ALL ABOUT DATA ENGINEERING

CRUSH YOUR DATA WAREHOUSING INTERVIEWS

Section 1: Introduction

Data is nothing but raw and unprocessed facts and statistics stored or free-flowing over a network. Data becomes **information** when it is processed, turning it into something meaningful. Collecting and storing data for analysis is a human activity and we have been doing it for thousands of years. In order to effectively work with huge amounts of data in an organized and efficient manner, databases were invented.

> *A **Database** is a collection of related data/information:*
>
> - ***organized** in a way that data can be easily accessed, managed, and updated.*
> - ***captures** information about an organization or organizational process*
> - ***supports** the enterprise by continually updating the database*

In database management, there are two main types of databases: operational databases and analytical databases.

Operational databases are primarily used in **online transactional processing (OLTP)** scenarios, where there is a need to collect, modify, and maintain data daily. This type of database stores dynamic data that changes constantly, reflecting up-to-the-minute information.

On the other hand, analytical databases are used in **online analytical processing (OLAP)** scenarios to store and track historical and time-dependent data. They are valuable for tracking trends, analyzing statistical data over time, and making business projections. Analytical databases store static data that is rarely modified, reflecting a point-in-time snapshot. Although analytical databases often use data from operational databases as their main source, they serve specific data processing needs and require different design methodologies.

	OLTP	OLAP
Characteristics	Handles a large number of small transactions with simple queries	Handles large volumes of data with complex queries
Operations	Based on INSERT, UPDATE, and DELETE commands	Based on SELECT commands to aggregate data for reporting
Response time	Milliseconds	Seconds, minutes, or hours depending on the amount of data to process
Source		Aggregated data from transactions **Multiple** data sources Dimension tables can be connected from SQL Database, CSV files, and more for analysis purposes

	OLTP	OLAP
Purpose	Control and run essential business operations in real-time	Plan, solve problems, support decisions, discover hidden insights
Data updates	Short, fast updates initiated by the user	Data periodically refreshed with scheduled, long-running batch jobs
Space requirements	Generally small if historical data is archived	Generally large due to aggregating large datasets
Backup and recovery	Regular backups are required to ensure business continuity	Lost data can be reloaded from the OLTP database as needed
Data view	Lists day-to-day business transactions	Multi-dimensional view of enterprise data
User examples	Customer-facing personnel, clerks, online shoppers	Knowledge workers such as data analysts, business analysts, and executives
Database design	**Normalization** concept applies and architecture is generally **SNOWFLAKE** schema	Strict adherence to normalization is not followed, **STAR** schema is followed (i.e. one or more dimension tables from SQL Database can be merged and/or appended to get a single dimension table in SQL Data Warehouse)

Section 2: Intro to Data Warehousing

What is a Data Warehouse?

A data warehouse is a large, centralized repository of data that is used to support data-driven decision-making and business intelligence (BI) activities. It is designed to provide a single, comprehensive view of all the data in an organization, and to allow users to easily analyze and report on that data.

Data warehouses are typically used to store historical data and are optimized for fast query and analysis performance. They often contain data from multiple sources and may include both structured and unstructured data. Data in a data warehouse is imported from operational systems and external sources, rather than being created within the warehouse itself. Importantly, data is **copied** into the warehouse, not **moved**, so it remains in the source systems as well.

Data warehouses follow a set of rules proposed by Bill Inmon in 1990. These rules are:

1. Integrated: They combine data from different source systems into a unified environment.
2. Subject-oriented: Data is reorganized by subjects, making it easier to analyze specific topics or areas of interest.
3. Time-variant: They store historical data, not just current data, allowing for trend analysis and tracking changes over time.
4. Non-volatile: Data warehouses remain stable between refreshes, with new and updated data loaded periodically in batches. This ensures that the data does not change during analysis, allowing for consistent strategic planning and decision-making.

As data is imported into the data warehouse, it is often restructured and reorganized to make it more useful for analysis. This process helps to optimize the data for querying and reporting, making it easier for users to extract valuable insights from the data.

Why do we need a Data Warehouse?

Primary reasons for investing time, resources, and money into building a data warehouse:

1. Data-driven decision-making: Data warehouses enable organizations to make decisions based on data, rather than solely relying on experience, intuition, or hunches.
2. One-stop shopping: A data warehouse consolidates data from various transactional and operational applications into a single location, making it easier to access and analyze the data.

Data warehouses provide a comprehensive view of an organization's past, present, and potential future/forecast data. They also offer insights into unknown patterns or trends through advanced analytics and Business Intelligence (BI). In conclusion, Business Intelligence and data warehousing are closely related disciplines that provide immense value to organizations by facilitating data-driven decision-making and offering a centralized data repository for analysis.

Data Warehouse vs Data Lake

Let's discuss the similarities and differences between data warehouses and data lakes, two valuable tools in data management.

