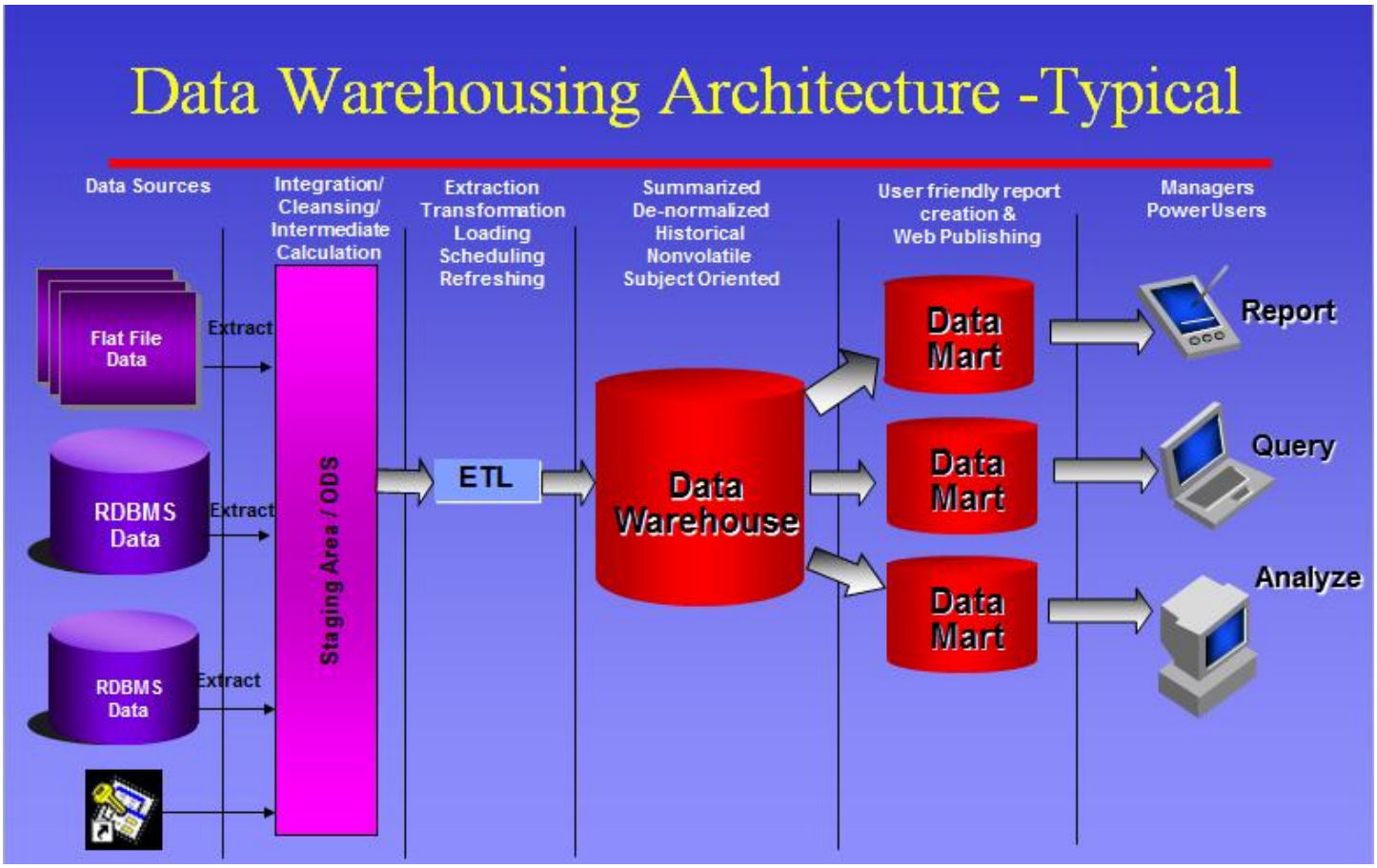
Data Warehouse Definition:

A data warehouse is a Subject-Oriented, Integrated, Time-Variant and Non-Volatile collection of data in support of management's decision making process.

Subject-Oriented: A data warehouse can be used to analyze a particular subject area. For example, "sales" can be a particular subject.
Integrated: A data warehouse integrates data from multiple data sources. For example, source A and source B may have different ways of identifying a product, but in a data warehouse, there will be only a single way of identifying a product.
Time-Variant: Historical data is kept in a data warehouse. For example, one can retrieve data from 3 months, 6 months, 12 months, or even older data from a data warehouse. This contrasts with a transactions system, where often only the most recent data is kept. For example, a transaction system may hold the most recent address of a customer, where a data warehouse can hold all addresses associated with a customer.
Non-volatile: Once data is in the data warehouse, it will not change. So, historical data in a data warehouse should never be altered.

