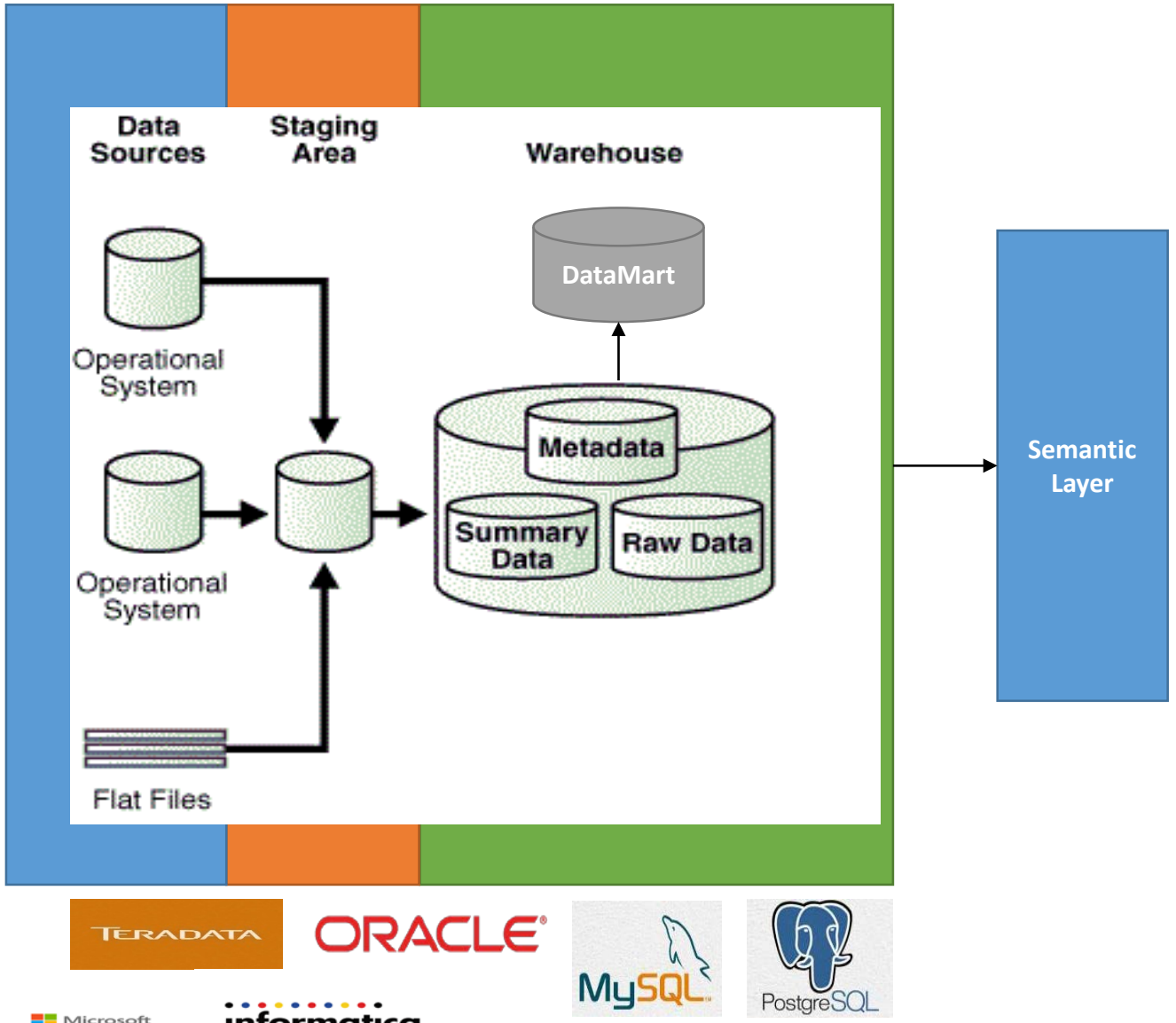
