees that Oracle is capable of uniquely determining the date value and using it for pruning
- The optimizer cannot perform pruning when an operator is on top of a partitioning column
- This could be an explicit operator (for example, a function) or even an implicit operator introduced by Oracle as part of the necessary data type conversion for executing the statement

```
WHERE time_id = TO_TIMESTAMP('1-jan'2000', 'dd-mon-yyyy');
```



Constraints in the Data Warehouse

UNIQUE Constraints

- In a data warehouse creating a unique index can be costly both in processing time and in disk space.
- An alternative mechanism for unique constraints is:
`ALTER TABLE sales ADD CONSTRAINT xx_uk UNIQUE (col1, col2, col3) DISABLE VALIDATE;`
- Creates a unique constraint, but, because the constraint is disabled, not a unique index is not required

FOREIGN KEYS

- In some situations, you may choose to use the FOREIGN KEY ENABLE NOVALIDATE state
- The tables contain data that currently disobeys the constraint, but the data warehouse administrator wishes to create a constraint for future enforcement
- An enforced constraint is not required immediately

Example: data is moved into the fact tables every day, but the dimension tables are only refreshed on the weekend



Parallelism

Integrity Constraints and Parallelism

- All constraints can be validated in parallel
- When validating constraints on very large tables, parallelism is often necessary to meet performance goals

The degree of parallelism for a given constraint operation is determined by the default degree of parallelism of the underlying table

In most cases, the recommendation is turn off parallelism in the table and achieve it in the queries through hints

SQL for Aggregation in Data Warehouses

To improve aggregation performance in your warehouse, Oracle Database provides the following extensions to the GROUP BY clause:

- CUBE and ROLLUP extensions to the GROUP BY clause
- GROUPING(), GROUPING_ID() and GROUPING SETS
- GROUPING SETS expression
- The CUBE, ROLLUP, and GROUPING SETS extensions to SQL make querying and reporting easier and faster
- CUBE, ROLLUP, and grouping sets produce a single result set that is equivalent to a UNION ALL of differently grouped rows
- ROLLUP calculates aggregations such as SUM, COUNT, MAX, MIN, and AVG at increasing levels of aggregation, from the most detailed up to a grand total. CUBE is an extension similar to ROLLUP, enabling a single statement to calculate all possible combinations of aggregations

Oracle's family of Enhanced Analytical Functions

Type	Used For
Ranking	Calculating ranks, percentiles, and n-tiles of the values in a result set.
Windowing	Calculating cumulative and moving aggregates. Works with these functions: SUM, AVG, MIN, MAX, COUNT, VARIANCE, STDDEV, FIRST_VALUE, LAST_VALUE, and new statistical functions. Note that the DISTINCT keyword is not supported in windowing functions except for MAX and MIN.
Reporting	Calculating shares, for example, market share. Works with these functions: SUM, AVG, MIN, MAX, COUNT (with/without DISTINCT), VARIANCE, STDDEV, RATIO_TO_REPORT, and new statistical functions. Note that the DISTINCT keyword may be used in those reporting functions that support DISTINCT in aggregate mode.
LAG/LEAD	Finding a value in a row a specified number of rows from a current row.
FIRST/LAST	First or last value in an ordered group.
Linear Regression	Calculating linear regression and other statistics (slope, intercept, and so on).
Inverse Percentile	The value in a data set that corresponds to a specified percentile.
Hypothetical Rank and Distribution	The rank or percentile that a row would have if inserted into a specified data set.

Advanced Analytic Functions

- The LISTAGG function orders data within each group based on the ORDER BY clause and then concatenates the values of the measure column
- The FIRST/LAST aggregate functions allow you to rank a data set and work with its top-ranked or bottom-ranked rows -- after finding the top or bottom ranked rows
- PERCENTILE_CONT is a continuous function computed by interpolation
- PERCENTILE_DISC is a step function that assumes discrete values
- PERCENTILE_DISC(x) is computed by scanning up the CUME_DIST values in each group till finding the first one greater than or equal to x, where x is the specified percentile value
- Hypothetical Rank: These functions provide functionality useful for what-if analysis. As an example, what would be the rank of a row, if the row was hypothetically inserted into a set of other rows?

