**Data Warehouse - Overview**

A Data Warehouse consists of data from **multiple heterogeneous data sources** and is used for analytical reporting and decision making. Data Warehouse is a central place where data is stored from different data sources and applications.

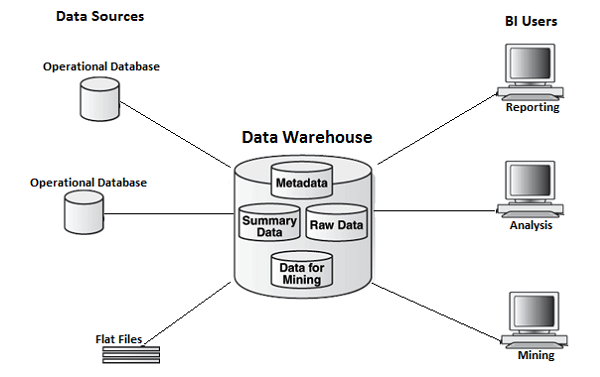
The term Data Warehouse was first invented by Bill Inmom in 1990. A Data Warehouse is always kept separate from an Operational Database.

The data in a DW system is loaded from operational transaction systems like −

* Sales
* Marketing
* HR
* SCM, etc.

It may pass through operational data store or other transformations before it is loaded to the DW system for information processing.

A Data Warehouse is used for reporting and analyzing of information and stores both historical and current data. The data in DW system is used for Analytical reporting, which is later used by Business Analysts, Sales Managers or Knowledge workers for decision-making.



In the above image, you can see that the data is coming from **multiple heterogeneous data** sources to a Data Warehouse. Common data sources for a data warehouse includes −

* Operational databases
* SAP and non-SAP Applications
* Flat Files (xls, csv, txt files)

Data in data warehouse is accessed by BI (Business Intelligence) users for Analytical Reporting, Data Mining and Analysis. This is used for decision making by Business Users, Sales Manager, Analysts to define future strategy.

**Features of a Data Warehouse**

It is a central data repository where data is stored from one or more heterogeneous data sources. A DW system stores both current and historical data. Normally a DW system stores 5-10 years of historical data. A DW system is always kept separate from an operational transaction system.

The data in a DW system is used for different types of analytical reporting range from Quarterly to Annual comparison.

**Data Warehouse Vs Operational Database**

The differences between a Data Warehouse and Operational Database are as follows −

* An **Operational System** is designed for known workloads and transactions like updating a user record, searching a record, etc. However, Data Warehouse transactions are more complex and present a general form of data.
* An **Operational System** contains the current data of an organization and Data warehouse normally contains the historical data.
* An **Operational Database** supports parallel processing of multiple transactions. Concurrency control and recovery mechanisms are required to maintain consistency of the database.
* An **Operational Database** query allows to read and modify operations (insert, delete and Update) while an OLAP query needs only read-only access of stored data (Select statement).

**Architecture of Data Warehouse**

Data Warehousing involves data cleaning, data integration, and data consolidations. A Data Warehouse has a 3-layer architecture −

**Data Source Layer**

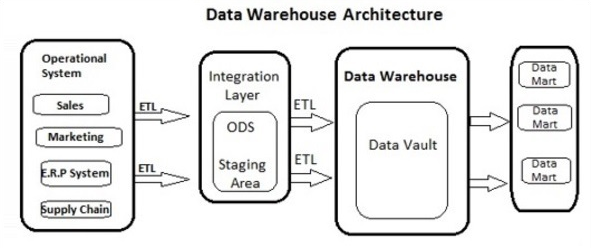
It defines how the data comes to a Data Warehouse. It involves various data sources and operational transaction systems, flat files, applications, etc.

**Integration Layer**

It consists of Operational Data Store and Staging area. Staging area is used to perform data cleansing, data transformation and loading data from different sources to a data warehouse. As multiple data sources are available for extraction at different time zones, staging area is used to store the data and later to apply transformations on data.

**Presentation Layer**

This is used to perform BI reporting by end users. The data in a DW system is accessed by BI users and used for reporting and analysis.

The following illustration shows the common architecture of a Data Warehouse System. 

**Characteristics of a Data Warehouse**

The following are the key characteristics of a Data Warehouse −

* **Subject Oriented** − In a DW system, the data is categorized and stored by a business subject rather than by application like equity plans, shares, loans, etc.
* **Integrated** − Data from multiple data sources are integrated in a Data Warehouse.
* **Non Volatile** − Data in data warehouse is non-volatile. It means when data is loaded in DW system, it is not altered.
* **Time Variant** − A DW system contains historical data as compared to Transactional system which contains only current data. In a Data warehouse you can see data for 3 months, 6 months, 1 year, 5 years, etc.

**OLTP vs OLAP**

Firstly, OLTP stands for **Online Transaction Processing**, while OLAP stands for **Online Analytical Processing**

In an OLTP system, there are a large number of short online transactions such as INSERT, UPDATE, and DELETE.

Whereas, in an OLTP system, an effective measure is the processing time of short transactions and is very less. It controls data integrity in multi-access environments. For an OLTP system, the number of transactions per second measures the effectiveness. An OLTP Data Warehouse System contains current and detailed data and is maintained in the schemas in the entity model (3NF).