A data warehouse is often built on top of a relational database, such as Microsoft SQL Server, Oracle, or IBM DB2. These databases are used for both transactional systems and data warehousing, making them versatile data management tools. Sometimes, data warehouses are built on multidimensional databases called "cubes," which are specialized databases.

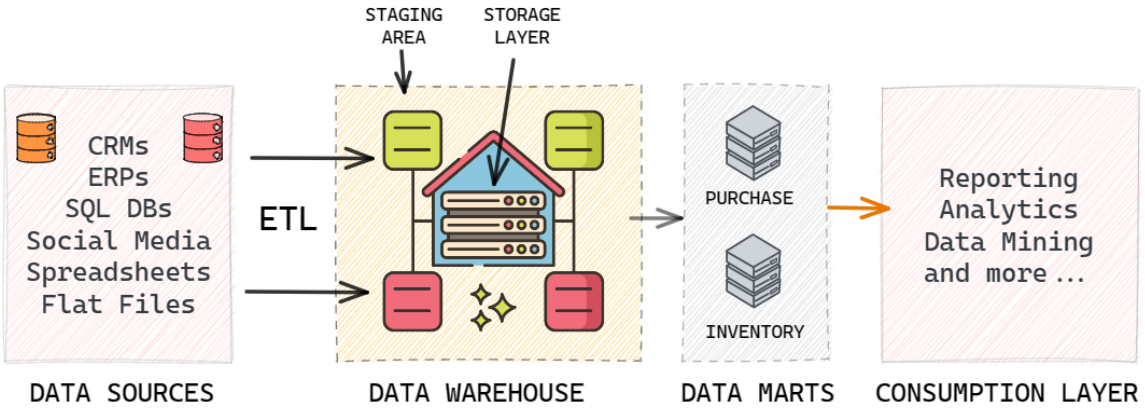
In contrast, a data lake is built on top of a big data environment rather than a traditional relational database. Big data environments allow for the management of extremely large volumes of data, rapid intake of new and changing data, and support a variety of data types (structured, semi-structured, and unstructured).

The lines between the two are increasingly blurred, as SQL, the standard relational database language, can be used on both data warehouses and data lakes. From a user perspective, traditional Business Intelligence (BI) can be performed against either a data warehouse or a data lake.

Parameters	Data Lake	Data Warehouse
Storage	All data is kept irrespective of the source and its structure in its raw form.	Processed data is stored that is extracted, cleaned, and transformed from transactional systems
History	Big data technologies used in data lakes	A data warehouse is often built on top of a relational database
Users	A data lake is ideal for users who indulge in deep analysis. Such users include data scientists who need advanced analytical tools with capabilities such as predictive modelling and statistical analysis.	The data warehouse is ideal for operational users because of being well structured, easy to use, and understandable.
Storage Costs	Data storing in big data technologies are relatively inexpensive than storing data in a data warehouse.	Storing data in a Data warehouse is costlier and more time-consuming.
Task	Data lakes can contain all data and data types; it empowers users to access data prior to the process of transforming and cleansing.	Data warehouses can provide insights into pre-defined questions for pre-defined data types.
Position of Schema	Typically, the schema is defined after data is stored. This offers high agility and ease of data capture but requires work at the end of the process	Typically schema is defined before data is stored. Requires work at the start of the process, but offers performance, security, and integration.
Data processing	Data Lakes use the ELT (Extract Load Transform) process.	Data warehouse uses a traditional ETL (Extract Transform Load) process.

Simple End-to-End Data Warehouse Environment

A simple end-to-end data warehousing environment consists of data sources, a data warehouse, and sometimes, smaller environments called data marts. The process connecting data sources to the data warehouse is known as ETL (Extract, Transform, and Load), a critical aspect of data warehousing.



An analogy to understand this relationship is to think of data sources as suppliers, the data warehouse as a wholesaler that collects data from various suppliers, and data marts as data retailers. Data marts store specific subsets of data tailored for different user groups or business functions. Users typically access data from these data marts for their data-driven decision-making processes.

Section 3: Data Warehouse Architecture

Introduction to DW Architecture Components

In data warehousing, there are various architectural options to plan and execute initiatives. These options include:

1. **Staging layer**: This is the segment within the data warehouse where data is initially loaded before it is transformed and fully integrated for reporting and analytical purposes. There are two types of staging layers: persistent and non-persistent.
2. **Data marts**: These are smaller-scale or more narrowly focused data warehouses.
3. **Cube**: A specialized type of database that can be part of the data warehousing environment.
4. **Centralized data warehouse:** This approach uses a single database to store data for business intelligence and analytics.
5. **Component-based data warehousing**: This approach consists of multiple components, such as data warehouses and data marts, that operate together to create an overall data warehousing environment.

