**Types of fact in data warehouse**

**Fact Table**

Each data warehouse or data mart includes one or more fact tables. The fact table captures the data that measures the organizations business operations. A fact table might contain business sales events such as cash register transactions or the contributions and expenditures of a nonprofit organization. Fact tables usually contain large numbers of rows, sometimes in the hundreds of millions of records when they contain one or more years of history for a large organization. A key characteristic of a fact table is that it contains numerical data (facts) that can be summarized to provide information about the history of the operation of the organization. Each fact table also includes a multipart index that contains as foreign keys the primary keys of related dimension tables, which contain the attributes of the fact records. Fact tables should not contain descriptive information or any data other than the numerical measurement fields and the index fields that relate the facts to corresponding entries in the dimension tables. An example of fact table is Sales\_Fact table that might contain the information like sale\_amount, unit\_price, discount, etc.

**TYPES :**

**Snapshot fact table(Transactional Periodic Snapshot)**

Deals with the particular period time. They contain non-additive and semi-additive facts.

OR

A fact table generally contains a HISTORY of transactions (facts)   
and the volume of data grows over a period of time. The growth   
of volume impacts query performance. A historical perspective is   
good for trend analysis etc.   
However, most information requirements are related to the current   
"period", last week, last month etc. A snapshot fact table is   
created to store the "current" facts for optimum performance.   
Many snapshot tables may also contain a level of aggregation as   
well. For example, the granular facts could be purchase   
transactions for a customer, store, product and date time. The   
information requirements are for facts at store, product, week   
level. The fact table would then have the transactions   
aggregated to the store, product, week level.   
Hope that clarifies things.

OR

This type of fact table describes the state of things in a particular instance of time, and usually includes more semi-additive and non-additive facts. The second example presented here is a snapshot fact table.

Eg: Daily balances fact can be summed up through the customers dimension but not through the time dimension.

**Cumulative**

This type of fact table describes what has happened over a period of time. For example, this fact table may describe the total sales by product by store by day. The facts for this type of fact tables are mostly additive facts. The first example presented here is a cumulative fact table.

Eg: Sales fact

OR

Describes what was happened over the period of time. They contain addition facts.

**Factless Fact Table**

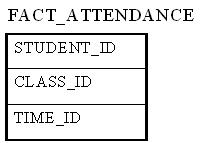
In the real world, it is possible to have a fact table that contains no measures or facts. These tables are called “Factless Fact tables”.

Eg: A fact table which has only product key and date key is a factless fact. There are no measures in this table. But still you can get the number products sold over a period of time.

OR

A factless fact table is a fact table that does not have any measures. It is essentially an intersection of dimensions. On the surface, a factless fact table does not make sense, since a fact table is, after all, about facts. However, there are situations where having this kind of relationship makes sense in data warehousing.

For example, think about a record of student attendance in classes. In this case, the fact table would consist of 3 dimensions: the student dimension, the time dimension, and the class dimension. This factless fact table would look like the following:



The only measure that you can possibly attach to each combination is "1" to show the presence of that particular combination. However, adding a fact that always shows 1 is redundant because we can simply use the COUNT function in SQL to answer the same questions.

Factless fact tables offer the most flexibility in data warehouse design. For example, one can easily answer the following questions with this factless fact table:

How many students attended a particular class on a particular day?

How many classes on average does a student attend on a given day?

Without using a factless fact table, we will need two separate fact tables to answer the above two questions. With the above factless fact table, it becomes the only fact table that's needed.

OR

We know that fact table is a collection of many facts and measures having multiple keys joined with one or more dimesion tables.Facts contain both numeric and additive fields.But factless fact table are different from all these.

A factless fact table is fact table that does not contain fact.They contain only dimesional keys and it captures events that happen only at information level but not included in the calculations level.just an information about an event that happenoveraperiod.

A factless fact table captures the many-to-many relationships between dimensions, but contains no numeric or textual facts. They are often used to record events or coverage information. Common examples of factless fact tables include:

Identifying product promotion events (to determine promoted products that didn’t sell)

Tracking student attendance or registration events

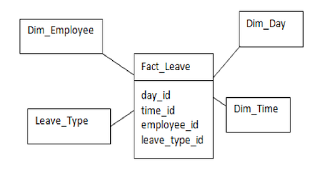
Tracking insurance-related accident events

Identifying building, facility, and equipment schedules for a hospital or university

Factless fact tables are used for tracking a process or collecting stats. They are called so because, the fact table does not have aggregatable numeric values or information.There are two types of factless fact tables: those that describe events, and those that describe conditions. Both may play important roles in your dimensional models.

FactlessfacttablesforEvents  
The first type of factless fact table is a table that records an event. Many event-tracking tables in dimensional data warehouses turn out to be factless.Sometimes there seem to be no facts associated with an important business process. Events or activities occur that you wish to track, but you find no measurements. In situations like this, build a standard transaction-grained fact table that contains no facts.

For eg.

[](http://4.bp.blogspot.com/-bubKBkFu5Wk/UBzk_1QdSZI/AAAAAAAAAEQ/k_kgAuQ6JW8/s1600/employee_leave.png)

The above fact is used to capture the leave taken by an employee.Whenever an employee takes leave a record is created with the dimensions.Using the fact FACT\_LEAVE we can answer many questions like

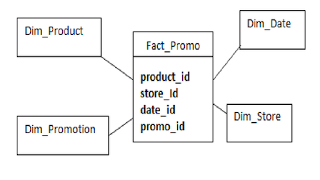
Number of leaves taken by an employee

The type of leave an employee takes

Details of the employee who took leave

FactlessfacttablesforConditions  
Factless fact tables are also used to model conditions or other important relationships among dimensions. In these cases, there are no clear transactions or events.It is used to support negative analysis report. For example a Store that did not sell a product for a given period.  To produce such report, you need to have a fact table to capture all the possible combinations.  You can then figure out what is missing.

For eg, fact\_promo gives the information about the products which have promotions but still did not sell

[](http://4.bp.blogspot.com/-xLLGsfur04I/UBzlehssIKI/AAAAAAAAAEY/auIUYkG9_04/s1600/promotional_ewvent.png)

This fact answers the below questions:

To find out products that have promotions.

To find out products that have promotion that sell.

The list of products that have promotion but did not sell.

This kind of factless fact table is used to track conditions, coverage or eligibility.  In Kimball terminology, it is called a "coverage table."

Note:

We may have the question that why we cannot include these information in the actual fact table .The problem is that if we do so then the fact size will increase enormously .

Factless fact table is crucial in many complex business processes. By applying you can design a dimensional model that has no clear facts to produce more meaningful information for your business processes.Factless fact table itself can be used to generate the useful reports.

**Types of Dimensions in data warehouse**

A dimension table consists of the attributes about the facts. Dimensions store the textual descriptions of the business. Without the dimensions, we cannot measure the facts. The different types of dimension tables are explained in detail below.  
  