**For Example** −

A Day-to-Day transaction system in a retail store, where the customer records are inserted, updated and deleted on a daily basis. It provides faster query processing. OLTP databases contain detailed and current data. The schema used to store OLTP database is the Entity model.

In an OLAP system, there are lesser number of transactions as compared to a transactional system. The queries executed are complex in nature and involves data aggregations.

**What is an Aggregation?**

We save tables with aggregated data like yearly (1 row), quarterly (4 rows), monthly (12 rows) or so, if someone has to do a year to year comparison, only one row will be processed. However, in an un-aggregated table it will compare all the rows. This is called Aggregation.

There are various Aggregation functions that can be used in an OLAP system like Sum, Avg, Max, Min, etc.

**For Example** −

SELECT Avg(salary)

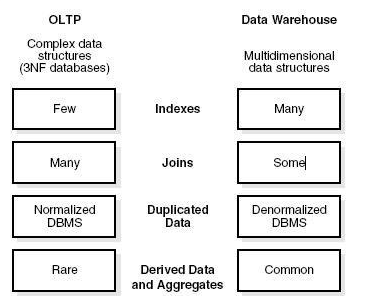
FROM employee

WHERE title = 'Programmer';

**Key Differences**

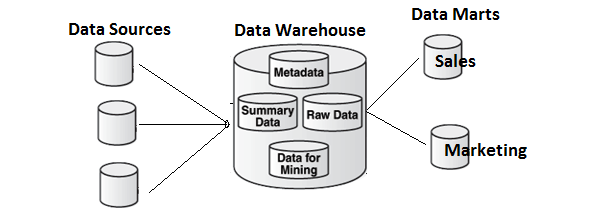
These are the major differences between an OLAP and an OLTP system.

* **Indexes** − An OLTP system has only few indexes while in an OLAP system there are many indexes for performance optimization.
* **Joins** − In an OLTP system, large number of joins and data are normalized. However, in an OLAP system there are less joins and are de-normalized.
* **Aggregation** − In an OLTP system, data is not aggregated while in an OLAP database more aggregations are used.
* **Normalization** − An OLTP system contains normalized data however data is not normalized in an OLAP system.



**Data Mart Vs Data Warehouse**

Data mart focuses on a single functional area and represents the simplest form of a Data Warehouse. Consider a Data Warehouse that contains data for Sales, Marketing, HR, and Finance. A Data mart focuses on a single functional area like Sales or Marketing.



In the above image, you can see the difference between a Data Warehouse and a data mart.

**Fact vs Dimension Table**

A fact table represents the measures on which analysis is performed. It also contains foreign keys for the dimension keys.

**For example** − Every sale is a fact.

|  |  |  |  |
| --- | --- | --- | --- |
| **Cust Id** | **Prod Id** | **Time Id** | **Qty Sold** |
| 1110 | 25 | 2 | 125 |
| 1210 | 28 | 4 | 252 |

The Dimension table represents the characteristics of a dimension. A Customer dimension can have Customer\_Name, Phone\_No, Sex, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **Cust Id** | **Cust\_Name** | **Phone** | **Sex** |
| 1110 | Sally | 1113334444 | F |
| 1210 | Adam | 2225556666 | M |

**Data Warehouse - Schemas**

A schema is defined as a logical description of database where fact and dimension tables are joined in a logical manner. Data Warehouse is maintained in the form of Star, Snow flakes, and Fact Constellation schema.

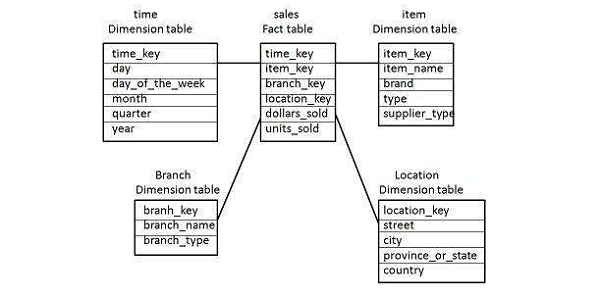
**Star Schema**

A Star schema contains a fact table and multiple dimension tables. Each dimension is represented with only one-dimension table and they are not normalized. The Dimension table contains a set of attributes.

**Characteristics**

* In a Star schema, there is only one fact table and multiple dimension tables.
* In a Star schema, each dimension is represented by one-dimension table.
* Dimension tables are not normalized in a Star schema.
* Each Dimension table is joined to a key in a fact table.

The following illustration shows the sales data of a company with respect to the four dimensions, namely Time, Item, Branch, and Location.



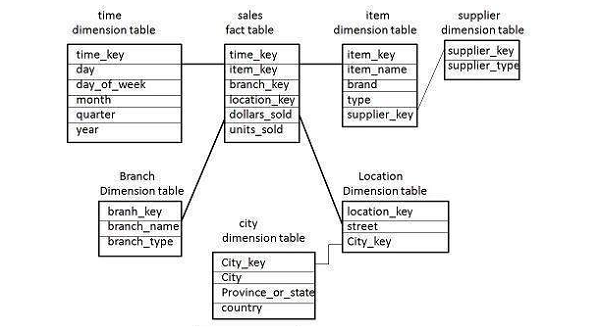
There is a fact table at the center. It contains the keys to each of four dimensions. The fact table also contains the attributes, namely dollars sold and units sold.

**Note** − Each dimension has only one-dimension table and each table holds a set of attributes. For example, the location dimension table contains the attribute set {location\_key, street, city, province\_or\_state, country}. This constraint may cause data redundancy.