Staging layer in a Data Warehouse

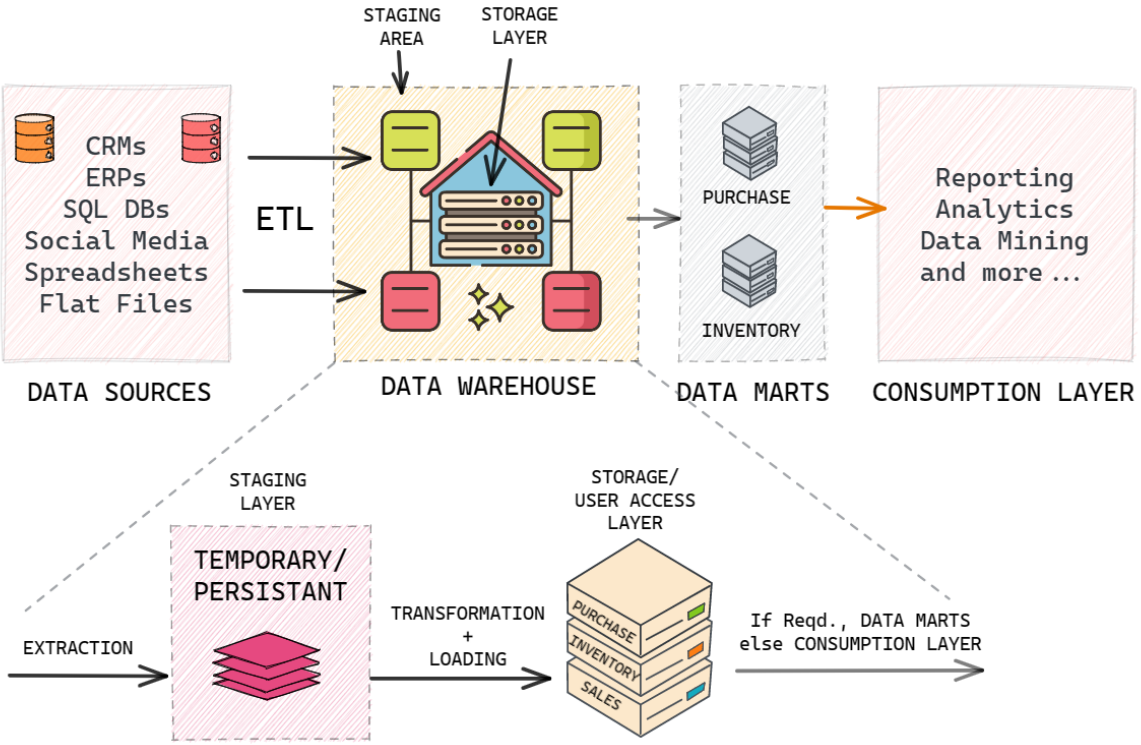
A data warehouse consists of two side-by-side layers: the staging layer and the user access layer. The staging layer serves as a landing zone for incoming data from source applications. The main objective is to quickly and non-intrusively copy the needed data from source applications into the data warehousing environment.

The user access layer is where users interact with the data warehouse or data mart. This layer deals with design, engineering, and dimensional modelling, such as star schema, snowflake schema, fact tables, and dimension tables.

There are two types of staging layers in a data warehouse: non-persistent staging layers and persistent staging layers. Both serve as landing zones for incoming data, which is then transformed, integrated, and loaded into the user access layer.

In a non-persistent staging layer, the data is temporary. After being loaded into the user access layer, the staging layer is emptied. This requires less storage space but makes it more difficult to rebuild the user layer or perform data quality assurance without accessing the original source systems.

In contrast, a persistent staging layer retains data even after it has been loaded into the user access layer. This enables easy rebuilding of the user layer and data quality assurance without accessing the source systems. However, it requires more storage space and can lead to ungoverned access by power users.