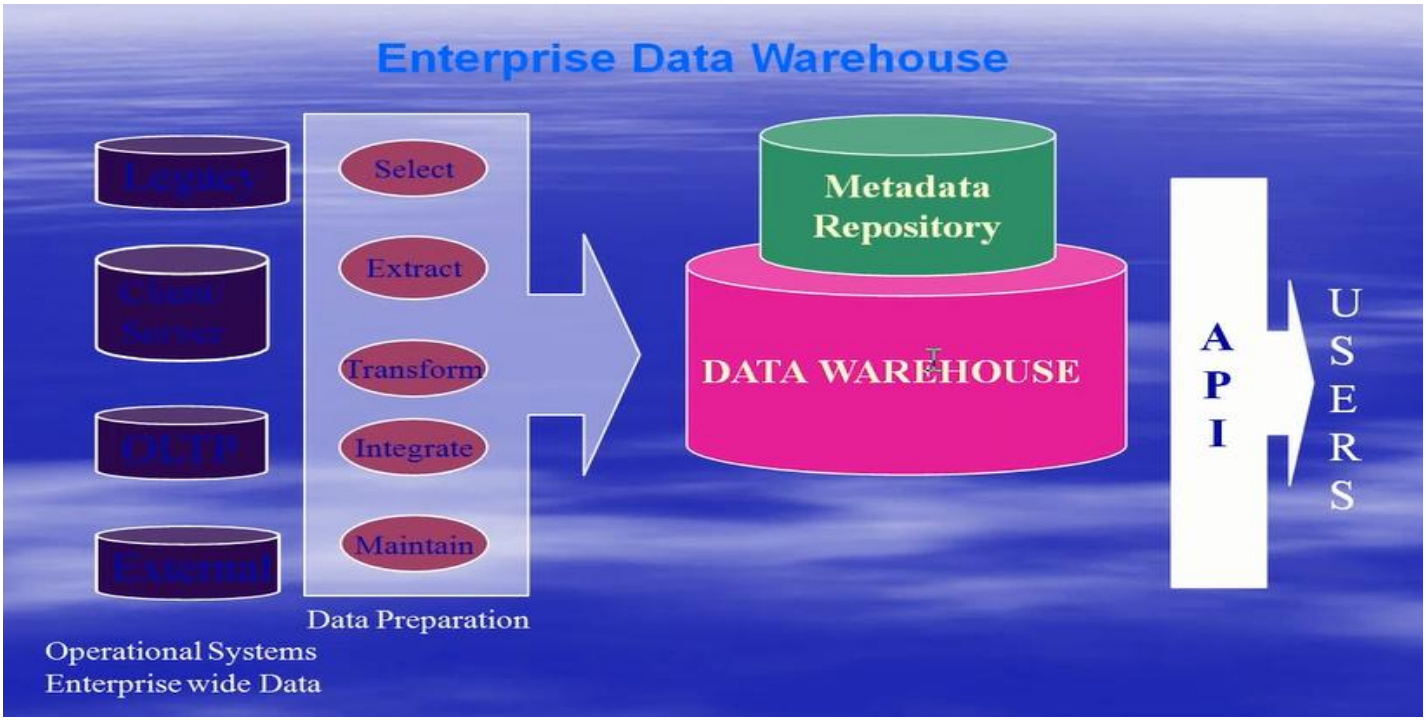
ETL Life Cycle:



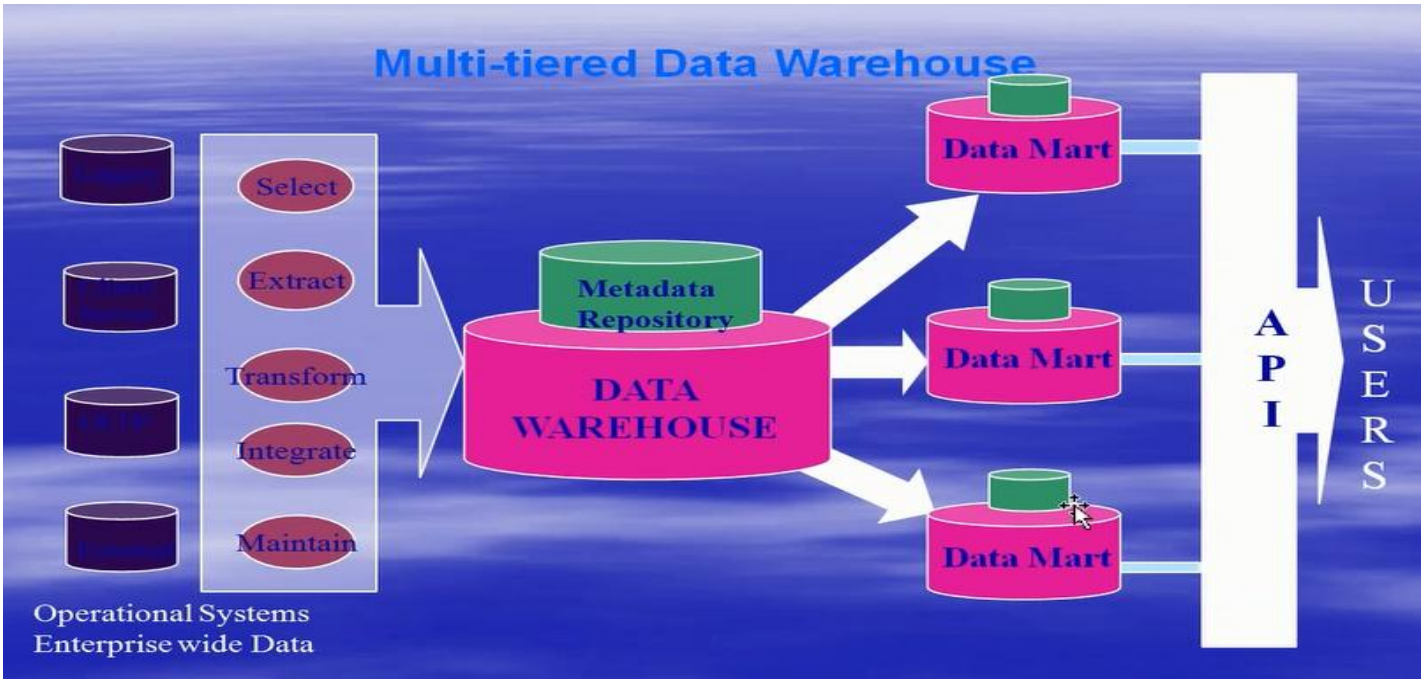
Data Warehouse Architectures:

1) Enterprise Data Warehouse (EDW)
2) Top Down Approach (Bill Inmon)
3) Bottom Up Approach (Ralph Kimball)