**For example** − "Vancouver" and "Victoria" both the cities are in the Canadian province of British Columbia. The entries for such cities may cause data redundancy along the attributes province\_or\_state and country.

**Snowflakes Schema**

Some dimension tables in the Snowflake schema are normalized. The normalization splits up the data into additional tables as shown in the following illustration.



Unlike in the Star schema, the dimension’s table in a snowflake schema are normalized.

**For example** − The item dimension table in a star schema is normalized and split into two dimension tables, namely item and supplier table. Now the item dimension table contains the attributes item\_key, item\_name, type, brand, and supplier-key.

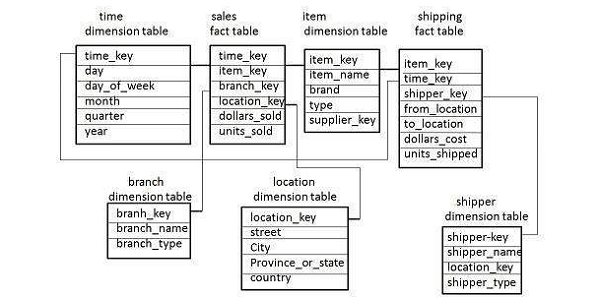
The supplier key is linked to the supplier dimension table. The supplier dimension table contains the attributes supplier\_key and supplier\_type.

**Note** − Due to the normalization in the Snowflake schema, the redundancy is reduced and therefore, it becomes easy to maintain and the save storage space.

**Fact Constellation Schema (Galaxy Schema)**

A fact constellation has multiple fact tables. It is also known as a Galaxy Schema.

The following illustration shows two fact tables, namely Sales and Shipping –



The sales fact table is the same as that in the Star Schema. The shipping fact table has five dimensions, namely item\_key, time\_key, shipper\_key, from\_location, to\_location. The shipping fact table also contains two measures, namely dollars sold and units sold. It is also possible to share dimension tables between fact tables.

**For example** − Time, item, and location dimension tables are shared between the sales and shipping fact table.

**Data Warehouse - ETL & Reporting Tools**

An ETL tool extracts the data from all these heterogeneous data sources, transforms the data (like applying calculations, joining fields, keys, removing incorrect data fields, etc.), and loads it into a Data Warehouse.

**Extraction**

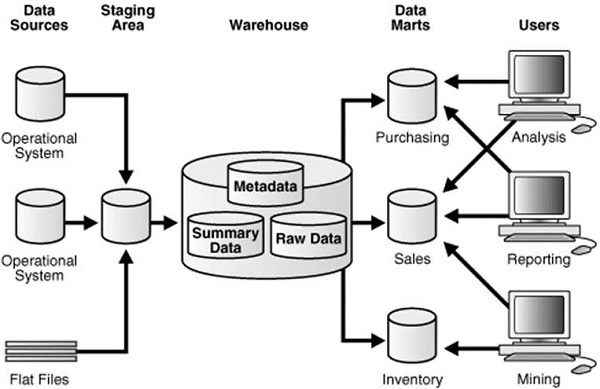
A staging area is required during the ETL load. There are various reasons why staging area is required. The source systems are only available for specific period of time to extract data. This period of time is less than the total data-load time. Therefore, staging area allows you to extract the data from the source system and keeps it in the staging area before the time slot ends.

The staging area is required when you want to get the data from multiple data sources together or if you want to join two or more systems together.

**For example** − You will not be able to perform an SQL Query joining two tables from two physically different databases.

The data extractions’ time slot for different systems vary as per the time zone and operational hours. The data extracted from the source systems can be used in multiple Data Warehouse Systems, Operation Data Stores, etc.

ETL allows you to perform complex transformations and requires extra area to store the data.



**Transform**

In data transformation, you apply a set of functions on extracted data to load it into the target system. The data that does not require any transformation is known as a direct move or pass through data.

You can apply different transformations on extracted data from the source system. For example, you can perform customized calculations. If you want sum-of-sales revenue and this is not in database, you can apply the SUM formula during transformation and load the data.

**For example** − If you have the first name and the last name in a table in different columns, you can use concatenate before loading.

**Load**

During the Load phase, data is loaded into the end-target system and it can be a flat file or a Data Warehouse system.

Online Analytical Processing Server (OLAP) is based on the multidimensional data model. It allows managers, and analysts to get an insight of the information through fast, consistent, and interactive access to information. This chapter cover the 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 −

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

**Relational OLAP**

ROLAP servers are placed between relational back-end server and client front-end tools. To store and manage warehouse data, ROLAP uses 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 uses 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 use two levels of data storage representation to handle dense and sparse data sets.

**Hybrid OLAP**

Hybrid OLAP is a combination of both ROLAP and MOLAP. It offers higher scalability of ROLAP and faster computation of MOLAP. HOLAP servers allows to store the large data volumes of detailed information. The aggregations are stored separately in MOLAP store.

**Specialized SQL Servers**

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

**OLAP Operations**

Since OLAP servers are based on multidimensional view of data, we will discuss OLAP operations in multidimensional data.