**Conformed Dimension:**Conformed dimensions mean the exact same thing with every possible fact table to which they are joined.   
  
Eg: The date dimension table connected to the sales facts is identical to the date dimension connected to the inventory facts.

OR

These dimensions are something that is built once in your model and can be reused multiple times with different fact tables.   For example, consider a model containing multiple fact tables, representing different data marts.  Now look for a dimension that is common to these facts tables.  In this example let’s consider that the product dimension is common and hence can be reused by creating short cuts and joining the different fact tables.Some of the examples are time dimension, customer dimensions, product dimension.

**Junk Dimension:**  
A junk dimension is a collection of random transactional codes flags and/or text attributes that are unrelated to any particular dimension. The junk dimension is simply a structure that provides a convenient place to store the junk attributes.  
  
Eg: Assume that we have a gender dimension and marital status dimension. In the fact table we need to maintain two keys referring to these dimensions. Instead of that create a junk dimension which has all the combinations of gender and marital status (cross join gender and marital status table and create a junk table). Now we can maintain only one key in the fact table.

OR

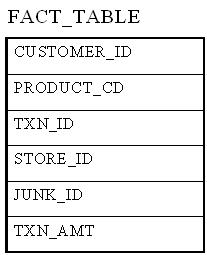
In data warehouse design, frequently we run into a situation where there are yes/no indicator fields in the source system. Through business analysis, we know it is necessary to keep such information in the fact table. However, if keep all those indicator fields in the fact table, not only do we need to build many small dimension tables, but the amount of information stored in the fact table also increases tremendously, leading to possible performance and management issues.

Junk dimension is the way to solve this problem. In a junk dimension, we combine these indicator fields into a single dimension. This way, we'll only need to build a single dimension table, and the number of fields in the fact table, as well as the size of the fact table, can be decreased. The content in the junk dimension table is the combination of all possible values of the individual indicator fields.

Let's look at an example. Assuming that we have the following fact table:

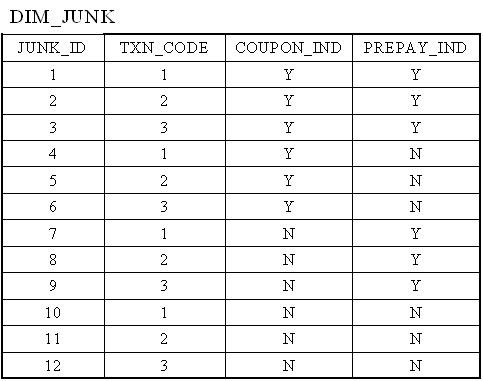


In this example, TXN\_CODE, COUPON\_IND, and PREPAY\_IND are all indicator fields. In this existing format, each one of them is a dimension. Using the junk dimension principle, we can combine them into a single junk dimension, resulting in the following fact table:



Note that now the number of dimensions in the fact table went from 7 to 5.

The content of the junk dimension table would look like the following:



In this case, we have 3 possible values for the TXN\_CODE field, 2 possible values for the COUPON\_IND field, and 2 possible values for the PREPAY\_IND field. This results in a total of 3 x 2 x 2 = 12 rows for the junk dimension table.

By using a junk dimension to replace the 3 indicator fields, we have decreased the number of dimensions by 2 and also decreased the number of fields in the fact table by 2. This will result in a data warehousing environment that offer better performance as well as being easier to manage.

OR

The junk dimension is simply a structure that provides a convenient place to store the junk attributes. It is just a collection of random transactional codes, flags and/or text attributes that are unrelated to any particular dimension.

In OLTP tables that are full of flag fields and yes/no attributes, many of which are used for operational support and have no documentation except for the column names and the memory banks of the person who created them. Not only do those types of attributes not integrate easily into conventional dimensions such as Customer, Vendor, Time, Location, and Product, but you also don’t want to carry bad design into the data warehouse.However, some of the miscellaneous attributes will contain data that has significant business value, so you have to do something with them.

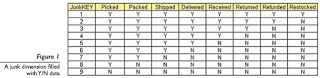
This scenario is especially common in legacy systems and databases that were created without solid, underlying design principles. Column names such as Completed, Packed, Shipped, Received, Delivered, and Returned (each with yes/no data values) are very common, and they do have business value.These miscellaneous indicators and flags that don't logically belong to the core dimension tables.  They are either too valuable to ignore or exclude.Often the meaning of the flags and text attributes is obscure. This situation leaves the designer with a number of bad alternatives

Designers sometimes want to treat them as Fact or make it into numerous small Dimensional tables. However, all of these options are less than ideal. Discarding the data can be dangerous because the miscellaneous values, flags, and yes/no fields might contain valuable business data. Including the miscellaneous attributes in the fact table could cause the fact table to swell to alarming proportions, especially if you have more than just a few miscellaneous attributes. The increased size of the fact table could cause serious performance problems because of the reduced number of records per physical I/O. Even if you tried to index these fields to minimize the performance problems, you still wouldn’t gain anything because so many of the miscellaneous fields contain flag values such as 0 and 1; Y and N; or open, pending, and closed.

A third, less obvious but preferable, solution is to incorporate a Junk Dimension as a holding place for these flags and indicators.

Advantage of junk dimension:

* It provides a recognizable location for related codes, indicators and their descriptors in a dimensional framework.
* This avoids the creation of multiple dimension tables.
* Provide a smaller, quicker point of entry for queries compared to performance when these attributes are directly in the fact table.
* An interesting use for a junk dimension is to capture the context of a specific transaction.  While our common, conformed dimensions contain the key dimensional attributes of interest, there are likely attributes about the transaction that are not known until the transaction is processed.

[](http://3.bp.blogspot.com/-lqMa_YA710U/UBlE8lbEGMI/AAAAAAAAADg/vn2EpXQGlI0/s1600/junk.jpg)

Above figure shows a junk dimension. As in any dimensional design, each of the rows in the fact table will be associated with a row in this junk dimension.

Simple Datawarehouse - Junk Dimension

You want to keep the data warehouse design as simple and straightforward as possible, so that users will be able to access data easily. Miscellaneous attributes that contain business value are a challenge to include in your data warehouse design because they don’t fit neatly into conventional dimensions, and if improperly handled, can cause the data warehouse to swell in size and perform suboptimally. By placing miscellaneous attributes into junk dimensions, you can circumvent both of these problems.

**Degenerated Dimension:**  
A degenerate dimension is a dimension which is derived from the fact table and doesn't have its own dimension table.  
  
Eg: A transactional code in a fact table.

OR

Degenerative Dimension is something dimensional in nature but exist fact table.(Invoice No)

A column of the key section of the fact table that does not have the associated dimension table but used for reporting and analysis, such column is called **Degenerate dimension or Line item dimensio**n.