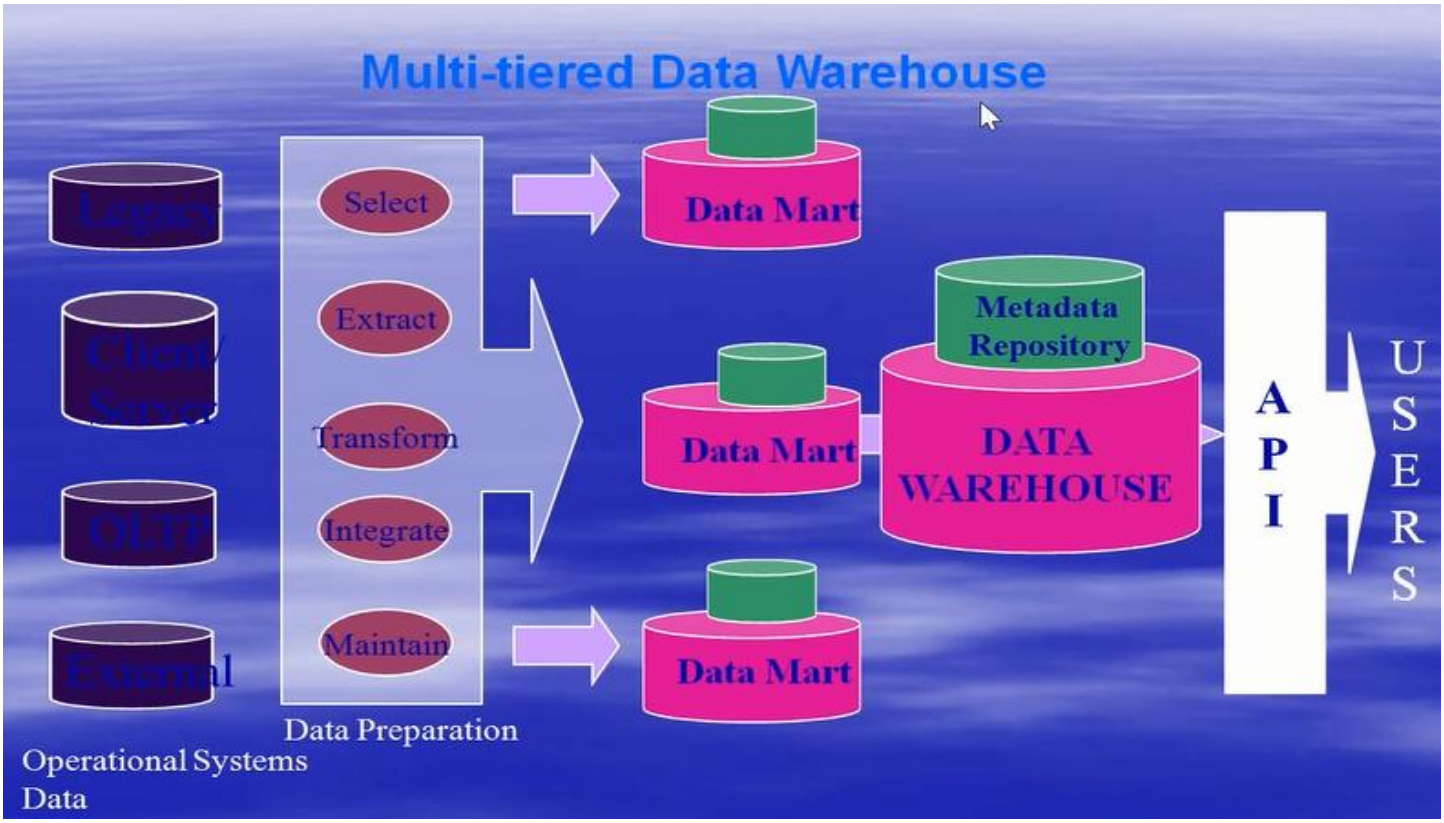
1) Enterprise Data Warehouse (EDW):



2) Top down Approach (Bill Inmon):



3) Bottom up Approach (Ralph Kimball):



OLTP VS OLAP:

Features	OLTP	OLAP
Characteristics	Operational Processing	Informational Processing
Purpose	Transaction	Analysis
User	Clerks, End User, DBA etc	Knowledge Workers
Function	Day to Day Operation	Decision Support
Data	Current	Historical
Permissions	Read/Write	Only Read
Execution	Simple Queries	Complex Queries
DB Design	E-R Model	Dimensional Modeling
Nos of Records	Tens & Thousands	Millions
Nos of Users	More	Less
DB Size	100MB to GB	100GB to TB
Metric	Transaction Throughput	Query Throughput
Focus	Day to Day Transactions	Future Predictions & Support
Normalization	Normalized Data	De-normalized Data
Access	Frequency is high	Frequency is Medium to Low

Why the need of Data Warehouse?

1)Reporting Pressures: Relieve reporting pressure on transactional databases.

2)Restructure Data: Restructure data to speed up data analysis and reportingcapabilities.

3)Reduce Complexity: Reduce the user complexity associated with generating new reports

4) Clean Data: Create a repository of "clean data" that does not require wholesale changes to the transactional systems or business processes.

5) Multiple Source Analysis: Allow easier reporting across multiple transactional systems and external data sources.

6) Historic Analysis: To provide a data source supporting a longer span of time than can be reasonable supported on the transactional systems.

Types of OLAP Tools:

1) ROLAP: Relations OALP**e.g. OBIEE, BO**
2) MOLAP: Multi-Dimensional OLAP**e.g. COGNOS, SSAS, Hyperion ESSBASE, Tableau**
3) HOLAP: Hybrid OLAP**e.g. Micro strategy**
4) DOAP: Desktop OALP**e.g. Tableau Desktop, Qlikview Desktop**

1) ROLAP:

This methodology relies on manipulating the data stored in the relational database to give the appearance of traditional OLAP's slicing and dicing functionality. In essence, each action of slicing and dicing is equivalent to adding a "WHERE" clause in the SQL statement.