Here is the list of OLAP operations −

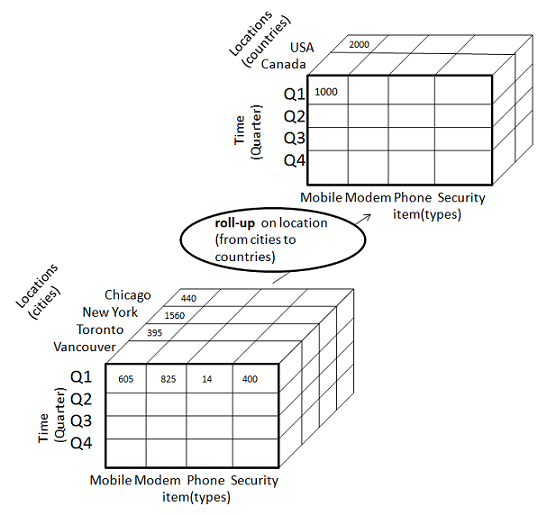
* Roll-up
* Drill-down
* Slice and dice
* Pivot (rotate)

**Roll-up**

Roll-up performs aggregation on a data cube in any of the following ways −

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

The following diagram illustrates how roll-up works.



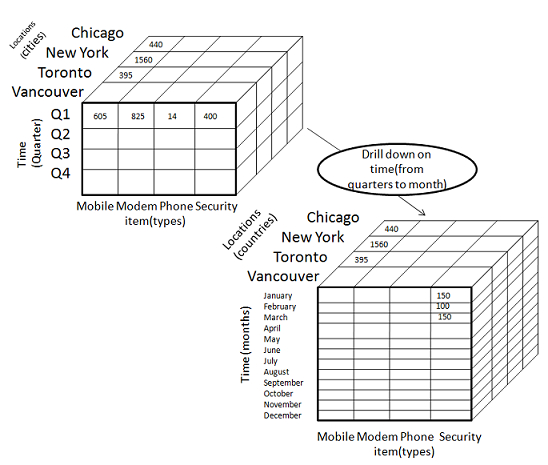
* Roll-up is performed by climbing up a concept hierarchy for the dimension location.
* Initially the concept hierarchy was "street < city < province < country".
* On rolling up, the data is aggregated by ascending the location hierarchy from the level of city to the level of country.
* The data is grouped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

**Drill-down**

Drill-down is the reverse operation of roll-up. It is performed by either of the following ways −

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

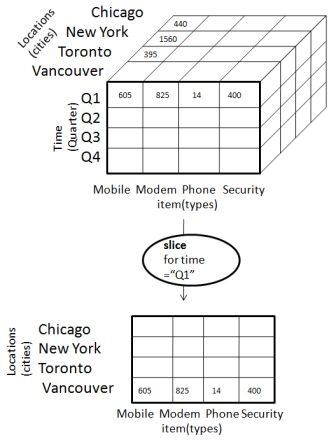
The following diagram illustrates how drill-down works −



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

**Slice**

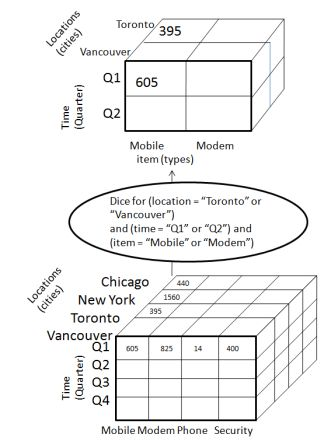
The slice operation selects one particular dimension from a given cube and provides a new sub-cube. Consider the following diagram that shows how slice works.



* Here Slice 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**

Dice selects two or more dimensions from a given cube and provides a new sub-cube. Consider the following diagram that shows the dice operation.

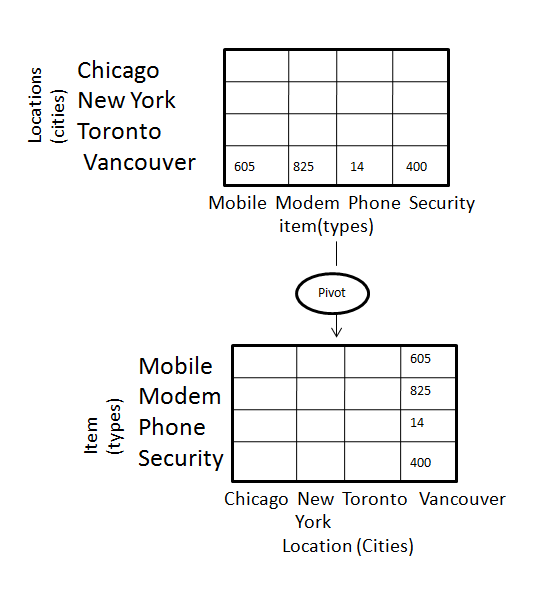


The dice operation on the cube based on the following selection criteria involves three dimensions.

* (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 that shows the pivot operation.