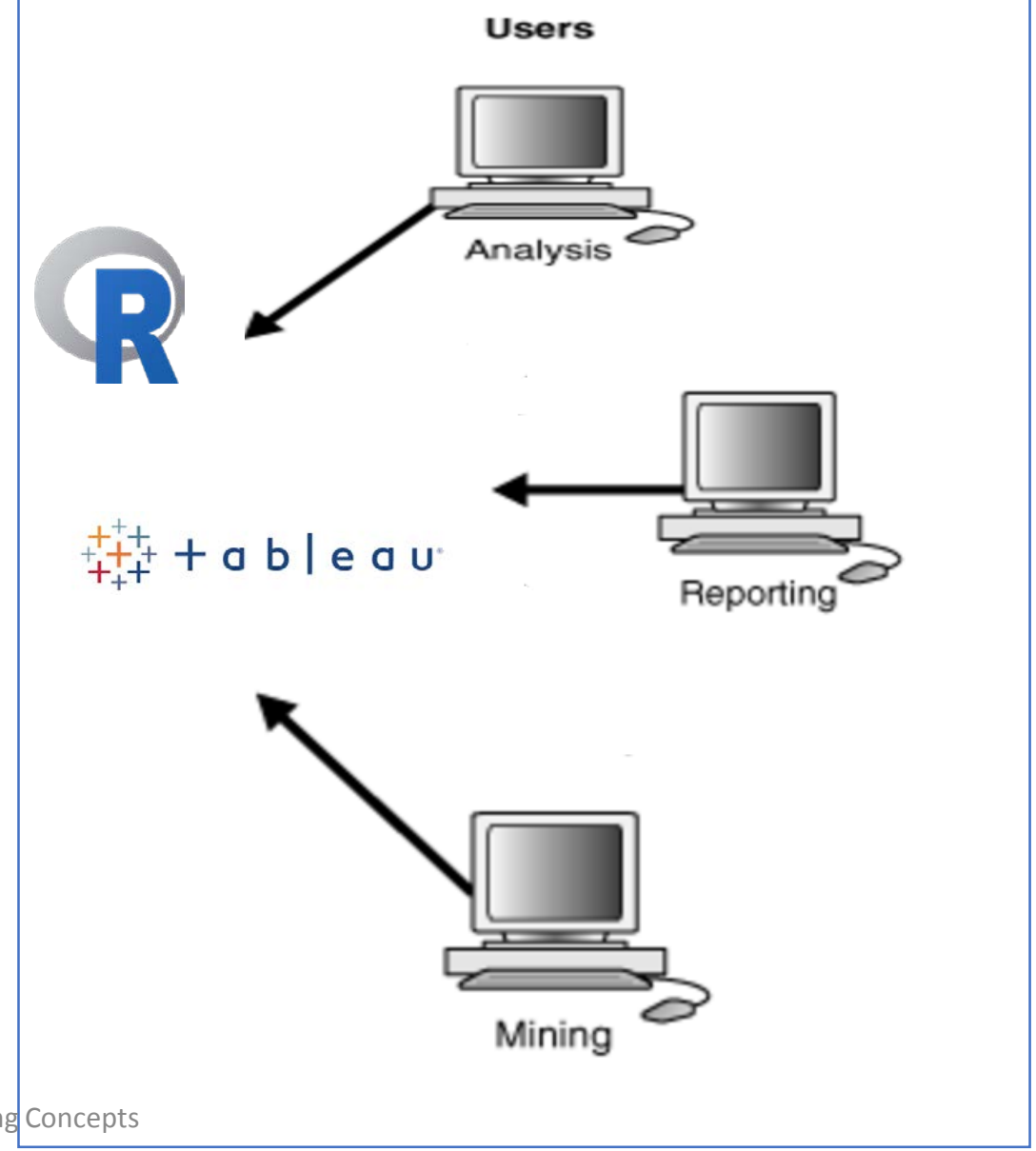
Stay Ahead