Advantages:

a) Can handle large amounts of data: The data size limitation of ROLAP technology is the limitation on data size of the underlying relational database. In other words, ROLAP itself places no limitation on data amount.

b) Can leverage functionalities inherent in the relational database: Often, relational database already comes with a host of functionalities. ROLAP technologies, since they sit on top of the relational database, can therefore leverage these functionalities.

Disadvantages:

a) Performance can be slow: Because each ROLAP report is essentially a SQL query (or multiple SQL queries) in the relational database, the query time can be long if the underlying data size is large.

b)Limited by SQL functionalities: Because ROLAP technology mainly relies on generating SQL statements to query the relational database, and SQL statements do not fit all needs (for example, it is difficult to perform complex calculations using SQL), ROLAP technologies are therefore traditionally limited by what SQL can do. ROLAP vendors have mitigated this risk by building into the tool out-of-the-box complex functions as well as the ability to allow users to define their own functions.

2) MOLAP:

This is the more traditional way of OLAP analysis. In MOLAP, data is stored in a multidimensional cube. The storage is not in the relational database, but in proprietary formats.

Advantages:

a) Excellent performance: MOLAP cubes are built for fast data retrieval, and are optimal for slicing and dicing operations.

b) Can perform complex calculations: All calculations have been pre-generated when the cube is created. Hence, complex calculations are not only doable, but they return quickly.

Disadvantages:

a) Limited in the amount of data it can handle: Because all calculations are performed when the cube is built, it is not possible to include a large amount of data in the cube itself. This is not to say that the data in the cube cannot be derived from a large amount of data. Indeed, this is possible. But in this case, only summary-level information will be included in the cube itself.

b) Requires additional investment: Cube technology are often proprietary and do not already exist in the organization. Therefore, to adopt MOLAP technology, chances are additional investments in human and capital resources are needed.

What is Data Modeling?

Data Modeling is representing the real world set of Data Structures or Entities and their relationship in their Data Models, required for a database.

Data Modeling consists of various types like:

a) Conceptual Data Modeling

b) Logical Data Modeling

c) Physical Data Modeling

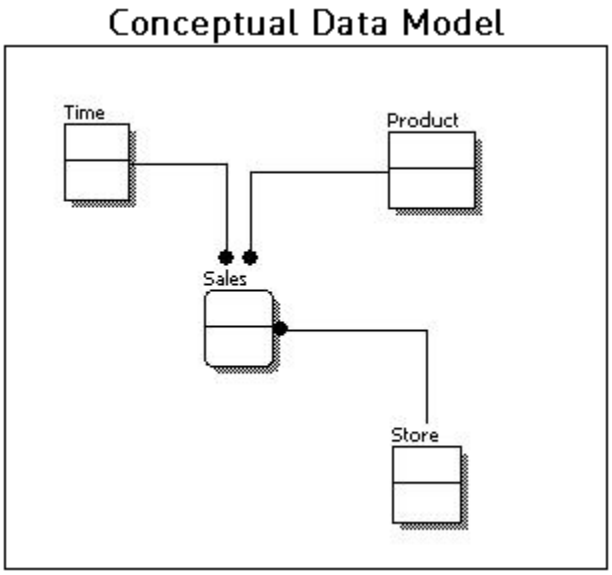
Different levels of Data models:

1) Conceptual Data Model:

At this level, the Data Modeler attempts to identify the important Entities and the Relationship among them.

a) No Attribute is specified.

b) No Primary Key is specified.



From the figure above, we can see that the only information shown via the Conceptual Data Model is the entities that describe the data and the Relationships between those Entities. No other information is shown through the Conceptual Data Model.

2) Logical Data Model:

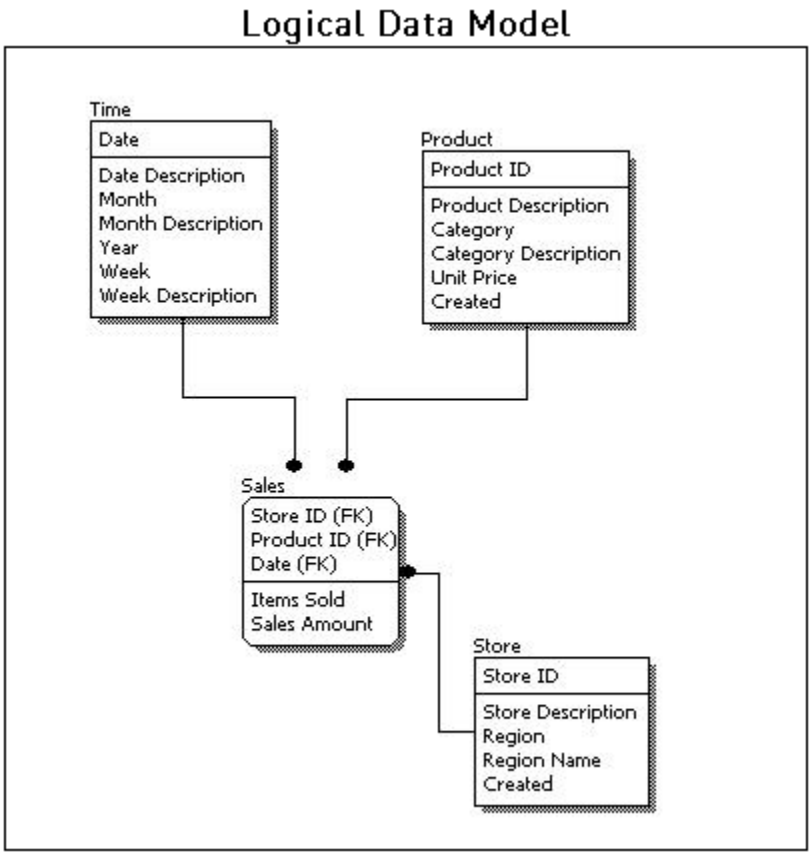
At this level, the Data Modeler attempts to describe the data in detail as possible, without knowing how they will be physically implemented in the database.