For ex, we have a fact table with customer\_id, product\_id, branch\_id, employee\_id, bill\_no, and date in key section and price, quantity, amount in measure section. In this fact table, bill\_no from key section is a single value; it has no associated dimension table. Instead of creating a Separate dimension table for that single value, we can Include it in fact table to improve performance. SO here the column, bill\_no is a degenerate dimension or line item dimension.

**Role-playing dimension:**  
Dimensions which are often used for multiple purposes within the same database are called role-playing dimensions. For example, a date dimension can be used for “date of sale", as well as "date of delivery", or "date of hire"

OR

A **Role - Playing** is a situation in data warehouse where a single dimension appears multiple number of times in the same fact table.

For example – The "**Date**" dimension can be used for "Date of Purchase", "Date of Sale", or "Date of Delivery".

.

Dimension Table features

1. It provides the context /descriptive information for fact table measurements.

2. Provides entry points to data.

3. Structure of Dimension - Surrogate key , one or more other fields that compose the natural key (nk) and set of Attributes.

4. Size of Dimension Table is smaller than Fact Table.

5. In a schema more number of dimensions are presented than Fact Table.

6. Surrogate Key is used to prevent the primary key (pk) violation(store historical data).

7. Values of fields are in numeric and text representation.

**Fact Table features**

1. It provides measurement of an enterprise.

2. Measurement is the amount determined by observation.

3. Structure of Fact Table - foreign key (fk), Degenerated Dimension and Measurements.

4. Size of Fact Table is larger than Dimension Table.

5. In a schema less number of Fact Tables observed compared to Dimension Tables.

6. Compose of Degenerate Dimension fields act as Primary Key.

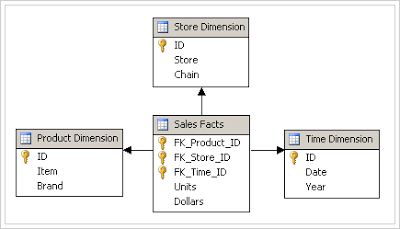
7. Values of the fields always in numeric or integer form.

**Dimensional Modeling/Datamodeling**

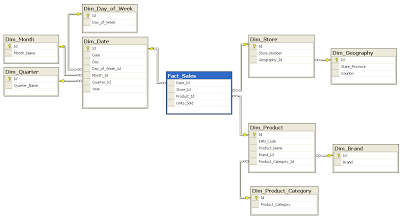
**Data Warehouse Dimensional Modelling (Types of Schemas)**

Dimensional modeling is the design concept used by many data warehouse designers to build their data warehouse. Dimensional model is the underlying data model used by many of the commercial OLAP products available today in the market. Designing a data warehouse is very different from designing an online transaction processing (OLTP) system. In contrast to an OLTP system in which the purpose is to capture high rates of data changes and additions, the purpose of a data warehouse is to organize large amounts of stable data for ease of analysis and retrieval. Because of these differing purposes, there are many considerations in data warehouse design that differ from OLTP database design. In dimensional model, all data is contained in two types of tables called Fact Table and Dimension Table.

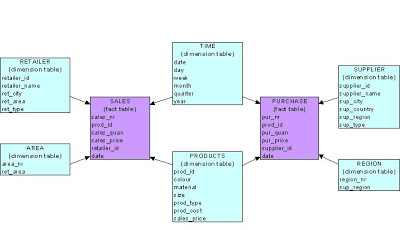
**StarSchema:**A star schema is the one in which a central fact table is sourrounded by denormalized dimensional tables. A star schema can be simple or complex. A simple star schema consists of one fact table where as a complex star schema have more than one fact table.

[](http://3.bp.blogspot.com/_pjSOGJIjDMo/S1w8XBuw8jI/AAAAAAAAADI/8SMt2Io0P-A/s1600-h/star-schema.png)

**Snow Flake Schema:**  
A snow flake schema is an enhancement of star schema by adding additional dimensions. Snow flake schema are useful when there are low cardinality attributes in the dimensions.

[](http://1.bp.blogspot.com/_pjSOGJIjDMo/S1w_JRMx3XI/AAAAAAAAADQ/J6cKyxJiwbM/s1600-h/Snowflake-schema.png)

**Galaxy Schema:**Galaxy schema contains many fact tables with some common dimensions (conformed dimensions). This schema is a combination of many data marts.

[](http://1.bp.blogspot.com/_pjSOGJIjDMo/S1w_SAaqJBI/AAAAAAAAADY/_kYiR3xwbCQ/s1600-h/galaxy.bmp)

**FactConstellationSchema:**The dimensions in this schema are segregated into independent dimensions based on the levels of hierarchy. For example, if geography has five levels of hierarchy like territory, region, country, state and city; constellation schema would have five dimensions instead of one.

**What is Cardinality in data warehouse**

In computing, Cardinality is a word that denotes the uniqueness of data in a column. Columns containing highly unique data—ID numbers, for example, which are unique to each individual— are said to have “high cardinality.” Columns can also be defined as possessing “normal” or “low” data cardinality.

Cardinality in data can also express the relationships that tables have with one another, and are defined in their cardinality as either “many to many,” “one-to-many” or “one-to-one.”

**What is a bitmap index :**

A bitmap index is a specialized variation of a B-tree index. If the degree of cardinality is high for the attribute, means that there are more unique number of values for a particular attribute. Low cardinality attribute is not suitable for bitmap index because more number of records are locked which result in the locking of a whole table, leading to the lock on a whole database. For e.g. A gender column, which has only two distinct values (male and female), is optimal for a bitmap index. However, data warehouse administrators also build bitmap indexes on columns with higher cardinalities.

You can use a bitmap index when both of the following conditions are true:

* The key values in the index contain many duplicates.
* More than one column in the table has an index that the optimizer can use to improve performance on a table scan.

Each bit in the bitmap corresponds to a possible rowid, and if the bit is set, it means that the row with the corresponding rowid contains the key value. A mapping function converts the bit position to an actual rowid, so that the bitmap index provides the same functionality as a regular index. Bitmap indexes store the bitmaps in a compressed way. If the number of distinct key values is small, bitmap indexes compress better and the space saving benefit compared to a B-tree index becomes even better

Note : When creating bitmap indexes, you should use NOLOGGING and COMPUTE STATISTICS. In addition, you should keep in mind that bitmap indexes are usually easier to destroy and re-create than to maintain.

**ADVANTAGES**

The Advantages of using bitmap indexes are greatest for columns in which the ratio of the number of distinct values to the number of rows in the table is small.

Space requirements for indexes in a warehouse are often significantly larger than the space needed to store the data, especially for the fact table and particularly if the indexes are B\*trees. Hence, you may want to keep indexing on the fact table to a minimum. Typically, you may have one or two concatenated B\*tree indexes on the fact table; however, most of your indexes should be bitmap indexes. Bitmap indexes also take up much less space than B\*tree indexes and so should be preferred

**How does it work?**

The bitmap index stores the column values in bits. Each bit represents a single value. For example, the gender column has two possible values: Male and Female. Three bit will be used in the bitmap to capture the index on the gender column. A good example can be seen in reference 1. So the more distinct value is, the more space is required to store the bitmap.