Data Warehousing concept

Big Bazaar Case Study

Big Bazaar Case Study





Data Warehousing concepts

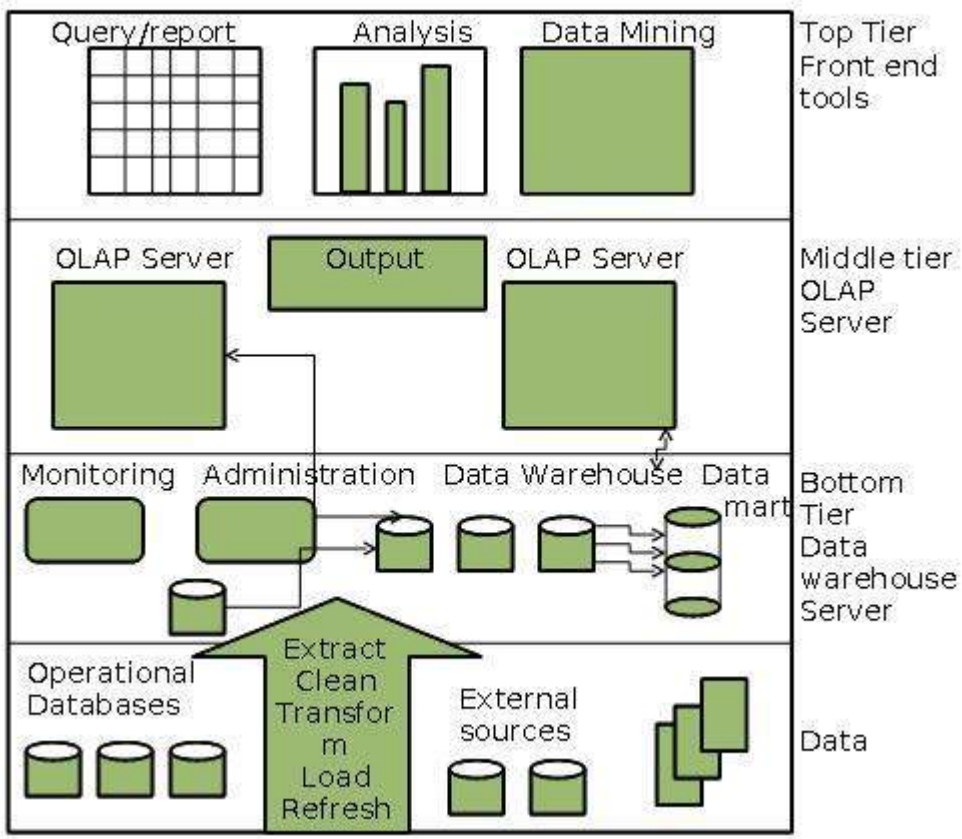
Data warehouse

Subject Oriented - A data warehouse is subject oriented because it provides information around a subject rather than the organization's ongoing operations.
These subjects can be product, customers, suppliers, sales, revenue, etc.
A data warehouse does not focus on the ongoing operations, rather it focuses on modelling and analysis of data for decision making.

Integrated - A data warehouse is constructed by integrating data from heterogeneous sources such as relational databases, flat files, etc.
This integration enhances the effective analysis of data.

Time Variant - The data collected in a data warehouse is identified with a particular time period.
The data in a data warehouse provides information from the historical point of view.

Non-volatile - Non-volatile means the previous data is not erased when new data is added to it.
A data warehouse is kept separate from the operational database and therefore frequent changes in operational database is not reflected in the data warehouse.



Function of data warehouse tools and utilities

Data Extraction - Involves gathering data from multiple heterogeneous sources.
Data Cleaning - Involves finding and correcting the errors in data.
Data Transformation - Involves converting the data from legacy format to warehouse format.
Data Loading - Involves sorting, summarizing, consolidating, checking integrity, and building indices and partitions.
Refreshing - Involves updating from data sources to warehouse.

Slowly Changing Dimension

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:

Rahul is a customer with ABC Inc. He first lived in Mumbai. So, the original entry in the customer lookup table has the following record:

Customer Key	Name	State
1001	Rahul	Mumbai

At a later date, he moved to Gurgaon 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: The new record replaces the original record. No trace of the old record exists.