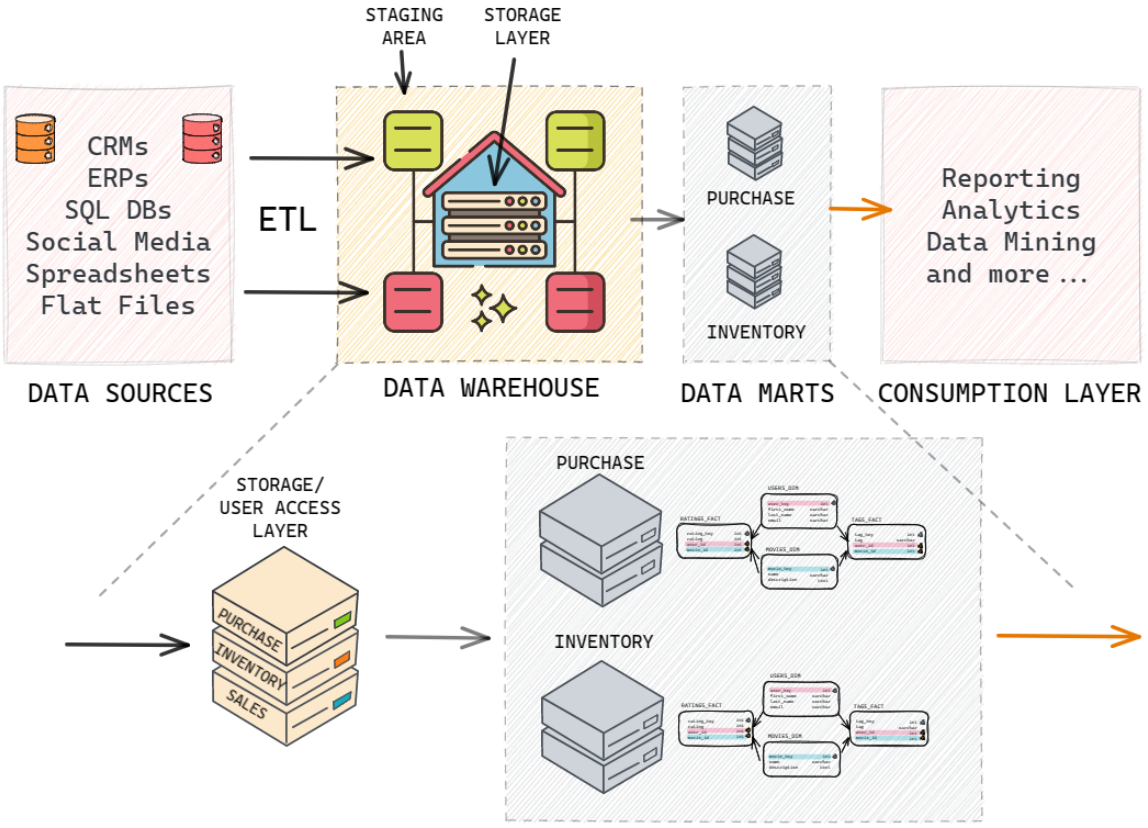


Data Warehouse vs Data Mart

Data marts are additional environments in the data warehousing process, often thought of as data retailers. There are two types of data marts: dependent and independent.

Dependent data marts rely on the existence of a data warehouse to be supplied with data. Independent data marts, on the other hand, draw data directly from one or more source applications and do not require a data warehouse. Independent data marts can be thought of as small-scale data warehouses with data organized internally.

The difference between a data warehouse and an independent data mart lies in the number of data sources and business requirements. Data warehouses typically have many sources (~10-50), while independent data marts have much fewer data sources. If a business requirement dictates the creation of well-defined business units due to the huge size of a data warehouse, data marts can be added to the architecture to create units specific to business requirements such as purchase-specific data mart, inventory-specific data mart, etc. Other than that, the properties of a data warehouse and an independent data mart are quite similar.

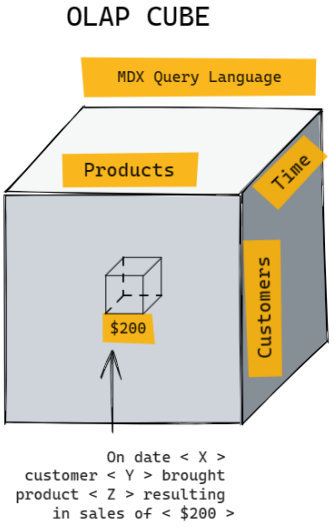


Cubes in Data Warehouse environment

Unlike relational database management systems (RDBMS), cubes are specialized databases with an inherent awareness of the dimensionality of their data contents. Cubes offer some advantages, such as fast query response times and suitability for modest data volumes (around 100 GB or less).

However, they are less flexible compared to RDBMS, as they have a rigid organizational structure for data. Structural changes to the data can be complex and time-consuming. Additionally, there is more vendor variation with cubes than with RDBMS.

In modern data warehousing environments, it is common to see a combination of relational databases and multi-dimensional databases. This mix-and-match approach can provide a powerful combination for organizations.