Internally, the database engine, like Oracle, uses a map function to converts the bit location to the distinct value. (See reference #2) Many bitmap indexes can be used together since database can merge it, so this can improve the response time. (See Reference #3 for the example of merging the index on Marital Status and Region)

**When to use it?**

**1. Low cardinality** – Some database vendors, like Oracle, provides very practical suggestion - (See Reference #3 and 4)

* If the number of distinct values of a column is less than 1% of the number of rows in the table, or if the values in a column are repeated more than 100 times, then the column is a candidate for a bitmap index.
* B-tree indexes are most effective for high-cardinality data: that is, data with many possible values, such as CUSTOMER\_NAME or PHONE\_NUMBER.
* There are 100 or more rows for each distinct value in the indexed column. When this limit is met, the bitmap index will be much smaller than a regular index, and you will be able to create the index much faster than a regular index.

**2. No or little insert/update -**

* Updating bitmap indexes take a lot of resources. Here are the suggestions: (See Reference 5)
* Building and maintaining an index structure can be expensive, and it can consume resources such as disk space, CPU, and I/O capacity. Designers must ensure that the benefits of any index outweigh the negatives of index maintenance.
* Use this simple estimation guide for the cost of index maintenance: each index maintained by an INSERT, DELETE, or UPDATE of the indexed keys requires about three times as much resource as the actual DML operation on the table. What this means is that if you INSERT into a table with three indexes, then it will be approximately 10 times slower than an INSERT into a table with no indexes. For DML, and particularly for INSERT-heavy applications, the index design should be seriously reviewed, which might require a compromise between the query and INSERT performance.

**3. Multiple Columns**

* One of the advantages is that multiple bitmap indexes can be merged and the column does not have to selective!
* More than one column in the table has an index that the optimizer can use to improve performance on a table scan. (See reference 6)
* Combining bitmap indexes on non-selective columns allows efficient AND and OR operations with a great number of rowids with minimal I/O.

**What is B-tree index**

B-tree indexes are most commonly used in a data warehouse to enforce unique keys. In many cases, it may not even be necessary to index these columns in a data warehouse, because the uniqueness was enforced as part of the preceding ETL processing, and because typical data warehouse queries may not work better with such indexes. B-tree indexes are more common in environments using third normal form schemas. In general, bitmap indexes should be more common than B-tree indexes in most data warehouse environments.

**Data Warehousing – OLAP**

Online Analytical Processing Server (OLAP) is based on multidimensional data model. It allows the managers , analysts to get insight the information through fast, consistent, interactive access to information. In this chapter we will discuss about types of OLAP, operations on OLAP, Difference between OLAP and Statistical Databases and OLTP.

**Types of OLAP Servers**

We have four types of OLAP servers that are listed below.

* Relational OLAP(ROLAP)
* Multidimensional OLAP (MOLAP)
* Hybrid OLAP (HOLAP)
* Specialized SQL Servers

**Relational OLAP(ROLAP)**

The Relational OLAP servers are placed between relational back-end server and client front-end tools. To store and manage warehouse data the Relational OLAP use relational or extended-relational DBMS.

ROLAP includes the following.

* implementation of aggregation navigation logic.
* optimization for each DBMS back end.
* additional tools and services.

**Multidimensional OLAP (MOLAP)**

Multidimensional OLAP (MOLAP) uses the array-based multidimensional storage engines for multidimensional views of data.With multidimensional data stores, the storage utilization may be low if the data set is sparse. Therefore many MOLAP Server uses the two level of data storage representation to handle dense and sparse data sets.

**Hybrid OLAP (HOLAP)**

The hybrid OLAP technique combination of ROLAP and MOLAP both. It has both the higher scalability of ROLAP and faster computation of MOLAP. HOLAP server allows to store the large data volumes of detail data. the aggregations are stored separated in MOLAP store.

**Specialized SQL Servers**

specialized SQL servers provides advanced query language and query processing support for SQL queries over star and snowflake schemas in a read-only environment.

**OLAP Operations**

As we know that the OLAP server is based on the multidimensional view of data hence we will discuss the OLAP operations in multidimensional data.

Here is the list of OLAP operations.

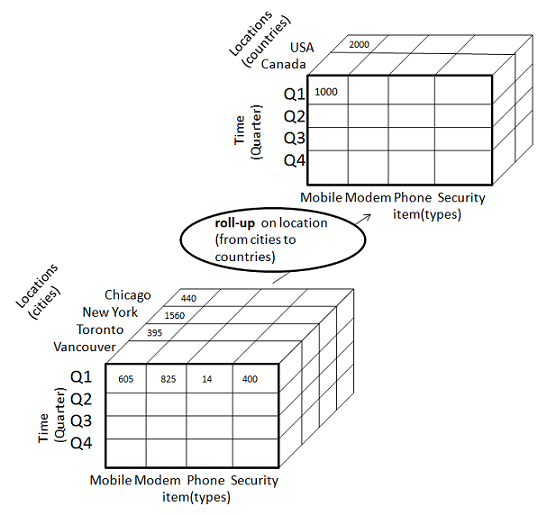
* Roll-up
* Drill-down
* Slice and dice
* Pivot (rotate)

**Roll-up**

This operation performs aggregation on a data cube in any of the following way:

* By climbing up a concept hierarchy for a dimension
* By dimension reduction.

Consider the following diagram showing the roll-up operation.



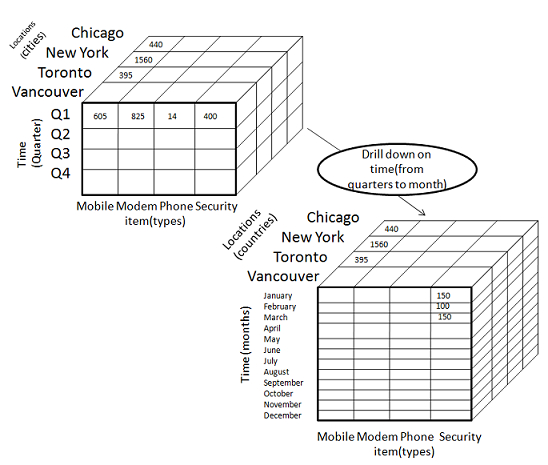
* The roll-up operation is performed by climbing up a concept hierarchy for the dimension location.
* Initially the concept hierarchy was "street < city < province < country".
* On rolling up the data is aggregated by ascending the location hierarchy from the level of city to level of country.
* The data is grouped into cities rather than countries.
* When roll-up operation is performed then one or more dimensions from the data cube are removed.

**Drill-down**

Drill-down operation is reverse of the roll-up. This operation is performed by either of the following way:

* By stepping down a concept hierarchy for a dimension.
* By introducing new dimension.