Type 2: A new record is added into the customer dimension table. Therefore, the customer is treated essentially as two people.

Type 3: 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.

Type 1

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	Rahul	Mumbai

After Rahul moved from Mumbai, the new information replaces the new record, and we have the following table:

Customer Key	Name	State
1001	Rahul	Gurgaon

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.

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

Type 2

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	Rahul	Mumbai

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

Customer Key	Name	State	Start Date	End Date
1001	Rahul	Mumbai	1/1/2013	1/1/2014
1005	Rahul	Gurgaon	1/1/2014	31/12/2999

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.

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

Type 3

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	Rahul	Mumbai

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	Rahul	Mumbai	Gurgaon	15/1/2003

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:

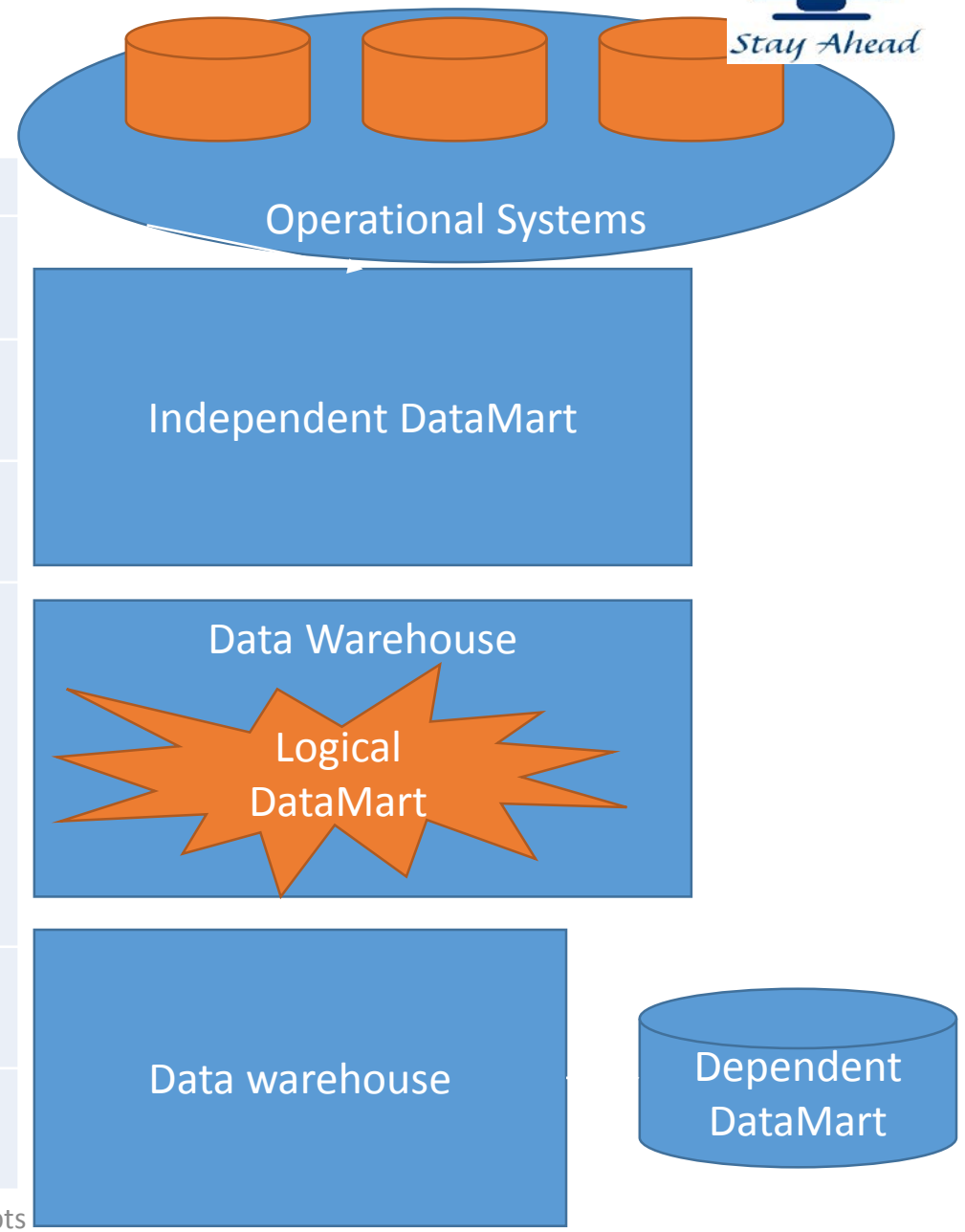
- Type 3 will not be able to keep all history where an attribute is changed more than once. For example, if Christina later moves to Texas on December 15, 2003, the California information will be lost.