Figuring out the appropriate approach to DW

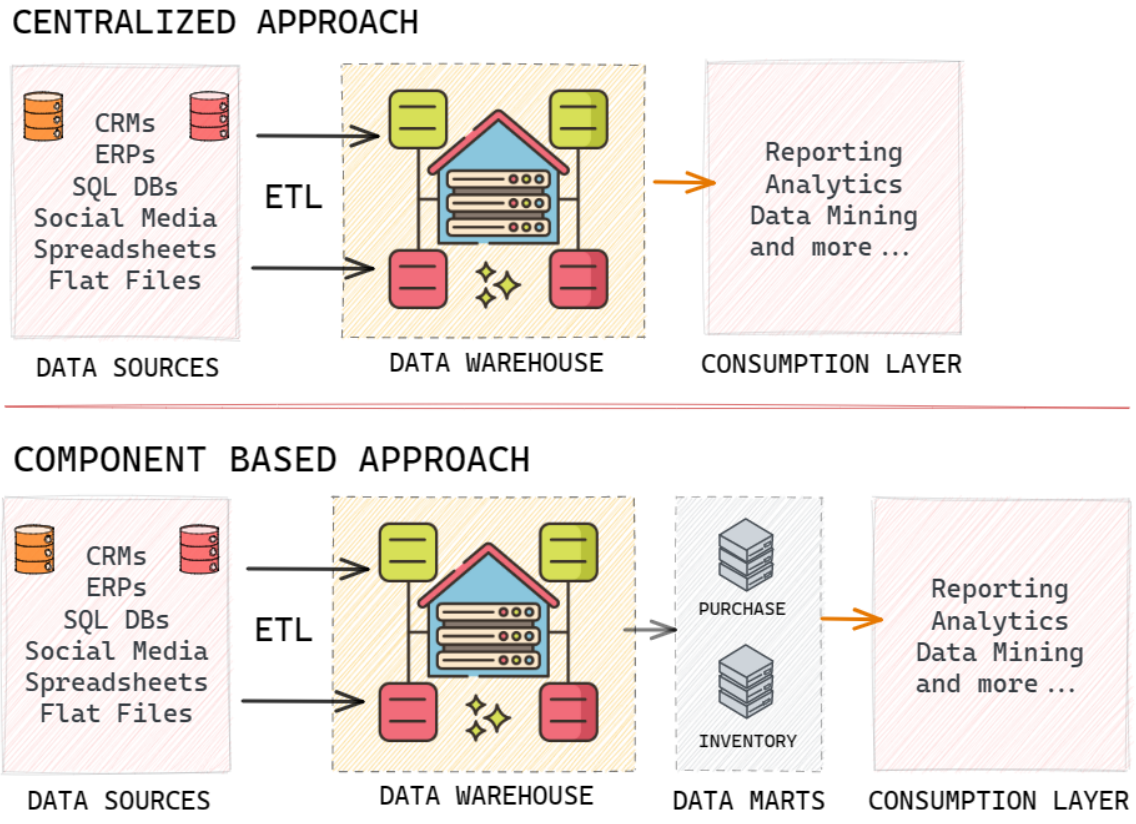
In summary, there are two main paths to choose from when deciding on a data warehouse environment: a centralized approach or a component-based approach.

A centralized approach, such as an Enterprise Data Warehouse (EDW) or a Data Lake, offers a single, unified environment for data storage and analysis. This enables one-stop shopping for all data needs but requires a high degree of cross-

organizational cooperation and strong data governance. It also carries the risk of ripple effects when changes are made to the environment.

A component-based approach, on the other hand, divides the data environment into multiple components such as data warehouses and data marts. This offers benefits like isolation from changes in other parts of the environment and the ability to mix and match technology. However, it can lead to inconsistent data across components and difficulties in cross-integration.

The choice of data warehouse environment ultimately depends on the specific needs and realities of each organization, and the decision-making process required.



Section 4: Data Warehousing Design: Building

The primary objective of a data warehouse is to enable data-driven decisions, which can involve past, present, future, and unknown aspects through different types of business intelligence and analytics. **Dimensional modelling** is particularly suited for basic reporting and online analytical processing (OLAP). However, if the data warehouse is primarily supporting predictive or exploratory analytics, different data models may be required. In such cases, only some aspects of the data may be structured dimensionally, while others will be built into forms suitable for those types of analytics.

The Basic Principles of Dimensionality

Dimensionality in data warehousing refers to the **organization of data in a structured way** that facilitates efficient querying and analysis. The two main components of dimensionality are **facts/measurements** and **dimensional** context, which are essential for understanding the data and making data-driven decisions.

The main benefits of using a dimensional approach in data warehousing include:

- **Improved query performance**: By organizing data in a structured manner, queries can be executed more efficiently, enabling faster data retrieval and analysis.
- **Simplified data analysis**: Dimensional models make it easier for analysts to understand the relationships between data elements and perform complex analyses.
- **Enhanced data visualization**: Organized data enables better visualization of patterns, trends, and relationships, which in turn helps in making informed decisions.

Facts/Measurements:

These are the quantifiable aspects of the data, such as sales amounts, user counts, or product quantities. They represent the actual values or metrics that are being analyzed. It is important to differentiate between a data warehousing fact and a logical fact, as the former is a numeric value while the latter is a statement that can be evaluated as true or false.