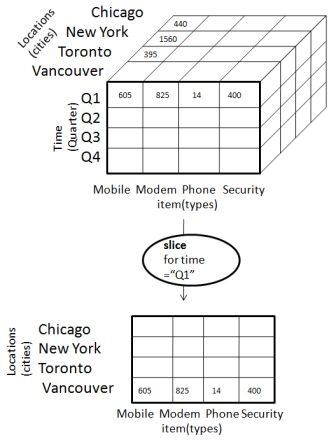
Consider the following diagram showing the drill-down operation:



* The drill-down operation is performed by stepping down a concept hierarchy for the dimension time.
* Initially the concept hierarchy was "day < month < quarter < year."
* On drill-up the time dimension is descended from the level quarter to the level of month.
* When drill-down operation is performed then one or more dimensions from the data cube are added.
* It navigates the data from less detailed data to highly detailed data.

**Slice**

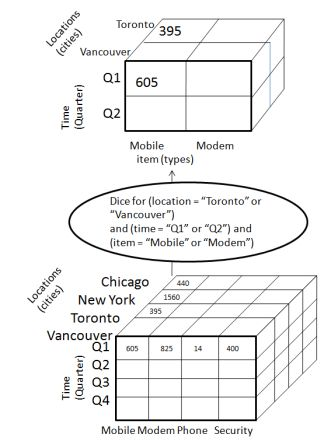
The slice operation perform selection of one dimension on a given cube and give us a new sub cube. Consider the following diagram showing the slice operation.



* The Slice operation is performed for the dimension time using the criterion time ="Q1".
* It will form a new sub cube by selecting one or more dimensions.

**Dice**

The Dice operation performs selection of two or more dimensions on a given cube and gives us a new sub-cube. Consider the following diagram showing the dice operation:

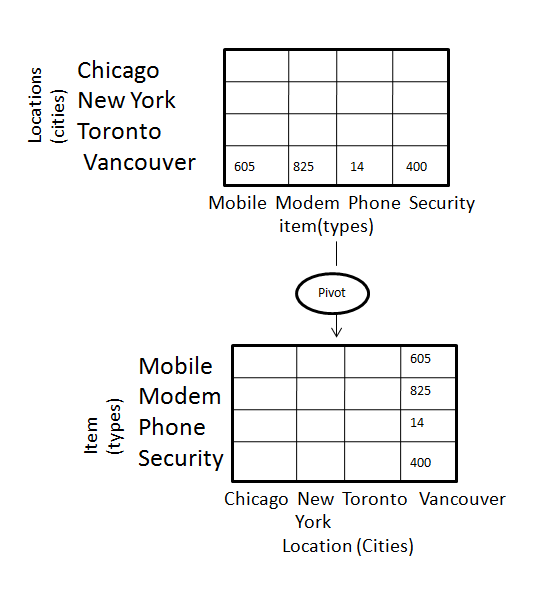


The dice operation on the cube based on the following selection criteria that involve three dimensions.

* (location = "Toronto" or "Vancouver")
* (time = "Q1" or "Q2")
* (item =" Mobile" or "Modem").

**Pivot**

The pivot operation is also known as rotation.It rotates the data axes in view in order to provide an alternative presentation of data.Consider the following diagram showing the pivot operation.



In this the item and location axes in 2-D slice are rotated.

**OLAP v/s OLTP**

|  |  |  |
| --- | --- | --- |
| **SN** | **Data Warehouse (OLAP)** | **Operational Database(OLTP)** |
| 1 | This involves historical processing of information. | This involves day to day processing. |
| 2 | OLAP systems are used by knowledge workers such as executive, manager and analyst. | OLTP systems are used by clerk, DBA, or database professionals. |
| 3 | This is used to analysis the business. | This is used to run the business. |
| 4 | It focuses on Information out. | It focuses on Data in. |
| 5 | This is based on Star Schema, Snowflake Schema and Fact Constellation Schema. | This is based on Entity Relationship Model. |
| 6 | It focuses on Information out. | This is application oriented. |
| 7 | This contains historical data. | This contains current data. |
| 8 | This provides summarized and consolidated data. | This provide primitive and highly detailed data. |
| 9 | This provides summarized and multidimensional view of data. | This provides detailed and flat relational view of data. |
| 10 | The number or users are in Hundreds. | The numbers of users are in thousands. |
| 11 | The number of records accessed is in millions. | The number of records accessed are in tens. |
| 12 | The database size is from 100GB to TB | The database size is from 100 MB to GB. |
| 13 | These are highly flexible. | This provides high performance. |

**Architectural difference between Informatica 9 and Informatica 8.x**

Architecture wise there are no differences between 8 and 9 but here are some new features added in powercenter 9   
  
**Update Lookup Cache :** Cache updates. We can update the lookup cache based on the results of an expression. When an expression is true, We can add to or update the lookup cache. We can update the dynamic lookup cache with the results of an expression.   
**Lookup return Multiple rows:** We can configure the Lookup transformation to return all rows that match a lookup condition. A Lookup transformation is an active transformation when it can return more than one row for any given input row.   
**SQL overrides for uncached lookups:** In previous versions We could create a SQL override for cached lookups only. We can create an SQL override for uncached lookup. We can include lookup ports in the SQL query.   
**Database deadlock resilience:** In previous releases, when the Integration Service encountered a database deadlock during a lookup, the session failed. Effective in 9.0, the session will not fail. When a deadlock occurs, the Integration Service attempts to run the last statement in a lookup. We can configure the number of retry attempts and time period between attempts.   
**SQL transformation:** Auto-commit for connections. We can enable auto-commit for each database connection. Each SQL statement in a query defines a transaction. A commit occurs when the SQL statement completes or the next statement is executed, whichever comes first.   
**Session Log files rollover:** We can limit the size of session logs for real-time sessions. We can limit the size by time or by file size. We can also limit the number of log files for a session.   
**Passive transformation:** We can configure the SQL transformation to run in passive mode instead of active mode. When the SQL transformation runs in passive mode, the SQL transformation returns one output row for each input row.   
**XML transformation:** XML Parser buffer validation. The XML Parser transformation can validate an XML document against a schema. The XML Parser transformation routes invalid XML to an error port. When the XML is not valid, the XML Parser transformation routes the XML and the error messages to a separate output group that We can connect to a target.   
**Model Repository Service:** Application service that manages the Model repository. The Model repository is a relational database that stores the metadata for projects created in Informatica Analyst and Informatica Designer. The Model repository also stores run-time and configuration information for applications deployed to a Data.   
**Integration Service:** Create and enable a Model Repository Service on the Domain tab of Informatica Administrator.   
**Connection management:** Database connections are centralized in the domain. We can create and view database connections in Informatica Administrator, Informatica Developer, or Informatica Analyst. Create, view, edit, and grant permissions on database connections in Informatica Administrator.   
**Deployment:** We can deploy, enable, and configure deployment units in the Informatica Administrator. Deploy Deployment units to one or more Data Integration Services. Create deployment units in Informatica Developer.   
**Monitoring:** We can monitor profile jobs, scorecard jobs, preview jobs, mapping jobs, and SQL Data Services for each Data Integration Service. View the status of each monitored object on the Monitoring tab of Informatica Administrator.