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.

Data Warehousing concepts

Data Marts

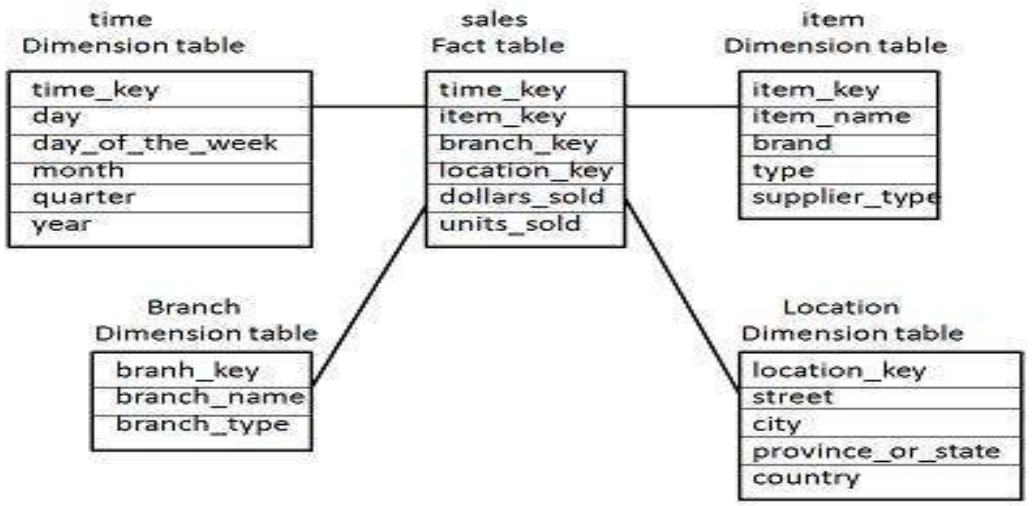
Data marts	Pro	Con
	Easy to implement	Not an enterprise wide solution, costly as more DMs are added
		Historical data limits
Independent	Payback value can be almost immediate	Transformation needed
	No historical data limits	
	Allows drill downs, trend analysis	
Logical	No transformation needed	Less physical control over the data
	All advantages of Logical DM	Additional movement of data may be necessary
Dependent	Allows phyical control over the data	Some transformation may needed