DBMS_STAT_FUNCS

- `MEDIAN (expr) [OVER (query_partition_clause)]`
- `STATS_MODE (expr)`
- `STATS_T_TEST_ONE (expr1, expr2 (a constant) [, return_value])`
- `STATS_T_TEST_PAIRED (expr1, expr2 [, return_value])`
- `STATS_T_TEST_INDEP (expr1, expr2 [, return_value])`
- `STATS_T_TEST_INDEPU (expr1, expr2 [, return_value])`
- `STATS_F_TEST (expr1, expr2 [, return_value])`
- `STATS_ONE_WAY_ANOVA (expr1, expr2 [, return_value])`
- `STATS_CROSSTAB (expr1, expr2 [, return_value])`
- `STATS_BINOMIAL_TEST (expr1, expr2, p [, return_value])`
- `STATS_WSR_TEST (expr1, expr2 [, return_value])`
- `STATS_MW_TEST (expr1, expr2 [, return_value])`
- `STATS_KS_TEST (expr1, expr2 [, return_value])`
- `CORR_S (expr1, expr2 [, return_value])`
- `CORR_K (expr1, expr2 [, return_value])`

Hypothesis Testing

Parametric Test

Others



PIVOT Operations

```
SELECT ....
FROM <table-expr>
PIVOT
  (aggregate-function(<column>)
  FOR <pivot-column> IN (<value1>, <value2>, ..., <valuen>)) AS <alias>
WHERE .....
```

Example :

```
CREATE VIEW sales_view AS
SELECT
  prod_name product, country_name country, channel_id channel,
  SUBSTR(calendar_quarter_desc, 6,2) quarter,
  SUM(amount_sold) amount_sold, SUM(quantity_sold) quantity_sold
FROM sales, times, customers, countries, products
WHERE sales.time_id = times.time_id AND
sales.prod_id = products.prod_id AND
sales.cust_id = customers.cust_id AND
customers.country_id = countries.country_id
GROUP BY prod_name, country_name, channel_id,
SUBSTR(calendar_quarter_desc, 6, 2);
```

- Pivoting operations can be performed on Multiple Columns
- Multiple Aggregates
- Unpivoting -- which do not reverse a PIVOT operation. Instead, it rotates data from columns into rows



Data Densification

- Data Densification for Reporting addresses the fact that data is normally stored in sparse form – if no value exists for a given combination of dimension values, no row exists in the fact table
- To view the data in dense form, with rows for all combination of dimension values are displayed even when no fact data exist for them
- Data densification converts sparse data into dense form
- It uses a partitioned outer join to fill the gaps in a dimension

```
SELECT .....  
FROM table_reference  
PARTITION BY (expr [, expr ]... )  
RIGHT OUTER JOIN table_reference  
SELECT .....  
FROM table_reference  
LEFT OUTER JOIN table_reference PARTITION BY {expr [,expr ]...}  
Note that FULL OUTER JOIN is not supported with a partitioned outer join.
```

The OBIEE “Stack”

- Oracle BI Server - Centralized data access and calculation via a logical Common Enterprise Information Model through to the end user products and other SQL based tools.
- Oracle BI Interactive Dashboards - Personalized, highly intuitive, guided and fully interactive access to cockpits of live analyses.
- Oracle BI Answers - Self-service ad-hoc capabilities allowing end users to easily create charts, pivot tables, reports, and visually appealing dashboards, all of which are fully interactive and drillable.
- Oracle BI Delivers - Proactive intelligence solution providing alerting that can reach users via multiple channels (email, dashboards, and mobile devices) as well as workflow integration.
- Oracle BI Disconnected Analytics - Full business intelligence functionality for the mobile professional, enabling fully interactive dashboards and ad hoc analysis while disconnected from the corporate network.
- Oracle BI Publisher - High-fidelity report templates that are created and published via common personal productivity applications delivered directly or through Interactive Dashboards to end users.
- Oracle BI for Microsoft Office - Allows users of Microsoft Excel and PowerPoint to access and run Oracle BI reports directly within these familiar tools