**What is ODS**

ODS is used to store the real-time data from OLTP systems.

This data is used for limited time period only.

It is a database designed to integrate data from multiple sources for additional operations of the data. And this data is taken from ETL  and load the data in it and used for the DWH or Data Marts.

And this data can be used for reporting and doing analysis purpose because it contains the real time data to access easily.

In some companies this may be used or may not be used

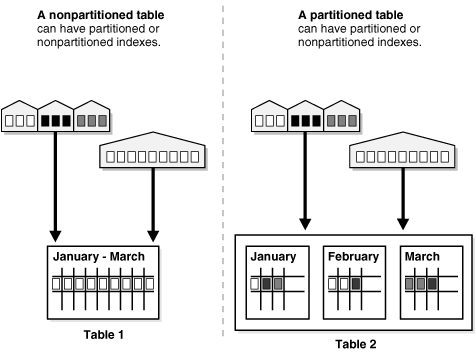
**Basics of Partitioning**

Partitioning allows a table, index, or index-organized table to be subdivided into smaller pieces, where each piece of such a 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 partitioned objects. However, from the perspective of the application, a partitioned table is identical to a non-partitioned table; no modifications are necessary when accessing a partitioned table using SQL queries and DML statements.

[Figure 2-1](https://docs.oracle.com/cd/B28359_01/server.111/b32024/partition.htm#i461710) offers a graphical view of how partitioned tables differ from non-partitioned tables.

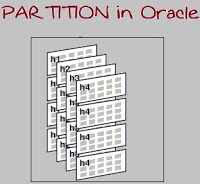
Figure 2-1 A View of Partitioned Tables



**What is a PARTITION in Oracle?Why to use Partition And Types of Partitions**

### PARTITIONS

Partitioning allows tables, indexes, and index-organized tables to be subdivided into smaller pieces, enabling these database objects to be managed and accessed at a finer level of granularity.

[](http://4.bp.blogspot.com/-6fOLLLqDSd0/UhhP8LeOD8I/AAAAAAAAAlY/NMZ0wvy28Eo/s1600/partion.jpg)

### When to Partition a Table??

* Tables greater than 2 GB should always be considered as candidates for partitioning.
* Tables containing historical data, in which new data is added into the newest partition. A typical example is a historical table where only the current month's data is updatable and the other 11 months are read only.
* When the contents of a table need to be distributed across different types of storage devices.

### TYPES

1     Range partitions

2     List partitions

3     Hash partitions

4     Sub partitions

### ADVANTAGES OF PARTITIONS

* Reducing downtime for scheduled maintenance, which allows maintenance operations to be carried out on selected partitions while other partitions are available to users.
* Reducing downtime due to data failure, failure of a particular partition will no way affect other partitions.
* Partition independence allows for concurrent use of the various partitions for various purposes.

 What is the advantage of partitions, by storing them in different Tablespaces??

1     Reduces the possibility of data corruption in multiple partitions.

2     Backup and recovery of each partition can be done independently.

### Partitioning Key

Each row in a partitioned table is unambiguously assigned to a single partition. The partitioning key is comprised of one or more columns that determine the partition where each row will be stored

\* Any table can be partitioned into a million separate partitions except those tables containing columns with LONG or LONG RAW datatypes. You can, however, use tables containing columns with CLOB or BLOB datatypes.

### 1.RANGE PARTITIONS

Definition: A table that is partitioned by range is partitioned in such a way that each partition contains rows for which the partitioning expression value lies within a given range.

 Creating range partitioned table

SQL> Create table Employee(emp\_no number(2),emp\_name varchar(2)) partition by range(emp\_no) (partition p1 values less than(100), partition p2 values less than(200), partition p3 values less than(300),partition p4 values less than(maxvalue));

Inserting records into range partitioned table

SQL> Insert into Employee values(101,’a’); -- this will go to p1

SQL> Insert into Employee values(201,’b’); -- this will go to p2

SQL> Insert into Employee values(301,’c’); -- this will go to p3

SQL> Insert into Employee values(401,’d’); -- this will go to p4

 Selecting records from range partitioned table

SQL> Select \*from Employee;

SQL> Select \*from Employee partition(p1);

 Adding a partition

SQL> Alter table Employee add partition p5 values less than(400);

Dropping a partition

SQL> Alter table Employee drop partition p1;

Renaming a partition

SQL> Alter table Employee rename partition p3 to p6;

Truncate a partition

SQL> Alter table Employee truncate partition p5;

Splitting a partition

SQL> Alter table Employee split partition p2 at(120) into (partition p21,partition p22);

Exchanging a partition

SQL> Alter table Employee exchange partition p2 with table Employee\_x;

Moving a partition

SQL> Alter table Employee move partition p21 tablespace ABC\_TBS;

### 2. LIST PARTITIONS

Definition: List partitioning enables you to explicitly control how rows map to partitions by specifying a list of discrete values for the partitioning key in the description for each partition.

 Creating list partitioned table

SQL> Create table Employee (Emp\_no number(2),Emp\_name varchar(2)) partition by list(Emp\_no) (partition p1 values(1,2,3,4,5), partition p2 values(6,7,8,9,10),partition p3 values(11,12,13,14,15), partition p4 values(16,17,18,19,20));

Inserting records into list partitioned table

SQL> Insert into Employee values(4,’xxx’); -- this will go to p1

SQL> Insert into Employee values(8,’yyy’); -- this will go to p2

SQL> Insert into Employee values(14,’zzz’); -- this will go to p3

SQL> Insert into Employee values(19,’bbb’); -- this will go to p4

 Selecting records from list partitioned table

SQL> Select \*from Employee;

SQL> Select \*from Employee partition(p1);

Adding a partition

SQL> Alter table Employee add partition p5 values(21,22,23,24,25);

Dropping a partition

SQL> Alter table Employee drop partition p5;

Renaming a partition

SQL> Alter table Employee rename partition p5to p1;

Truncate a partition

SQL> Alter table Employee truncate partition p5;

Exchanging a partition

SQL> Alter table Employee exchange partition p1 with table Employee\_x;

Moving a partition

SQL> Alter table Employee move partition p2 tablespace ABC\_TBS;

### 3. HASH PARTITIONS

Definition:Hash partitioning maps data to partitions based on a hashing algorithm that Oracle applies to the partitioning key that you identify.