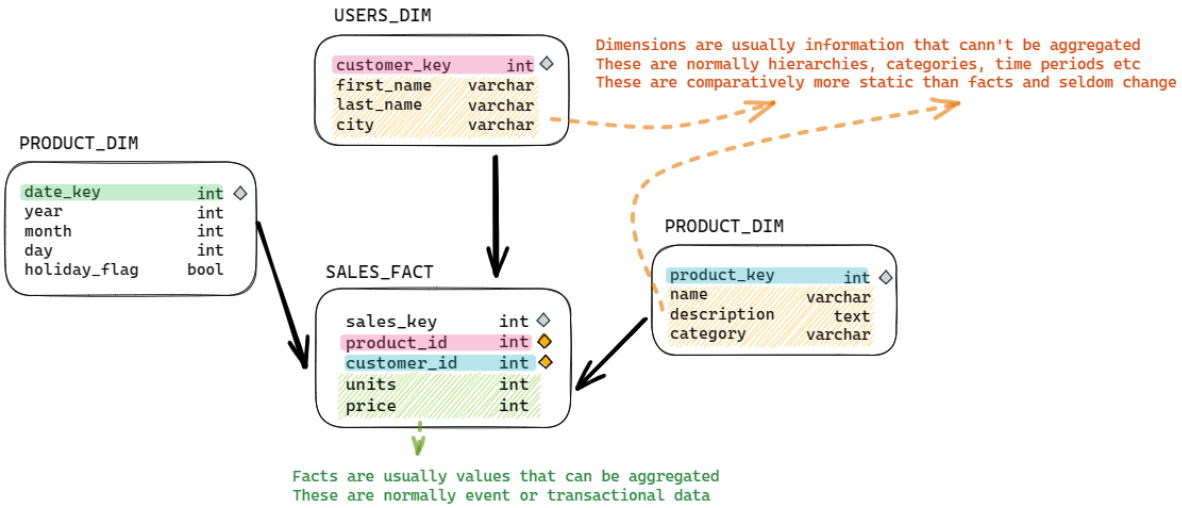
Fact tables store facts in a relational database used for a data warehouse. They are distinct from facts themselves, and there are specific rules for determining which facts can be stored in the same fact table. Fact tables can be of different types and can be connected with dimension tables to link measurements and context in the data warehouse.

Dimensional context:

This is the descriptive information that provides context to the measurements, helping to understand how the data is organized and what it represents. Dimensions often include categories, hierarchies, or time periods, which enable data analysts to slice and dice the data in various ways. Common dimensions include geography (e.g., country, region, city), time (e.g., year, quarter, month), and product categories (e.g., electronics, clothing, food).

In a relational database, dimensions are stored in dimension tables. However, the terms "dimension" and "dimension table" are sometimes used interchangeably, although they are technically different.

This interchangeability of terms is due to the differences between star and snowflake schemas. In a star schema, all information about multiple levels of a hierarchy (e.g., product, product family, and product category) is stored in a single-dimension table, which is often called the **product dimension table**. In a snowflake schema, each level of the hierarchy is stored in a separate table, resulting in **three dimensions and three dimension tables**.



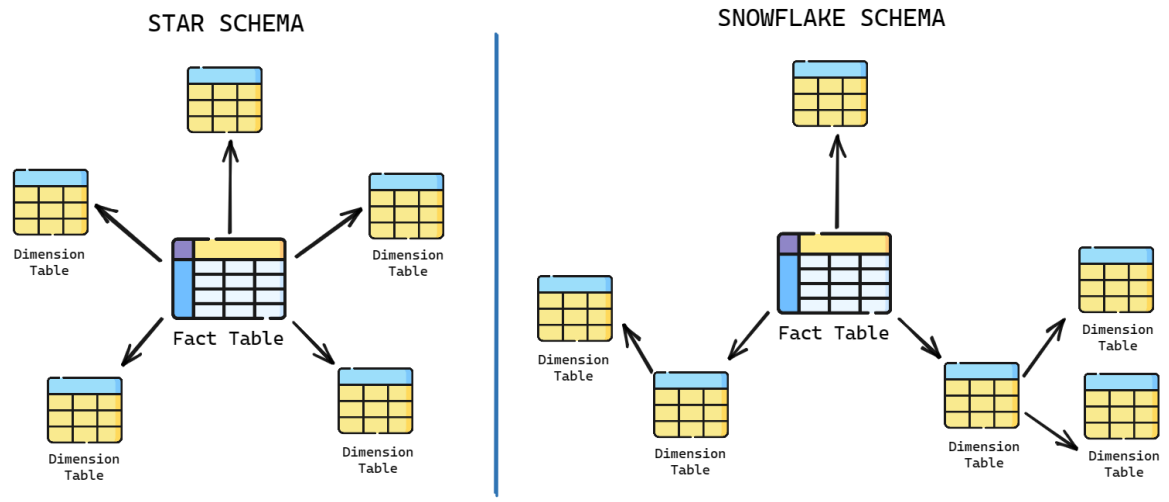
Star schema and snowflake schema are two popular approaches to dimensional modelling. Both involve organizing data into fact tables (which store measurements) and dimension tables (which store contextual information).