A Rich User Experience

ORACLE Interactive Dashboards My Dashboard 00 Overview 01 Ranks & Toppers 02 History & Benching 03 Tiering & Distribution

03 Tiering & Distribution Administrator! Dashboards - Answers - More Products - Settings - Page Opt

31 80 20 32 Tiering 33 Deciling Comparative 34 Distribution 35 Comparative Distr. 36 Variability

Set # of Bins: 15 Go Year: Quarter: Go

Customer: Team: Go

Statistical Distribution Select Grain: C1 Cust Name

1 - Value Distribution

2 - Counts Distribution

Sophisticated Graphics

Ranking, Statical Functions, Dimensional Analysis

Tabular Displays

All Displays Feature Interactive Drill-Down Capabilities

Bin	Bin Bds	Avg Value	Records Count	Cum	1-01 Revenue (Sum All)	Cum Value	Pct Value	Cum
1	6.95-29.75	19,140	44	44	842,139	842,139	3.38%	3%
2	29.75-52.55	43,274	28	72	1,211,678	2,053,817	4.87%	8%
3	52.55-75.34	83,590	24	98	1,526,169	3,579,986	6.13%	14%
4	75.34-98.14	89,879	31	127	2,786,238	6,366,224	11.19%	26%



Dashboards – A User Tool

Answers: An Ad Hoc Tool for Users and a Powerful Reporting Tool for Developers

User Friendly Presentation Layer

Click and Drop to Define Queries

Format Data Sets into multiple views and charts

The screenshot displays the Oracle BI Answers web interface. At the top, there are browser tabs for 'Oracle OLAP 11g Sample S...', 'Oracle Business Intelligenc...', and 'Oracle BI Answers'. The main interface has a blue header with 'ORACLE Answers' and tabs for 'Criteria', 'Results', 'Prompts', and 'Advanced'. A left-hand pane shows a tree view of data sources, including 'CHANNELS' and 'PRODUCTS'. The main area contains a query builder with columns for 'TIMES', 'CHANNELS', 'PRODUCTS', and 'SALES'. Below this is a 'Display Results' button. A second screenshot below shows the 'Results' tab with a 'Pivot Table' dropdown menu open, listing options like 'Compound Layout', 'Table', 'Chart', and 'Pivot Table'. The interface also shows 'Columns' and 'Measures' sections for configuring the data display.

TIMES	CHANNELS	PRODUCTS	SALES
FISCAL_MONTH_NAME	CHANNEL_CLASS	PROD_CATEGORY	AMOUNT_SOLD

Columns	Measures
CHANNELS	SALES
CHANNEL_CLASS	SUM(AMOUNT_SOLD)

A Rich User Experience

ORACLE Interactive Dashboards My Dashboard 00 Overview 01 Ranks & Toppers 02 History & Benching 03 Tiering & Distribution

03 Tiering & Distribution Administrator! Dashboards - Answers - More Products - Settings - Page Opt

31 80 20 32 Tiering 33 Deciling Comparative 34 Distribution 35 Comparative Distr. 36 Variability

Set # of Bins: 15 Go Year: Quarter: Go

Customer: Team: Go

Statistical Distribution Select Grain: C1 Cust Name

1 - Value Distribution

2 - Counts Distribution

Sophisticated Graphics

Ranking, Statical Functions, Dimensional Analysis

Tabular Displays

All Displays Feature Interactive Drill-Down Capabilities

Bin	Bin Bds	Avg Value	Records Count	Cum	1-01 Revenue (Sum All)	Cum Value	Pct Value	Cum
1	6.95-29.75	19,140	44	44	842,139	842,139	3.38%	3%
2	29.75-52.55	43,274	28	72	1,211,678	2,053,817	4.87%	8%
3	52.55-75.34	83,590	24	98	1,526,169	3,579,986	6.13%	14%
4	75.34-98.14	89,879	31	127	2,786,238	6,366,224	11.19%	26%