**OLAP vs OLTP**

|  |  |  |
| --- | --- | --- |
| **Sr.No.** | **Data Warehouse (OLAP)** | **Operational Database (OLTP)** |
| 1 | Involves historical processing of information. | Involves day-to-day processing. |
| 2 | OLAP systems are used by knowledge workers such as executives, managers and analysts. | OLTP systems are used by clerks, DBAs, or database professionals. |
| 3 | Useful in analyzing the business. | Useful in running the business. |
| 4 | It focuses on Information out. | It focuses on Data in. |
| 5 | Based on Star Schema, Snowflake, Schema and Fact Constellation Schema. | Based on Entity Relationship Model. |
| 6 | Contains historical data. | Contains current data. |
| 7 | Provides summarized and consolidated data. | Provides primitive and highly detailed data. |
| 8 | Provides summarized and multidimensional view of data. | Provides detailed and flat relational view of data. |
| 9 | Number or users is in hundreds. | Number of users is in thousands. |
| 10 | Number of records accessed is in millions. | Number of records accessed is in tens. |
| 11 | Database size is from 100 GB to 1 TB | Database size is from 100 MB to 1 GB. |
| 12 | Highly flexible. | Provides high performance. |

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

ROLAP includes the following −

* Implementation of aggregation navigation logic
* Optimization for each DBMS back-end
* Additional tools and services

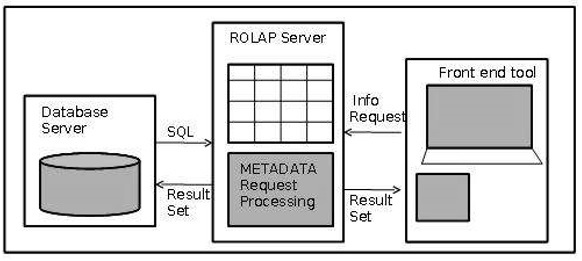
**Points to Remember**

* ROLAP servers are highly scalable.
* ROLAP tools analyze large volumes of data across multiple dimensions.
* ROLAP tools store and analyze highly volatile and changeable data.

**Relational OLAP Architecture**

ROLAP includes the following components −

* Database server
* ROLAP server
* Front-end tool.



**Advantages**

* ROLAP servers can be easily used with existing RDBMS.
* Data can be stored efficiently, since no zero facts can be stored.
* ROLAP tools do not use pre-calculated data cubes.
* DSS server of micro-strategy adopts the ROLAP approach.

**Disadvantages**

* Poor query performance.
* Some limitations of scalability depending on the technology architecture that is utilized.

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

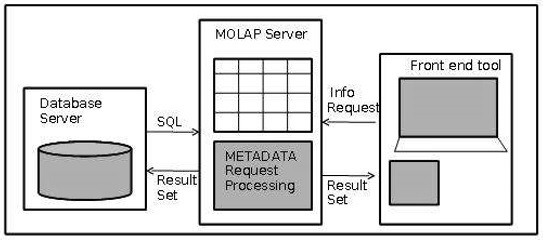
**Points to Remember −**

* MOLAP tools process information with consistent response time regardless of level of summarizing or calculations selected.
* MOLAP tools need to avoid many of the complexities of creating a relational database to store data for analysis.
* MOLAP tools need fastest possible performance.
* MOLAP server adopts two level of storage representation to handle dense and sparse data sets.
* Denser sub-cubes are identified and stored as array structure.
* Sparse sub-cubes employ compression technology.

**MOLAP Architecture**

MOLAP includes the following components −

* Database server.
* MOLAP server.
* Front-end tool.



**Advantages**

* MOLAP allows fastest indexing to the pre-computed summarized data.
* Helps the users connected to a network who need to analyze larger, less-defined data.
* Easier to use, therefore MOLAP is suitable for inexperienced users.

**Disadvantages**

* MOLAP are not capable of containing detailed data.
* The storage utilization may be low if the data set is sparse.

**MOLAP vs ROLAP**

|  |  |  |
| --- | --- | --- |
| **Sr.No.** | **MOLAP** | **ROLAP** |
| 1 | Information retrieval is fast. | Information retrieval is comparatively slow. |
| 2 | Uses sparse array to store data-sets. | Uses relational table. |
| 3 | MOLAP is best suited for inexperienced users, since it is very easy to use. | ROLAP is best suited for experienced users. |
| 4 | Maintains a separate database for data cubes. | It may not require space other than available in the Data warehouse. |
| 5 | DBMS facility is weak. | DBMS facility is strong. |

Incremental loading a.k.a Delta loading is an widely used method to load data in data warehouses from the respective source systems. This technique is employed to perform faster load in less time utilizing less system resources. In this tutorial we will understand the basic methods of incremental loading.

## What is Incremental Loading and why is it required

In almost all data warehousing scenario, we extract data from one or more source systems and keep storing them in the data warehouse for future analysis. The source systems are generally OLTP systems which store everyday transactional data. Now when it comes to loading these transactional data to data warehouse, we have 2 ways to accomplish this, Full Load or Incremental Load.

To understand these two loads better, consider a simple scenario. Let's say my source system in RDBMS - that is, a database - and I have 2 tables, customer and Sales.

In the customer table I have details of all my customers in this format:

CustomerID CustomerName Type Entry Date

1 John Individual 22-Mar-2012

2 Ryan Individual 22-Mar-2012

3 Bakers' Corporate 23-Mar-2012

In the sales table, I have the details of product sold to customers. This is how the sales table looks like:

ID CustomerID ProductDescription Qty Revenue Sales Date

1 1 White sheet (A4) 100 4.00 22-Mar-2012

2 1 James Clip (Box) 1 2.50 22-Mar-2012

3 2 Whiteboard Marker 1 2.00 22-Mar-2012

4 3 Letter Envelop 200 75.00 23-Mar-2012

5 1 Paper Clip 12 4.00 23-Mar-2012

As you can see, above tables store data for 2 consecutive days - 22 Mar and 23 Mar. On 22 Mar, I had only 2 customers (John and Ryan) who made 3 transactions in the sales table. Next day, I have got one more customer (Bakers') and I have recorded 2 transactions - one from Bakers' and 1 from my old customer John.