Star Schema vs Snowflake Schema

Star schema and snowflake schema are two different structural approaches to building dimensional models in a data warehouse. Both are used to represent fact tables and dimension tables, which are essential for dimensional analysis in OLAP

and business intelligence.

Star Schema	Snowflake Schema
All dimensions along a given hierarchy in one dimension table	Each dimension/dimensional table in its own table
One level away from the fact table thus requiring straightforward relationships	One or more levels away from the fact table thus requiring complex relationships
Fewer database joins at the cost of more data storage requirements	More database joins while consuming much less storage



Both star and snowflake schemas have the same number of dimensions, but their table representations differ. The choice between star and snowflake schema depends on factors such as the complexity of the database relationships, storage requirements, and the number of joins needed for data analysis.

Database Keys for Data Warehousing

Database keys play a crucial role in data warehousing. They help maintain relationships, ensure data integrity, and improve query performance.

1. **Primary Keys:** A primary key is a unique identifier for each row in a database table. It can be a single column or a combination of columns that guarantee the uniqueness of each record. Primary keys are essential for maintaining data integrity, preventing duplicate entries, and providing a unique reference point for other tables.
2. **Foreign Keys:** A foreign key is a column or set of columns that reference the primary key of another table. The purpose of foreign keys is to maintain the relationships between tables and ensure referential integrity. This means that a foreign key value must always match an existing primary key value in the related table or be NULL. Foreign keys help with data consistency, reduce redundancy, and make it easier to enforce constraints and retrieve related information.

Natural and Surrogate Keys

1. **Natural Keys:** Natural keys (also known as business keys or domain keys) are derived from the source system data and are used to identify a record based on its inherent attributes. They can be cryptic or easily understandable, depending on the nature of the data. Natural keys have their limitations, as they can be subject to change, and sometimes they may not guarantee uniqueness. However, keeping natural keys in dimension tables as secondary keys can be beneficial for traceability, data validation, and troubleshooting purposes.

   For example, in a table storing employee details, the employee's social security number could serve as a natural key. Similarly, a customer's email address could be a natural key in a customers table (assuming each customer has a unique email address)

2. **Surrogate Keys:** Surrogate keys are system-generated, unique identifiers that have no business meaning. They are used as primary keys in data warehousing to ensure data integrity and simplify the relationships between tables. Surrogate keys are usually auto-incremented numbers. Surrogate keys are preferred over natural keys for several reasons, including the ability to handle changes in the source data, improved performance, and the support for slowly changing dimensions.

   For example, consider a table storing customer details. Even though each customer might have a unique email address, we might choose to use a surrogate key, such as CustomerID, to uniquely identify customers. So, a customer might be

assigned an ID like 1, 2, 3, and so on, with no relation to the actual data about the customer.

Comparison:

- **Meaning**: Natural keys have a business meaning, while surrogate keys do not.
- **Change**: Natural keys can change (for example, a person might change their email address), while surrogate keys are static and do not change once assigned.
- **Simplicity**: Natural keys can be complex if they're composed of multiple attributes, while surrogate keys are simple (usually just a number).
- **Performance**: Surrogate keys can improve query performance because they're usually indexed and simpler to manage.
- **Data Anonymity**: Surrogate keys provide a level of abstraction and data protection, particularly useful when sharing data without exposing sensitive information.

When designing a data warehouse, it is crucial to make the right choices regarding key usage. As a general guideline, use surrogate keys as primary and foreign keys for better data integrity and handling of data changes. Keep natural keys in dimension tables as secondary keys to maintain traceability and ease troubleshooting. Finally, discard or do not use natural keys in fact tables, as they can lead to redundancy and complexity in managing the data warehouse.

Section 5: Design Facts, Fact Tables, Dimensions, and Dimension Tables

Different Forms of Additivity in Facts

Additivity is an important concept in data warehousing that pertains to the rules for storing facts in fact tables. Facts can be additive, non-additive, or semi-additive.

Additive facts can be added under all circumstances. For example, faculty members' salaries or students' credit hours can be added together to find a total salary or total credit hours completed. These are valid uses of addition in the context of additive facts.

Non-additive facts, on the other hand, cannot be added together to produce a valid result. Examples include grade point averages (GPA), ratios, or percentages. To handle non-additive facts, it's best to store the underlying components in fact tables rather than the ratio or average itself. You can store the non-additive fact at an individual row level for easy access but should prevent users from adding them up. Instead, you can calculate aggregate averages or ratios from the totals of the underlying components.

Semi-additive facts fall somewhere between additive and non-additive facts, as they can sometimes be added together while at other times they cannot. Semi-additive facts are often used in periodic snapshot fact tables, which will be covered in more detail in future discussions.