Creating hash partitioned table

SQL> Create table Employee(emp\_no number(2),emp\_name varchar(2)) partition by hash(emp\_no) partitions 5;

Here oracle automatically gives partition names like

SYS\_P1

SYS\_P2

SYS\_P3

SYS\_P4

SYS\_P5

Inserting records into hash partitioned table(based on hash function)

SQL> Insert into Employee values(5,’a’);

SQL> Insert into Employee values(8,’b’);

SQL> Insert into Employee values(14,’c’);

SQL> Insert into Employee values(19,’d’);

Selecting records from hash partitioned table

SQL> Select \*from Employee;

SQL> Select \*from Employee partition(SYS\_P2);

Adding a partition

SQL> Alter table Employee add partition p9;

Renaming a partition

SQL> Alter table Employee rename partition p9 to p10;

Truncate a partition

SQL> Alter table Employee truncate partition p9;

Exchanging a partition

SQL> Alter table Employee exchange partition SYS\_P1 with table Employee\_X;

Moving a partition

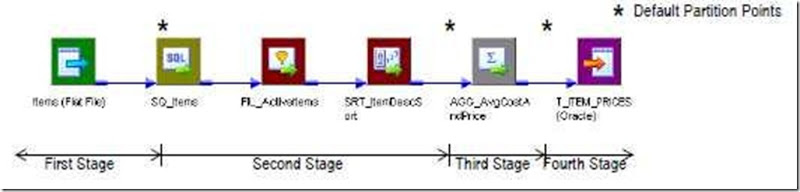
SQL> Alter table Employee move partition SYS\_P1 tablespace ABC\_TBS;

**Informatica Partitioning**

**PARTITIONING ATTRIBUTES**

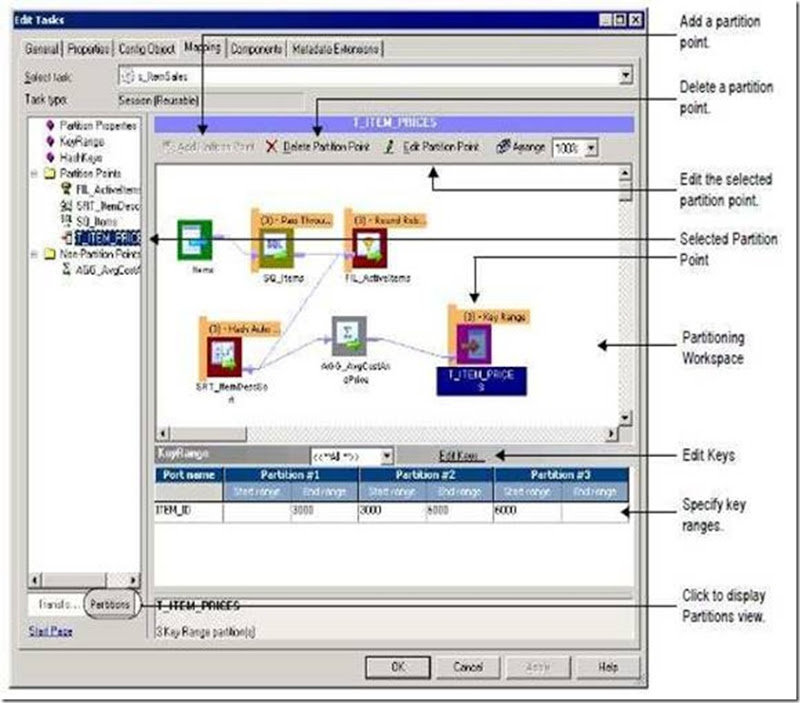
1. Partition points

* By default, IS sets partition points at various transformations in the pipeline.
* Partition points mark thread boundaries and divide the pipeline into stages.
* A stage is a section of a pipeline between any two partition points.

[](http://lh3.ggpht.com/_MbhSjEtmzI8/TapQ7ozy5HI/AAAAAAAAALI/fqfaVfWcuzg/s1600-h/clip_image001%5b2%5d.jpg)

### 2. Number of Partitions

* We can define up to 64 partitions at any partition point in a pipeline.
* When we increase or decrease the number of partitions at any partition point, the Workflow Manager increases or decreases the number of partitions at all Partition points in the pipeline.
* Increasing the number of partitions or partition points increases the number of threads.
* The number of partitions we create equals the number of connections to the source or target. For one partition, one database connection will be used.

[](http://lh3.ggpht.com/_MbhSjEtmzI8/TapQ9DLYfxI/AAAAAAAAALQ/N44iRKoz0fw/s1600-h/clip_image003%5b1%5d.jpg)[clip_image006](http://lh5.ggpht.com/_MbhSjEtmzI8/TapQ-kl8ZOI/AAAAAAAAALY/YbKFtydQ-50/s1600-h/clip_image0065.gif)[clip_image007](http://lh3.ggpht.com/_MbhSjEtmzI8/TapQ_WYbjjI/AAAAAAAAALg/CSTesptpjWQ/s1600-h/clip_image0075.gif)[clip_image011](http://lh5.ggpht.com/_MbhSjEtmzI8/TapRAVP1rVI/AAAAAAAAALo/yGRRj-aZ8vc/s1600-h/clip_image0115.gif)[clip_image027](http://lh3.ggpht.com/_MbhSjEtmzI8/TapRBWxTPVI/AAAAAAAAALw/5-VEZuT22uY/s1600-h/clip_image0275.gif)

### 3. Partition types

* The Integration Service creates a default partition type at each partition point.
* If we have the Partitioning option, we can change the partition type. This option is purchased separately.
* The partition type controls how the Integration Service distributes data among partitions at partition points.

[clip_image007[1]](http://lh4.ggpht.com/_MbhSjEtmzI8/TapRCYM8H7I/AAAAAAAAAL4/ar-Lv-o43MM/s1600-h/clip_image00712.gif)[clip_image010[1]](http://lh4.ggpht.com/_MbhSjEtmzI8/TapRDZFQG9I/AAAAAAAAAMA/yhpdMJzTPR0/s1600-h/clip_image01012.gif)[clip_image029[1]](http://lh5.ggpht.com/_MbhSjEtmzI8/TapREFoicbI/AAAAAAAAAMI/99FzzFpTxfg/s1600-h/clip_image02912.gif)

### PARTITIONING TYPES

### 1. Round Robin Partition Type

* In round-robin partitioning, the Integration Service distributes rows of data evenly to all partitions.
* Each partition processes approximately the same number of rows.
* Use round-robin partitioning when we need to distribute rows evenly and do not need to group data among partitions.

### 2. Pass-Through Partition Type

* In pass-through partitioning, the Integration Service processes data without Redistributing rows among partitions.
* All rows in a single partition stay in that partition after crossing a pass-Through partition point.
* Use pass-through partitioning when we want to increase data throughput, but we do not want to increase the number of partitions.

### 3. Database Partitioning Partition Type

* Use database partitioning for Oracle and IBM DB2 sources and IBM DB2 targets only. Use any number of pipeline partitions and any number of database partitions.
* We can improve performance when the number of pipeline partitions equals the number of database partitions.