Also imagine, we have a data warehouse which is loaded everyday in the night with the data from this system.

### FULL LOAD METHOD FOR LOADING DATA WAREHOUSE

In case we are to opt for full load method for loading, we will read the 2 source tables (Customer and Sales) everyday in full. So,

On 22 Mar 2012: We will read 2 records from Customer and 3 records from Sales and load all of them in the target.

On 23 Mar 2012: We will read 3 records from customer (including the 2 older records) and 5 records from sales (including 3 old records) and will load or update them in the target data warehouse.

As you can clearly guess, this method of loading unnecessarily read old records that we need not read as we have already processed them before. Hence we need to implement a smarter way of loading.

### INCREMENTAL LOAD METHOD FOR LOADING DATA WAREHOUSE

In case of incremental loading, we will only read those records that are not already read and loaded into our target system (data warehouse). That is, on 22 March, we will read 2 records from customer and 3 records from sales - however - on 23 March, we will read 1 record from customer and 2 records from sales.

But how do we ensure that we "only" read those records that are not "already" read? How do we know which records are already read and which records are not?

This is a tricky question but the answer is, fortunately, easy!

We can make use of "entry date" field in the customer table and "sales date" field in the sales table to keep track of this. After each loading we will "store" the date until which the loading has been performed in some data warehouse table and next day we only extract those records that has a date greater than our stored date. Let's create a new table to store this date. We will call this table as "Batch"

Batch

Batch\_ID Loaded\_Until Status

1 22-Mar-2012 Success

2 23-Mar-2012 Success

Once we have done this, all we have to do to perform incremental or delta loading is to rite our data extraction SQL queries in this format:

Customer Table Extraction SQL

SELECT t.\*

FROM Customer t

WHERE t.entry\_date > (select nvl(

max(b.loaded\_until),

to\_date('01-01-1900', 'MM-DD-YYYY')

)

from batch b

where b.status = 'Success');

Sales Table Extraction SQL

SELECT t.\*

FROM Sales t

WHERE t.sales\_date > (select nvl(

max(b.loaded\_until),

to\_date('01-01-1900', 'MM-DD-YYYY')

)

from batch b

where b.status = 'Success');

Okay, now at this point you may wonder and ask

How does the above query work?

Let's see...

On First day (22 Mar):

There wont be any record in our batch table since we have not loaded any batch yet. So "SELECT max(b.loaded\_until)" will return NULL. That is why we have put one NVL() function to replace the NULL with a very old historical date - 01 Jan 1900 in this case.

So in the first day, we are asking the select query to extract all the data having entry date (or sales date) greater than 01-Jan-1900. This will essentially extract everything from the table. Once 22 Mar loading is complete, we will make one entry in the batch table (entry 1) to mark the successful extraction of records.

Second Day (23 Mar):

Next day, the query "SELECT max(b.loaded\_until)" will return me 22-Mar-2012. So in effect, above queries will reduce to this:

Customer Table Extraction SQL

SELECT t.\*

FROM Customer t

WHERE t.entry\_date > '22-Mar-2012';

Sales Table Extraction SQL

SELECT t.\*

FROM Sales t

WHERE t.sales\_date > '22-Mar-2012';

As you can understand, this will ensure that only 23-Mar records are extracted from the table thereby performing a successful incremental loading. After this loading is complete successfully, we will make one more entry in the batch table (entry number 2).

### Why MAX() is used in the above query?

When we try to load 23 Mar data, there was only one entry in the batch table (that of 22nd). But when we go to load 24th data or any data after that, there will be multiple entries in the batch table. We must take the max of these entries.

### Why status field is created in batch table?

This is because it might so happen that 23rd load has failed. So when we start loading again on 24th, we must take into consideration both 23rd data and 24th data.

Batch\_ID Loaded\_Until Status

1 22-Mar-2012 Success

2 23-Mar-2012 Fail

3 24-Mar-2012 Success

In the above case, 23rd batch load was a failure. That is why next day we have selected all the data after 22-Mar (including 23rd and 24th Mar).

Now that we have discussed the general concepts of Incremental loading, next please read [Incremental Loading for Dimension Table](https://dwbi.org/etl/etl/54-incremental-loading-for-dimension-table.html) and [Incremental Loading for Fact Tables](https://dwbi.org/etl/etl/55-incremental-loading-for-fact-tables.html) - where we will discuss specific approaches.

**TYPES OF DIMESIONS**

### Conformed dimension

A conformed dimension is a set of data attributes that have been physically referenced in multiple database tables using the same key value to refer to the same structure, attributes, domain values, definitions and concepts. A conformed dimension cuts across many facts.

Dimensions are conformed when they are either exactly the same (including keys) or one is a perfect subset of the other. Most important, the row headers produced in two different answer sets from the same conformed dimension(s) must be able to match perfectly.