Data Warehousing concepts

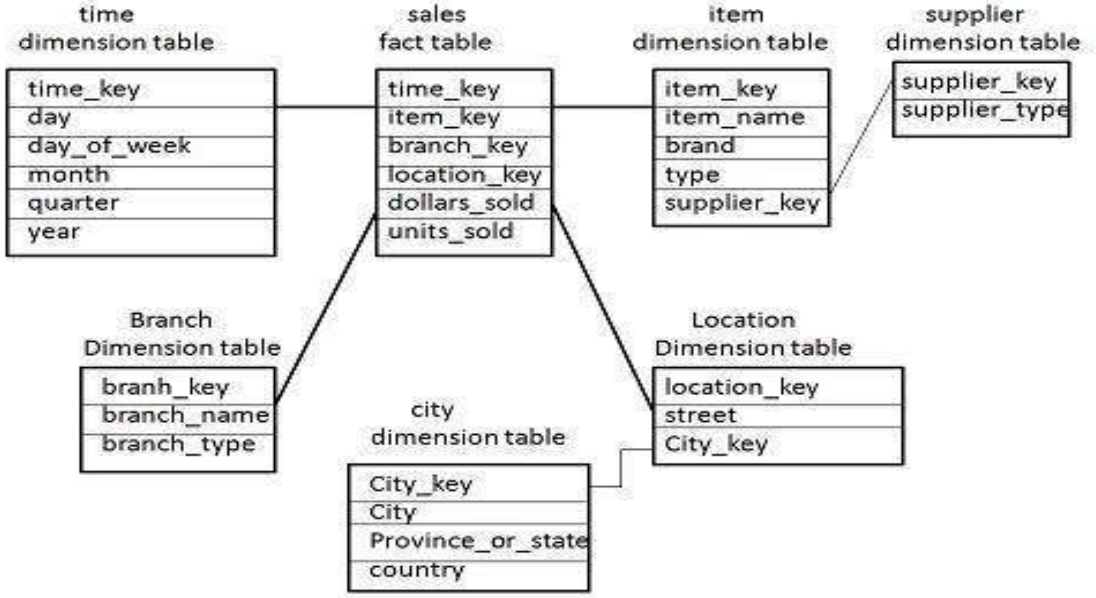
Star Schema

Each dimension in a star schema is represented with only one-dimension table. This dimension table contains the set of attributes. The following diagram 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.



Snowflake Schema

Some dimension tables in the Snowflake schema are normalized. The normalization splits up the data into additional tables. Unlike Star schema, the dimensions table in a snowflake schema are normalized. For example, the item dimension table in 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 normalization in the Snowflake schema, the redundancy is reduced and therefore, it becomes easy to maintain and the save storage space.

Data Warehousing concepts

Types of Facts and Dimensions

Facts:

Additive: Additive facts are facts that can be summed up through all of the dimensions in the fact table. **(Sales Amount)**

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 **(Current Balance)**

Non-Additive: Non-additive facts are facts that cannot be summed up for any of the dimensions present in the fact table **(Profit margin)**

Fact less fact: A fact less fact table is a fact table that does not have any measures.

Date
Store
Product
Sales_Amount

Date
Account
Current_Balance
Profit_Margin

Dimensions:

Junk Dimension: A fact less fact table is a fact table that does not have any measures.

DIM_JUNK

JUNK_ID	TXN_CODE	COUPON_IND	PREPAY_IND
1	1	Y	Y
2	2	Y	Y
3	3	Y	Y
4	1	Y	N
5	2	Y	N
6	3	Y	N
7	1	N	Y
8	2	N	Y
9	3	N	Y
10	1	N	N
11	2	N	N
12	3	N	N

Conformed Dimension: A conformed dimension is a dimension that has exactly the same meaning and content when being referred from different fact tables. A conformed dimension can refer to multiple tables in multiple data marts within the same organization. For two dimension tables to be considered as conformed, they must either be identical or one must be a subset of another. There cannot be any other type of difference between the two tables. For example, two dimension tables that are exactly the same except for the primary key are not considered conformed dimensions.

Time Dimension, Location Dimension

Normalization Concept – Why required?

Updation Anamoly : If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.

Insertion Anamoly : We tried to insert data in a record that does not exist at all.

Deletion Anamoly : We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.

1NF

First Normal Form is defined in the definition of relations (tables) itself. This rule defines that all the attributes in a relation must have atomic domains. The values in an atomic domain are indivisible units.

Course	Content
Programming	Java, c++
Web	HTML, PHP, ASP

Course	Content
Programming	Java
Programming	c++
Web	HTML
Web	PHP
Web	ASP

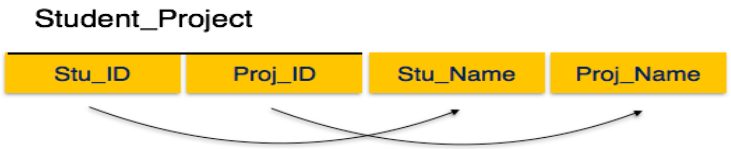
2NF

Before we learn about the second normal form, we need to understand the following –

•**Prime attribute** – An attribute, which is a part of the prime-key, is known as a prime attribute.