a) All attributes for each entity are specified.

b) Primary Keys for each entity are specified.

c) Foreign keys for identifying the relationship between different entities are specified.

d) Normalization occurs at this level



Comparing the Logical Data Model shown above with the Conceptual Data Model diagram, we see the main differences between the two:

- In a Logical Data Model, Primary Keys are present, whereas in a Conceptual Data Model, no Primary Key is present.
- In a Logical Data Model, all attributes are specified within an entity. No attributes are specified in a Conceptual Data Model.
- Relationships between Entities are specified using Primary Keys and Foreign Keys in a Logical Data Model. In a Conceptual Data Model, the Relationships are simply stated, not specified, so we simply know that Two Entities are related, but we do not specify what Attributes are used for this Relationship.

3) Physical Data Model:

At this level, the Data Modeler will specify how the Logical Data Model will be realized in the database schema.Physical Data Model represents how the model will be built in the database. A physical database model shows all Table Structures, including Column Name, Column Data Type, Column Constraints, Primary Key, Foreign Key, and Relationships between Tables.

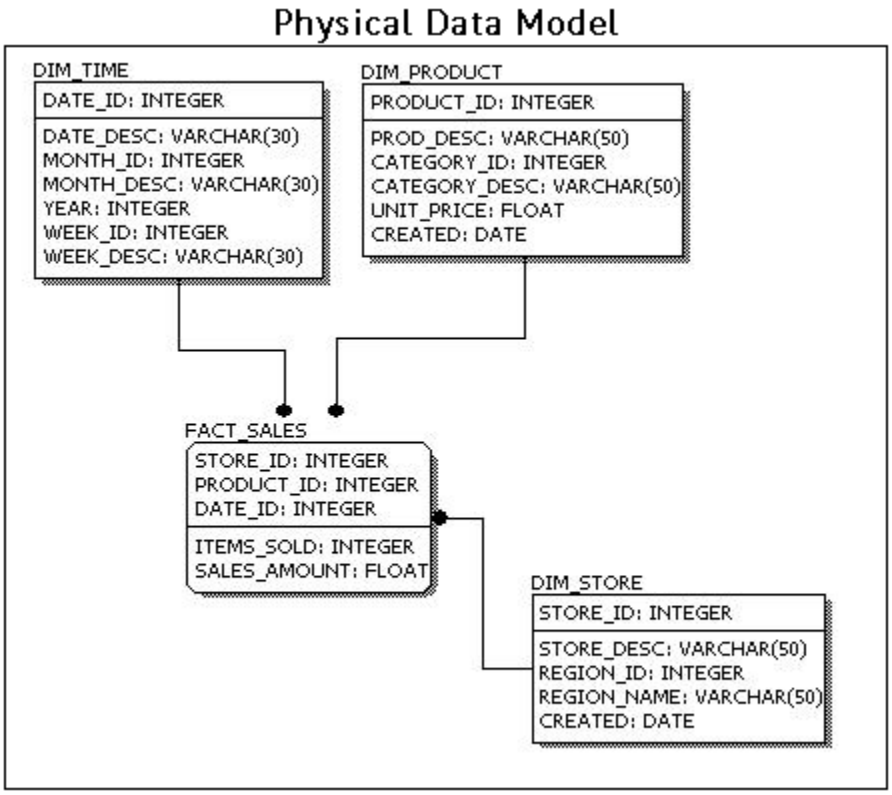
The steps for Physical Data Model design are as follows:

a) Convert entities into tables.

b) Convert attributes into columns.

c) Convert relationships into PK's & FK's

d) Modify the Physical Data Model based on Physical Constraints/Requirements.



Comparing the Logical Data Model shown above with the Logical Data Model Diagram, we see the main differences between the two:

- Entity Names are now Table Names.
- Attributes are now Column Names.
- Data Type for each column is specified. Data Types can be different depending on the actual database being used.

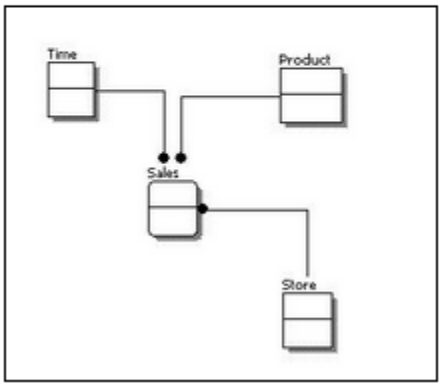
Data Modeling - Conceptual, Logical, and Physical Data Models

The three levels of data modeling, Conceptual Data Model, Logical Data Model, and Physical Data Model, were discussed in prior sections. Here we compare these three types of data models. The table below compares the different features:

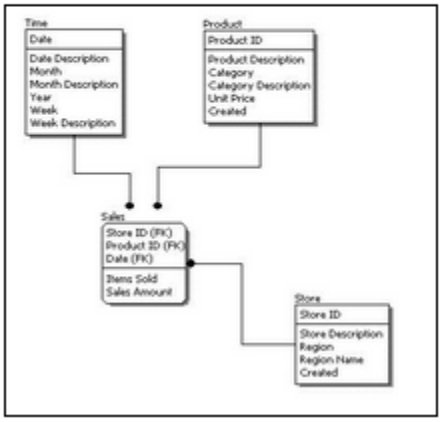
Feature	Conceptual	Logical	Physical
Entity Names	✓	✓	
Entity Relationships	✓	✓	
Attributes		✓	
Primary Keys		✓	✓
Foreign Keys		✓	✓
Table Names			✓
Column Names			✓
Column Data Types			✓

Below we show the Conceptual, Logical, and Physical versions of a Single Data Model.