Conformed dimensions are either identical or strict mathematical subsets of the most granular, detailed dimension. Dimension tables are not conformed if the attributes are labeled differently or contain different values. Conformed dimensions come in several different flavors. At the most basic level, conformed dimensions mean exactly the same thing with every possible fact table to which they are joined. The date dimension table connected to the sales facts is identical to the date dimension connected to the inventory facts.[[3]](https://en.wikipedia.org/wiki/Dimension_(data_warehouse)#cite_note-3)

### Junk dimension

A junk dimension is a convenient grouping of typically low-cardinality flags and indicators. By creating an abstract dimension, these flags and indicators are removed from the fact table while placing them into a useful dimensional framework.[[4]](https://en.wikipedia.org/wiki/Dimension_(data_warehouse)#cite_note-4) A Junk Dimension is a dimension table consisting of attributes that do not belong in the fact table or in any of the existing dimension tables. The nature of these attributes is usually text or various flags, e.g. non-generic comments or just simple yes/no or true/false indicators.

One solution is to create a new dimension for each of the remaining attributes, but due to their nature, it could be necessary to create a vast number of new dimensions resulting in a fact table with a very large number of foreign keys. The designer could also decide to leave the remaining attributes in the fact table but this could make the row length of the table unnecessarily large if, for example, the attributes is a long text string.

The solution to this challenge is to identify all the attributes and then put them into one or several Junk Dimensions. One Junk Dimension can hold several true/false or yes/no indicators that have no correlation with each other, so it would be convenient to convert the indicators into a more describing attribute. An example would be an indicator about whether a package had arrived: instead of indicating this as “yes” or “no”, it would be converted into “arrived” or “pending” in the junk dimension. The designer can choose to build the dimension table so it ends up holding all the indicators occurring with every other indicator so that all combinations are covered. This sets up a fixed size for the table itself which would be 2*x* rows, where *x* is the number of indicators. This solution is appropriate in situations where the designer would expect to encounter a lot of different combinations and where the possible combinations are limited to an acceptable level. In a situation where the number of indicators are large, thus creating a very big table or where the designer only expect to encounter a few of the possible combinations, it would be more appropriate to build each row in the junk dimension as new combinations are encountered. To limit the size of the tables, multiple junk dimensions might be appropriate in other situations depending on the correlation between various indicators.

Junk dimensions are also appropriate for placing attributes like non-generic comments from the fact table. Such attributes might consist of data from an optional comment field when a customer places an order and as a result will probably be blank in many cases. Therefore, the junk dimension should contain a single row representing the blanks as a surrogate key that will be used in the fact table for every row returned with a blank comment field[[5]](https://en.wikipedia.org/wiki/Dimension_(data_warehouse)" \l "cite_note-5)

### Degenerate dimension

A degenerate dimension is a key, such as a transaction number, invoice number, ticket number, or bill-of-lading number, that has no attributes and hence does not join to an actual dimension table. Degenerate dimensions are very common when the grain of a fact table represents a single transaction item or line item because the degenerate dimension represents the unique identifier of the parent. Degenerate dimensions often play an integral role in the fact table's primary key.[[6]](https://en.wikipedia.org/wiki/Dimension_(data_warehouse)#cite_note-6)

### Role-playing dimension

Dimensions are often recycled for multiple applications within the same database. For instance, a "Date" dimension can be used for "Date of Sale", as well as "Date of Delivery", or "Date of Hire". This is often referred to as a "role-playing dimension".

**Types of Facts**

There are three types of facts:

* **Additive**: Additive facts are facts that can be summed up through all of the dimensions in the fact table.
* **Semi-Additive**: Semi-additive facts are facts that can be summed up for some of the dimensions in the fact table, but not the others.
* **Non-Additive**: Non-additive facts are facts that cannot be summed up for any of the dimensions present in the fact table.

Let us use examples to illustrate each of the three types of facts. The first example assumes that we are a retailer, and we have a fact table with the following columns:

|  |
| --- |
| Date |
| Store |
| Product |
| Sales\_Amount |

The purpose of this table is to record the sales amount for each product in each store on a daily basis. **Sales\_Amount** is the fact. In this case, **Sales\_Amount** is an additive fact, because you can sum up this fact along any of the three dimensions present in the fact table -- date, store, and product. For example, the sum of **Sales\_Amount** for all 7 days in a week represents the total sales amount for that week.

Say we are a bank with the following fact table:

|  |
| --- |
| Date |
| Account |
| Current\_Balance |
| Profit\_Margin |

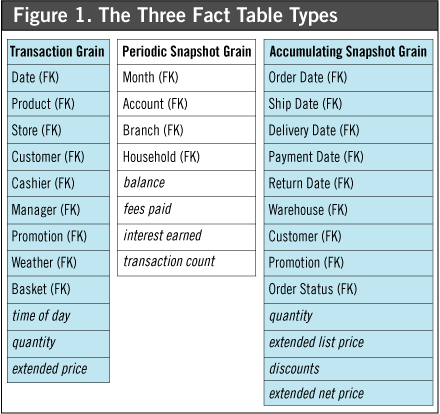
The purpose of this table is to record the current balance for each account at the end of each day, as well as the profit margin for each account for each day. **Current\_Balance** and **Profit\_Margin** are the facts. **Current\_Balance** is a semi-additive fact, as it makes sense to add them up for all accounts (what's the total current balance for all accounts in the bank?), but it does not make sense to add them up through time (adding up all current balances for a given account for each day of the month does not give us any useful information). **Profit\_Margin** is a non-additive fact, for it does not make sense to add them up for the account level or the day level.