•**Non-prime attribute** – An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.

If we follow second normal form, then every non-prime attribute should be fully functionally dependent on prime key attribute. That is, if X → A holds, then there should not be any proper subset Y of X, for which Y → A also holds true.



We see here in Student_Project relation that the prime key attributes are Stu_ID and Proj_ID. According to the rule, non-key attributes, i.e. Stu_Name and Proj_Name must be dependent upon both and not on any of the prime key attribute individually. But we find that Stu_Name can be identified by Stu_ID and Proj_Name can be identified by Proj_ID independently. This is called **partial dependency**, which is not allowed in Second Normal Form.

Data Warehousing concepts

3NF

For a relation to be in Third Normal Form, it must be in Second Normal form and the following must satisfy −

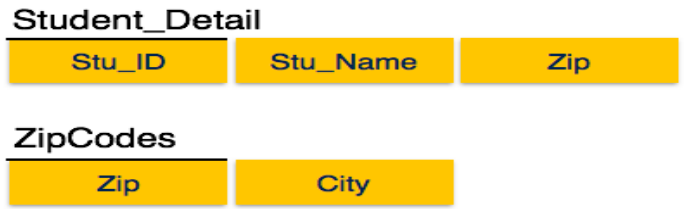
•No non-prime attribute is transitively dependent on prime key attribute.

•For any non-trivial functional dependency, X → A, then either −

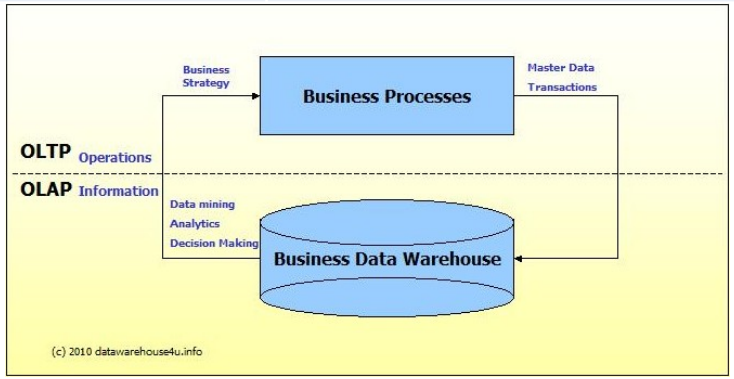
- X is a superkey or,
- A is prime attribute.

We find that in the above Student_detail relation, Stu_ID is the key and only prime key attribute. We find that City can be identified by Stu_ID as well as Zip itself. Neither Zip is a superkey nor is City a prime attribute. Additionally, Stu_ID → Zip → City, so there exists **transitive dependency**.

To bring this relation into third normal form, we break the relation into two relations as follows



	Online Transaction Processing (OLTP)	Online Analytical Processing (OLAP)
Source of data	Operational data; OLTPs are the original source of the data.	Consolidation data; OLAP data comes from the various OLTP Databases
Purpose of data	To control and run fundamental business tasks	To help with planning, problem solving, and decisionsupport
What the data	Reveals a snapshot of ongoing business processes	Multi-dimensional views of various kinds of business activities
Queries	Relatively standardized and simple queries Returning relatively few records	Often complex queries involving aggregations
Processing Speed	Typically very fast	Depends on the amount of data involved; batch data refreshes and complex queries may take many hours; query speed can be improved by creating indexes
Space Requirements	Can be relatively small if historical data is archived	Larger due to the existence of aggregation structures and history data; requires more indexes than OLTP
Database Design	Highly normalized with many tables	Typically de-normalized with fewer tables; use of star and/or snowflake schemas



Primary and Foreign Key

In Oracle, a **primary key** is a single field or combination of fields that uniquely defines a record. None of the fields that are part of the primary key can contain a null value. A table can have only one primary key.