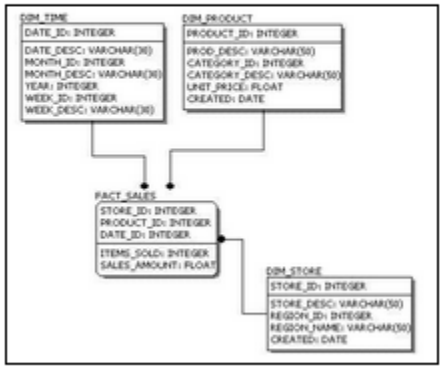
Conceptual Model Design



Logical Model Design



Physical Model Design



We can see that the complexity increases from Conceptual to Logical to Physical. This is why we always first start with the Conceptual Data Model (so we understand at high level what are the different entities in our data and how they relate to one another), then move on to the Logical Data Model (so we understand the details of our data without worrying about how they will actually implemented), and finally the Physical Data Model (so we know exactly how to implement our data model in the database of choice). In a data warehousing project, sometimes the conceptual data model and the Logical Data Model are considered as a single deliverable.

Dimension Tables: Dimension tables are used to describe dimensions; i.e. they contain primary keys, and the detailed values and attributes related to Dimensions. Without having the dimensions using fact table is meaningless.
E.g.Sales is fact table, product, year are dimensions.
How can you find the sales for a particular year/month without using the dimension (year/product)?

Types of dimension:
1) Slowly Changing Dimension
2) Fast Changing Dimensions
3) Role Playing Dimension
4) Conformed Dimension
5) Garbage/Junk Dimension
6) Degenerate Dimension

1) Slowly Changing Dimensions:
A Slowly Changing Dimension is a Dimension whose attribute or attributes for a record (row) change slowly over time.
Ex: Customers change their Names, Address etc.
Types: SCD1, SCD2 & SCD3.

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: 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 partial 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 Slowly Changing Dimension:

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.

Type 2 Slowly Changing Dimension:

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.

Type 3 Slowly Changing Dimension:

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.

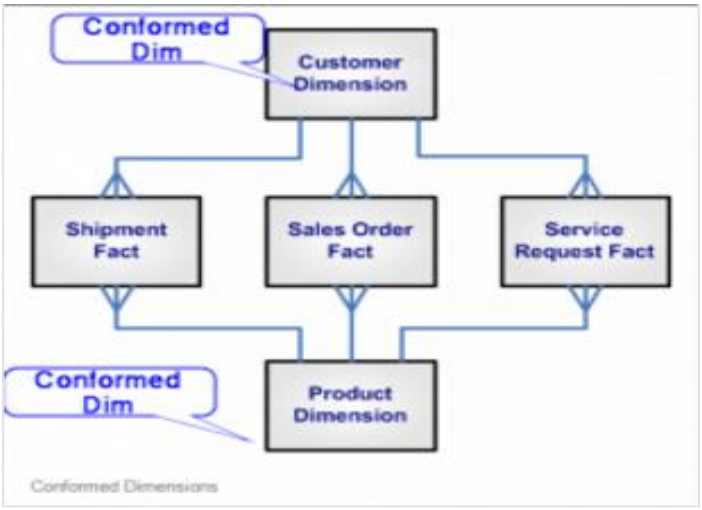
2) Fast Changing Dimension:

A Fast Changing Dimension is a Dimension whose attribute or attributes for a record (row) change rapidly over time.

Ex: Age of associates, daily balance etc.

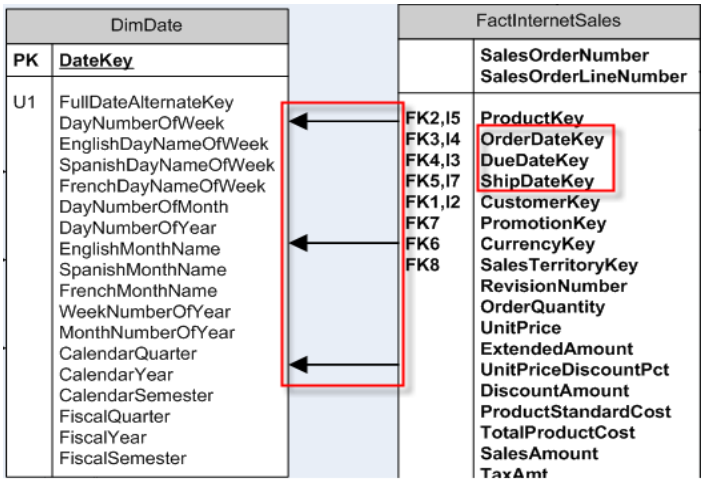
3) Conformed Dimension:

A Dimension that is used in multiple locations is called conformed dimensions. A conformed dimension may be used with multiple fact tables in single database, or across multiple data marts or Data warehouses.



4) Role Playing Dimension:

Role Playing Dimensions are the Dimensions which often used for multiple purposes within same database.Here same dimension key is associated with more than one foreign key in the fact table in the database for the different purposes.



i.e.: In Date dimensions, [**FullDateAlternateKey]** is associated with [**OrderdateKey], [DuedateKey], and [ShipdateKey]** in the fact table to solve different purpose in Data warehouse.

Hence, "Date" Dimension can be used for "Order Date", "Due Date", or "Ship Date". This is often referred to as a "Role Playing Dimension".

5)Garbage Dimension/Junk dimension:

A Garbage/Junk Dimension is a Dimension that consists of low cardinality columns such as Codes, Indicators, and Status Flags.

6) Degenerate Dimension

Degenerate Dimensions are the key which is stored in Fact Table and has no related dimension. It is neither a Fact nor an Attribute.