**Types of Fact Tables**

Based on the above classifications, there are two types of fact tables:

* **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.

**Snapshot**: 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.



**SLOWLY CHANGING DIMENSIONS**

The "Slowly Changing Dimension" problem is a common one particular to data warehousing. In a nutshell, this applies to cases where the attribute for a record varies over time. We give an example below:

Christina is a customer with ABC Inc. She first lived in Chicago, Illinois. So, the original entry in the customer lookup table has the following record:

|  |  |  |
| --- | --- | --- |
| Customer Key | Name | State |
| 1001 | Christina | Illinois |

At a later date, she moved to Los Angeles, California on January, 2003. How should ABC Inc. now modify its customer table to reflect this change? This is the "Slowly Changing Dimension" problem.

There are in general three ways to solve this type of problem, and they are categorized as follows:

[Type 1](https://www.1keydata.com/datawarehousing/slowly-changing-dimensions-type-1.html): The new record replaces the original record. No trace of the old record exists.

[Type 2](https://www.1keydata.com/datawarehousing/slowly-changing-dimensions-type-2.html): A new record is added into the customer dimension table. Therefore, the customer is treated essentially as two people.

[Type 3](https://www.1keydata.com/datawarehousing/slowly-changing-dimensions-type-3.html): The original record is modified to reflect the change.

We next take a look at each of the scenarios and how the data model and the data looks like for each of them. Finally, we compare and contrast among the three alternatives.

In Type 1 Slowly Changing Dimension, the new information simply overwrites the original information. In other words, no history is kept.

In our example, recall we originally have the following table:

|  |  |  |
| --- | --- | --- |
| Customer Key | Name | State |
| 1001 | Christina | Illinois |

After Christina moved from Illinois to California, the new information replaces the new record, and we have the following table:

|  |  |  |
| --- | --- | --- |
| Customer Key | Name | State |
| 1001 | Christina | California |

Advantages:

- This is the easiest way to handle the Slowly Changing Dimension problem, since there is no need to keep track of the old information.

Disadvantages:

- All history is lost. By applying this methodology, it is not possible to trace back in history. For example, in this case, the company would not be able to know that Christina lived in Illinois before.

Usage:

About 50% of the time.

When to use Type 1:

Type 1 slowly changing dimension should be used when it is not necessary for the data warehouse to keep track of historical changes.

In Type 2 Slowly Changing Dimension, a new record is added to the table to represent the new information. Therefore, both the original and the new record will be present. The new record gets its own primary key.

In our example, recall we originally have the following table:

|  |  |  |
| --- | --- | --- |
| Customer Key | Name | State |
| 1001 | Christina | Illinois |

After Christina moved from Illinois to California, we add the new information as a new row into the table:

|  |  |  |
| --- | --- | --- |
| Customer Key | Name | State |
| 1001 | Christina | Illinois |
| 1005 | Christina | California |

Advantages:

- This allows us to accurately keep all historical information.

Disadvantages:

- This will cause the size of the table to grow fast. In cases where the number of rows for the table is very high to start with, storage and performance can become a concern.

- This necessarily complicates the ETL process.

Usage:

About 50% of the time.

When to use Type 2:

Type 2 slowly changing dimension should be used when it is necessary for the data warehouse to track historical changes.

In Type 3 Slowly Changing Dimension, there will be two columns to indicate the particular attribute of interest, one indicating the original value, and one indicating the current value. There will also be a column that indicates when the current value becomes active.

In our example, recall we originally have the following table:

|  |  |  |
| --- | --- | --- |
| Customer Key | Name | State |
| 1001 | Christina | Illinois |

To accommodate Type 3 Slowly Changing Dimension, we will now have the following columns:

* Customer Key
* Name
* Original State
* Current State
* Effective Date

After Christina moved from Illinois to California, the original information gets updated, and we have the following table (assuming the effective date of change is January 15, 2003):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Customer Key | Name | Original State | Current State | Effective Date |
| 1001 | Christina | Illinois | California | 15-JAN-2003 |

Advantages:

- This does not increase the size of the table, since new information is updated.

- This allows us to keep some part of history.

Disadvantages:

- Type 3 will not be able to keep all history where an attribute is changed more than once. For example, if Christina later moves to Texas on December 15, 2003, the California information will be lost.

Usage:

Type 3 is rarely used in actual practice.

When to use Type 3:

Type III slowly changing dimension should only be used when it is necessary for the data warehouse to track historical changes, and when such changes will only occur for a finite number of time.

**What is the difference between OLAP and data warehouse?**

The following are the differences between OLAP and data warehousing:  
  
**Data Warehouse**  
  
Data from different data sources is stored in a relational database for end use analysis.  
Data organization is in the form of summarized, aggregated, non volatile and subject oriented patterns.  
Supports the analysis of data but does not support data of online analysis.  
  
**Online Analytical Processing**  
  
With the usage of analytical queries, data is analyzed and evaluated in the data ware house.  
Data aggregation and summarization is utilized to organize data using multidimensional models.  
Speed and flexibility for online data analysis is supported for data analyst in real time environment.

**What is the difference between OLAP and data warehouse?**

A data warehouse serves as a repository to store historical data that can be used for analysis. OLAP is Online Analytical processing that can be used to analyze and evaluate data in a warehouse. The warehouse has data coming from varied sources. OLAP tool helps to organize data in the warehouse using multidimensional models.