Dashboards – A User Tool

Answers: An Ad Hoc Tool for Users and a Powerful Reporting Tool for Developers

User Friendly Presentation Layer

Click and Drop to Define Queries

Format Data Sets into multiple views and charts

The screenshot displays the Oracle BI Answers web interface. At the top, there are browser tabs for 'Oracle OLAP 11g Sample S...', 'Oracle Business Intelligenc...', and 'Oracle BI Answers'. The main interface has a blue header with 'ORACLE Answers' and tabs for 'Criteria', 'Results', 'Prompts', and 'Advanced'. A left-hand pane shows a tree view of data sources, including 'CHANNELS' and 'PRODUCTS'. The main area contains a query builder with columns for 'TIMES', 'CHANNELS', 'PRODUCTS', and 'SALES'. Below this is a 'Display Results' button. A second screenshot below shows the 'Results' tab with a 'Pivot Table' dropdown menu open, listing options like 'Compound Layout', 'Table', 'Chart', and 'Pivot Table'. The interface also shows 'Columns' and 'Measures' sections for configuring the data display.

TIMES	CHANNELS	PRODUCTS	SALES
FISCAL_MONTH_NAME	CHANNEL_CLASS	PROD_CATEGORY	AMOUNT_SOLD

Columns

Measure Labels	
CHANNELS	CHANNEL_CLASS

Measures

SALES	SUM(AMOUNT_SOLD)
-------	------------------

Tuning

- Run Stats. Make sure they are up to date –
 - When you create a new instance, run database stats
 - When you load tables or build indexes
 - Run schema stats – Run schema stats – run schema stats
- Don't load fact tables – use CTAS and IAS and swap partitions

Sqldeveloper
has reports which
are handy



The screenshot shows the Oracle SQL Developer interface. The 'Active Sessions' window displays a list of sessions with columns: INST_ID, PROGRAM, MODULE, EVENT, SQL_ID, and SQL_TEXT. The selected session shows the SQL text: `select s.inst_id,s.program,s.module`. Below this, the 'Explain Plan' window is open, showing the query plan for the same SQL statement. The plan includes the following steps:

INST_ID	PLAN_HASH_VALUE	Query Plan	Rows
1	1351996005	SELECT STATEMENT Cost = 1	
1	1351996005	NESTED LOOPS	1
1	1351996005	NESTED LOOPS	1
1	1351996005	NESTED LOOPS	1
1	1351996005	FIXED TABLE FULL X%KSUSE	1
1	1351996005	FIXED TABLE FIXED INDEX X%KSLWT (ind:1)	1
1	1351996005	FIXED TABLE FIXED INDEX X%KSLED (ind:2)	1
1	1351996005	FIXED TABLE FIXED INDEX X%KGLCURSOR_CHILD (ind:2)	1

Bibliography

- Kimball, Ralph and Ross, Margy, *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*, Second Edition, John Wiley and Sons, New York, NY
- Oracle Database Data Warehousing Guide, *11g Release 2 (11.2)*, E10810-03, Copyright © 2001, 2010, Oracle and/or its affiliates. All rights reserved., Primary Author: Paul Lane, Contributing Author: Viv Schupmann (Change Data Capture)
- ORACLE DATA SHEET, ORACLE DATABASE 10G OLAP OPTION, *January 2004 1*
- Oracle Database Data Warehousing Guide, *10g Release 2 (10.2)*, B14223-02, Copyright © 2001, 2005, Oracle. All rights reserved.. Primary Author: Paul Lane, Contributing Author: Viv Schupmann and Ingrid Stuart (Change Data Capture)
- Best practices for a Data Warehouse on Oracle Database 11g, An Oracle White Paper, September 2008
- Oracle Database 11g for Data Warehousing and Business Intelligence, An Oracle White Paper September, 2009, Author: George Lumpkin