OR

Degenerate dimension is a dimension which is derived from the fact table and doesn't have its own dimension table.

FK_OrderDate	FK_Product	DD_OrderID	Quantity	Amount
20130508	455	152	6	52.45
20130508	213	152	1	12.44
20130508	211	152	2	83.82
20130508	321	128	8	9.54
20130508	751	128	4	16.22

E.g.We have a fact table with FK_OrderDate, FK_Product, DD_OrderID, Quantity, Amount. In this fact table, **DD_OrderID** 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, DD_OrderID is a degenerate dimension or line item dimension.

Note:

1) Do not place Attributes in Facts
2) Key without Related Dimensions
3) Provides Grouping & Business meaning

Fact Tables: Fact tables contain Foreign Keys referring to Dimension tables where descriptive information is kept as well as measurable facts that Data Analysts would want to examine. Fact Tables are designed to a low level of uniform detail (referred to as "Granularity" or "Grain"), i.e. Facts can record events at a very atomic level. This can result in the accumulation of a large number of records in a fact table over a period of time.

Type 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.
  Ex: Units and Total Price.
- **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.
  Ex: Account Balances,Quantity on hand.
- **Non-Additive:** Non-additive facts are facts that cannot be summed up for any of the dimensions present in the fact table.
  Ex: Textual facts, Percentages and Ratios.

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:

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:

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.

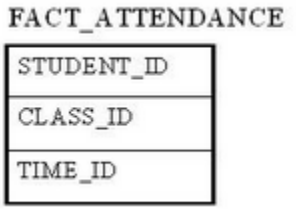
Types of Fact Tables:
Based on the above classifications, there are two types of fact tables:

a) Cumulative: This type of fact table describes what has happened over a period of time.
E.g. 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.

b) 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.

c) Factless: A **Factless Fact Table** is a fact table that does not contain fact. They contain only dimensional keys and it captures events that happen only at information level but not included in the calculations level just information about an event that happen over a period.

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:

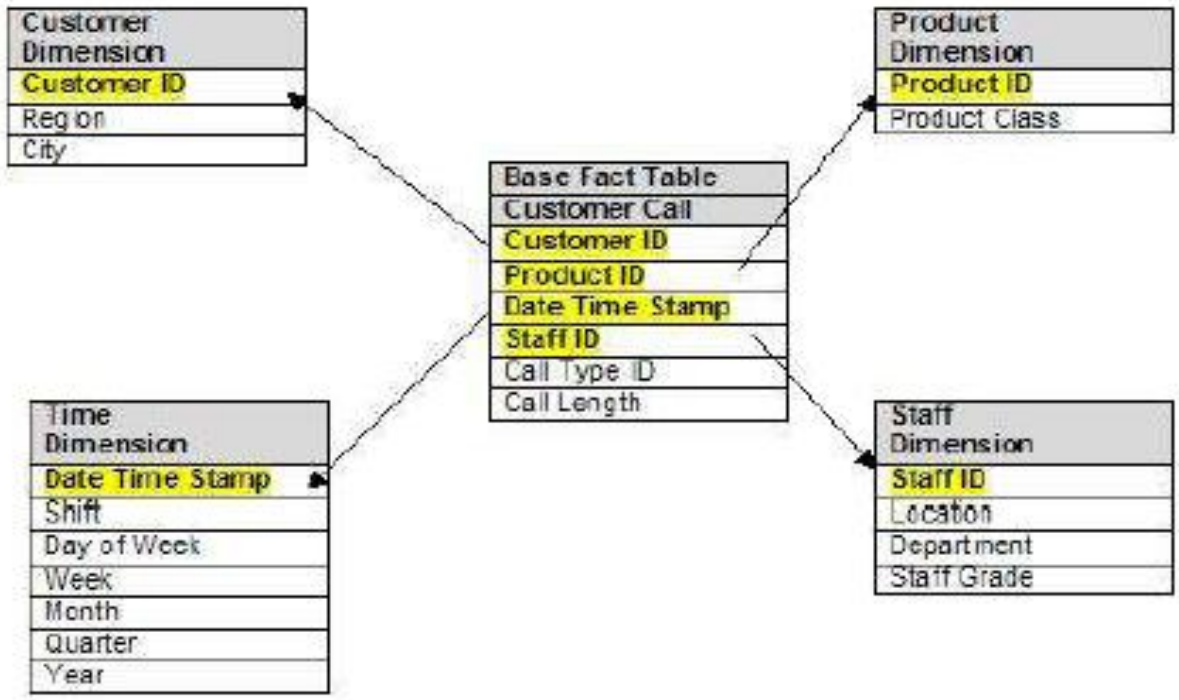


Factless fact tables offer the most flexibility in data warehouse design. For example, one can easily answer the following questions with this Factless fact table:

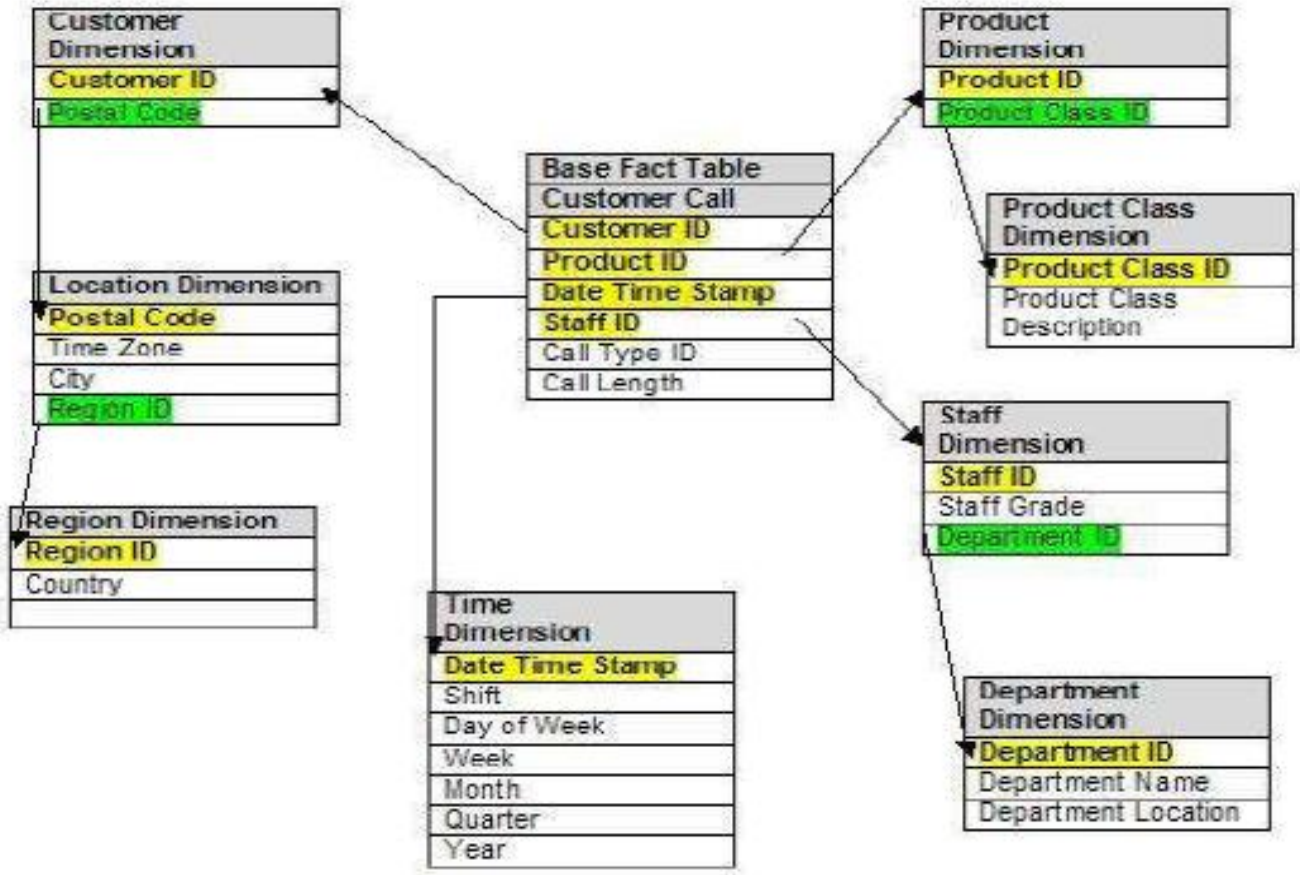
- How many students attended a particular class on a particular day?
- How many classes on average does a student attend on a given day?
- Without using a Factless fact table, we will need two separate fact tables to answer the above two questions. With the above Factless fact table, it becomes the only fact table that's needed.

1) Star Schema:

a) The Star Schema consists of one or more fact tables referencing any number of Dimension tables. The primary key in each dimension table is related to a foreign key in the fact table.
b) The Star Schema separates business process data into Facts, which hold the measurable, quantitative data about a business, and Dimensions which are descriptive attributes related to fact data.
c) Star Schemas are Denormalized, meaning the normal rules of normalization applied to transactional relational databases are relaxed during Star Schema design and implementation
d) A star schema can be simple or complex. A simple star consists of one fact table; a complex star can have more than one fact table.



2) Snowflake Schema: The snowflake schema is an extension of the star schema, where each point of the star explodes into more points. In a star schema, each dimension is represented by a single dimensional table, whereas in a snowflake schema, that dimensional table is normalized into multiple lookup tables, each representing a level in the dimensional hierarchy.



For example, the Time Dimension that consists of 2 different hierarchies:
1. Year → Month → Day
2. Week → Day
We will have 4 lookup tables in a snowflake schema: A lookup table for year, a lookup table for month, a lookup table for week, and a lookup table for day. Year is connected to Month, which is then connected to Day. Week is only connected to Day.

The main advantage of the snowflake schema is the improvement in query performance due to minimized disk storage requirements and joining smaller lookup tables.

The main disadvantage of the snowflake schema is the additional maintenance efforts needed due to the increase number of lookup tables.

Difference between View and Materialized View:

View: store the SQL statement in the database and let you use it as a table. Every time you access the view, the SQL statement executes.

Materialized view: stores the results of the SQL in table form in the database. SQL statement only executes once and after that every time you run the query, the stored result set is used. Pros include quick query results.

1) Moment of Execution:

A view's SQL is executed at run-time. The results are fetched from the view's base tables when the view is queried.

A materialized view (called snapshot in older Oracle versions) is a "pre-answered" query – the query is executed when the materialized view is refreshed. Its result is stored in the database and the query only browses the result.

2) Space:

A view occupies no space (other than that for its definition in the data dictionary).

A materialized view occupies space. It exists in the same way as a table: it sits on a disk and could be indexed or partitioned.

3) Freshness of Output:

A view's output is built on the fly; it shows real-time data from the base tables being queried.

A materialized view does not reflect real-time data. The data is only as up to date as the last time the materialized view was refreshed.

4) Where to Use:

A view is best used when:

You want to hide the implementation details of a complex query

You want to restrict access to a set of rows/columns in the base tables

A materialized view is best used when:

You have a really big table and people do frequent aggregates on it, and you want fast response

You don't mind the result being a little out of date, or your application data has more queries than updates (as in a BI/data warehousing system)

Syntax (Oracle):

CREATE MATERIALIZED VIEW <View_Name>

REFRESH FAST START WITH SYSDATE

*NEXT SYSDATE + 1AS SELECT * FROM <Table_Name>;*