### Database Partitioning with One Source

When we use database partitioning with a source qualifier with one source, the Integration Service generates SQL queries for each database partition and distributes the data from the database partitions among the session partitions Equally. For example, when a session has three partitions and the database has five partitions, 1st and 2nd session partitions will receive data from 2 database partitions each. Thus four DB partitions used. 3rd Session partition will receive Data from the remaining 1 DB partition.

### Partitioning a Source Qualifier with Multiple Sources Tables

### The Integration Service creates SQL queries for database partitions based on the Number of partitions in the database table with the most partitions. If the session has three partitions and the database table has two partitions, one of the session partitions receives no data.

### 4. Hash Auto-Keys Partition Type

* The Integration Service uses all grouped or sorted ports as a compound Partition key. Use hash auto-keys partitioning at or before Rank, Sorter, Joiner, and Unsorted Aggregator transformations to ensure that rows are grouped Properly before they enter these transformations.

### 5. Hash User-Keys Partition Type

* The Integration Service uses a hash function to group rows of data among Partitions. we define the number of ports to generate the partition key.
* we choose the ports that define the partition key .

### 6. Key range Partition Type

* We specify one or more ports to form a compound partition key. The Integration Service passes data to each partition depending on the Ranges we specify for each port. Use key range partitioning where the sources or targets in the pipeline are Partitioned by key range.
* Example: Customer 1-100 in one partition, 101-200 in another and so on. We Define the range for each partition.

**OLAP vs OLTP:**

OLPT and OLAP are complementing technologies. You can't live without OLTP: it runs your business day by day. So, using getting strategic information from OLTP is usually first “quick and dirty” approach, but can become limiting later.

This post explores key differences between two technologies.

[OLTP](http://en.wikipedia.org/wiki/Online_analytical_processing) stands for On Line Transaction Processing and is a data modeling approach typically used to facilitate and manage usual business applications. Most of applications you see and use are OLTP based.

[OLAP](http://en.wikipedia.org/wiki/OLTP) stands for On Line Analytic Processing and is an approach to answer multi-dimensional queries. OLAP was conceived for Management Information Systems and Decision Support Systems but is still widely underused: every day I see too much people making out business intelligence from OLTP data!

With the constant growth of data analysis and business intelligence applications (now even in small business) understanding OLAP nuances and benefits is a must if you want provide valid and useful analytics to management.

The following table summarized main differences between OLPT and OLAP:

|  |  |  |
| --- | --- | --- |
|  | OLTP | OLAP |
| Application | Operational: ERP, CRM, legacy apps, ... | Management Information System, Decision Support System |
| Typical users | Staff | Managers, Executives |
| Horizon | Weeks, Months | Years |
| Refresh | Immediate | Periodic |
| Data model | Entity-relationship | Multi-dimensional |
| Schema | Normalized | Star |
| Emphasis | Update | Retrieval |

**What is schema**

A schema is a collection of logical structures of data, or schema objects. A schema is owned by a database user and has the same name as that user. Each user owns a single schema. Schema objects can be created and manipulated with SQL and include the following types of objects:

* Clusters
* Database links
* Database triggers
* Dimensions
* External procedure libraries
* Indexes and index types
* Java classes, Java resources, and Java sources
* Materialized views and materialized view logs
* Object tables, object types, and object views
* Operators
* Sequences
* Stored functions, procedures, and packages
* Synonyms
* Tables and index-organized tables
* Views

**Mapping -** represents the flow and transformation of data from source to taraget.   
**Mapplet -** a group of transformations that can be called within a mapping.   
**Session -** a task associated with a mapping to define the connections and other configurations for that mapping.   
**Workflow -** controls the execution of tasks such as commands, emails and sessions.   
**Worklet -** a workflow that can be called within a workflow.

**What is session**

A session is a set of instructions that tells the Informatica Server how and when to move data from sources to targets. You create and maintain sessions in the Server Manager.

When you create a session, you enter general information such as the session name, session schedule, and the Informatica Server to run the session. You can also select options to execute pre-session shell commands, send post-session email, and FTP source and target files. Using session properties, you can also override parameters established in the mapping, such as source and target location, source and target type, error tracing levels, and transformation attributes. For details on server activity while executing a session.

You can group sessions into a batch. The Informatica Server can run the sessions in a batch in sequential order, or start them concurrently. Some batch settings override session settings.

Once you create a session, you can use either the Server Manager or the command line program pmcmd to start or stop the session. You can also use the Server Manager to monitor, edit, schedule, abort, copy, and delete the session.

Informatica

**Reuseable Transformation in informatica**

Using Reusable Transformation is considered as a Good Practice as it   
reduces time for the developer. As far as the Informatica is concerned,   
there won’t be any performance difference between the Reusable and   
Non-Reusable component. If you drill down more finer, the repository tables   
will have only one entry instead of multiple entries in case of   
non-reusable. You can consider this as the increase in performance while   
using reusable though its marginal or none.   
  
Mapplet by default is reusable. You need not use reusable Tx to create a   
Mapplet. Its perfectly alright even if you use a reusable Tx. Hope You are   
clear now.

Property Make Reusable

1. Expression 9) Rank
2. Update Stratergy 10) Sorter
3. Union 11) Router
4. SQL 12) Filter
5. Store Procedure 13) Lookup
6. Aggregator 14) Java
7. Normalizer
8. Custom Transformation

**Non Reuseable Transformation**

1. Source Qualifier

**Types of Tasks in Informatica**

There are different types of tasks in Informatica Workflow manager which we use while running a workflow.I have listed it below.

|  |  |
| --- | --- |
| **Tasks** | **Description** |
| Assignment | Used to assign a value to a workflow variable |
| Command | Used to run a shell command during the workflow |
| Control | Used to stop or abort the workflow |
| Decision | Tells a condition to evaluate |
| Email | Used to send email during the workflow |
| Event-Raise | Notifies the Event-Wait task that an event has occurred |
| Event-Wait | It waits for the event to completed in order to start the next task |
| Session | Used to run the mapping created in Designer by linking to session |
| Timer | It waits for a already timed event to start |

**Data Mart :**

The data mart is a subset of the data warehouse that is usually oriented to a specific business line or teamData marts improve end-user response time by allowing users to have access to the specific type of data they need to view most often by providing the data in a way that supports the collective view of a group of users Each data mart is dedicated to a specific business function or region.

* Often holds only one subject area- for example, Finance, or Sales
* May hold more summarized data (although many hold full detail)
* Concentrates on integrating information from a given subject area or set of source systems
* Is built focused on a dimensional model using a star schema.

**Reasons for creating Data Marts :**

* Easy access to frequently needed data
* Creates collective view by a group of users
* Improves end-user response time
* Ease of creation
* Lower cost than implementing a full data warehouse
* Potential users are more clearly defined than in a full data warehouse
* Contains only business essential data and is less cluttered.

**Data Mining:**

-The analysis step of the "Knowledge Discovery in Databases" process, or KDD

**-** The overall goal of the data mining process is to extract information from a data set and transform it into an understandable structure for further use

-