Note
In Oracle, a primary key can not contain more than 32 columns.
A primary key can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.

Create Primary Key using CREATE TABLE command
You can create a primary key in Oracle with the CREATE TABLE statement.

```
CREATE TABLE table_name
(
  column1 datatype null/not null,
  column2 datatype null/not null,
  ...

  CONSTRAINT constraint_name PRIMARY KEY (column1, column2, ... column_n)
);
```

```
CREATE TABLE supplier
(
  supplier_id numeric(10) not null,
  supplier_name varchar2(50) not null,
  contact_name varchar2(50),
  CONSTRAINT supplier_pk PRIMARY KEY (supplier_id)
);
```

```
CREATE TABLE supplier
(
  supplier_id numeric(10) not null,
  supplier_name varchar2(50) not null,
  contact_name varchar2(50),
  CONSTRAINT supplier_pk PRIMARY KEY (supplier_id, supplier_name)
);
```

Create Primary Key using ALTER TABLE command
You can create a primary key in Oracle with the ALTER TABLE statement.

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name PRIMARY KEY (column1, column2, ... column_n);
```

```
ALTER TABLE supplier
ADD CONSTRAINT supplier_pk PRIMARY KEY (supplier_id);
```

Primary and Foreign Key

A foreign key is a way to enforce referential integrity within your Oracle database. A foreign key means that values in one table must also appear in another table.
The referenced table is called the *parent table* while the table with the foreign key is called the *child table*. The foreign key in the child table will generally reference a primary key in the parent table.
A foreign key can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.

Create Primary Key using CREATE TABLE command

```
CREATE TABLE table_name
(
 column1 datatype null/not null,
 column2 datatype null/not null,
 CONSTRAINT fk_column
   FOREIGN KEY (column1, column2, ... column_n)
   REFERENCES parent_table (column1, column2, ... column_n)
);
```

```
CREATE TABLE supplier
( supplier_id numeric(10) not null,
 supplier_name varchar2(50) not null,
 contact_name varchar2(50),
 CONSTRAINT supplier_pk PRIMARY KEY (supplier_id)
);
```

```
CREATE TABLE products
( product_id numeric(10) not null,
 supplier_id numeric(10) not null,
 CONSTRAINT fk_supplier
   FOREIGN KEY (supplier_id)
   REFERENCES supplier(supplier_id)
);
```

```
CREATE TABLE supplier
( supplier_id numeric(10) not null,
 supplier_name varchar2(50) not null,
 contact_name varchar2(50),
 CONSTRAINT supplier_pk PRIMARY KEY (supplier_id, supplier_name)
);
```

```
CREATE TABLE products
( product_id numeric(10) not null,
 supplier_id numeric(10) not null,
 supplier_name varchar2(50) not null,
 CONSTRAINT fk_supplier_comp
   FOREIGN KEY (supplier_id, supplier_name)
   REFERENCES supplier(supplier_id, supplier_name)
);
```

Create Foreign Key using ALTER TABLE command

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
  FOREIGN KEY (column1, column2, ... column_n)
  REFERENCES parent_table (column1, column2, ... column_n);
```

```
ALTER TABLE products
ADD CONSTRAINT fk_supplier
 FOREIGN KEY (supplier_id)
 REFERENCES supplier(supplier_id);
```

Data Warehousing concepts

Primary and Foreign Key

Primary Key Rules
A primary key is required
A primary key value must be unique
The primary key value cannot be NULL
The primary key value should not be changed
The primary key column should not be changed
A primary key may be any number of columns. In oracle the restriction is 32 columns

Foreign Key Rules
Foreign keys are optional
A foreign key value may be non-unique
The foreign key value may be NULL
The foreign key value may be changed
A foreign key may be any number of columns.In oracle the restriction is 32 columns
Each foreign key must exist as a primary key in a related table