NULLs in Facts

A **NULL** represents a **missing** or **unknown** value; **Note** that a Null **does not** represent a zero, a character string of one or more blank spaces, or a “zero-length” character string.

The major drawback of Nulls is their adverse effect on mathematical operations. Any operation involving a Null evaluates to Null. This is logically reasonable—if a number is unknown, then the result of the operation is necessarily unknown

```
SELECT 5 + NULL; -- Result: NULL
SELECT 10 - NULL; -- Result: NULL
SELECT 3 * NULL; -- Result: NULL
SELECT 15 / NULL; -- Result: NULL
```

```
SELECT 8 % NULL; -- Result: NULL

-- Assume a table 'sales' with a column 'revenue' containing the following values:
--(10, NULL, 15)
SELECT SUM(revenue) FROM sales; -- Result: 25 (ignores NULL values)
SELECT AVG(revenue) FROM sales; -- Result: 12.5 (ignores NULL values)


SELECT COUNT(revenue) FROM sales; -- Result: 2 (ignores NULL values)
SELECT COUNT(*) FROM sales; -- Result: 3 (includes NULL values)
```

To handle NULL values in mathematical operations in MySQL, you can use functions like COALESCE or IFNULL to replace NULL values with a default value. Here's an example:

```
-- Replace NULL values with 0 and perform addition
SELECT 5 + COALESCE(NULL, 0); -- Result: 5
SELECT 5 + IFNULL(NULL, 0); -- Result: 5
```

NULLs also should be avoided in the foreign key column of fact tables. Instead of using NULLs, we can assign default values to columns in the fact table. we can choose a value that represents the absence of data or a not applicable scenario. However, even though this approach can simplify querying and analysis but it may introduce ambiguity if the default value can be mistaken for actual data.

The Four Main Types of Data Warehousing Fact Tables

In data warehousing, there are four main types of fact tables used to store different types of facts or measurements.

1. **Transaction Fact Table:** This type of fact table records facts or measurements from transactions occurring in source systems. It manages these transactions at an appropriate level of detail in the data warehouse.
2. **Periodic Snapshot Fact Table:** This table tracks specific measurements at regular intervals. It focuses on periodic readings rather than lower-level transactions.
3. **Accumulating Snapshot Fact Table:** Similar to the periodic snapshot fact table, this table shows a snapshot at a point in time. However, it is specifically used to track the progress of a well-defined business process through various stages.
4. **Factless Fact Table:** This type of fact table has two main uses. Firstly, it records the occurrence of a transaction even if there are no measurements to record. Secondly, it is used to officially document coverage or eligibility relationships, even if nothing occurred as part of that particular relationship.

The Role of Transaction Fact Tables

Transactional fact tables are the most common type of fact table used in a data warehouse. These fact tables store transaction-level data, meaning that each row in the table represents a single event or transaction. Transactional fact tables are useful for capturing and analyzing detailed, granular data about the individual events that occur in a business. When dealing with transaction fact tables, we can store multiple facts together in the same table if they meet the following rules:

1. Facts are available at the same grain or level of detail.
2. Facts occur simultaneously.

In a transactional fact table, each row is associated with one or more dimension tables. These dimension tables help provide context and additional information about the transaction. Facts within the transactional fact table are usually numeric and additive, such as sales amount, quantity sold, or total revenue.

Example:

Consider a retail store that wants to track sales transactions. In this example, the transactional fact table could be called 'sales_fact'. For each sale transaction, the following information might be stored:

- Date of sale (Date Dimension)
- Store location (Store Dimension)
- Product sold (Product Dimension)
- Customer information (Customer Dimension)
- Payment method (Payment Method Dimension)
- Quantity sold (Fact)

- Sales amount (Fact)
- Discount applied (Fact)

Here is a simplified version of what the 'sales_fact' table might look like:

transaction_id (PK)	date_key (FK)	store_key (FK)	product_key (FK)	customer_key (FK)	payment_key (FK)	quantity (fact)	sales_amount (fact)
1	1	1	1	1	1	2	50
2	1	2	2	2	2	1	25
3	1	1	3	1	1	3	75

In this example, the transactional fact table contains three fact columns (quantity, sales_amount, and discount) and five foreign key columns that connect to the respective dimension tables (date_key, store_key, product_key, customer_key, and payment_key).

Remember to use surrogate keys (not natural keys) as primary and foreign keys to maintain relationships between tables. These surrogate keys serve as foreign keys that point to primary keys in the respective dimension tables, which provide the context needed for decision-making based on the data.

The Role of Periodic Snapshot Fact Tables

Periodic Snapshot fact tables are used in data warehouses to capture the state of a particular measure at specific points in time, such as daily, weekly, or monthly snapshots. These fact tables help analyze trends and changes over time, which can be valuable for understanding the business's performance and making informed decisions.

In a periodic snapshot fact table, each row represents the state of a specific measure at a particular point in time. The table typically includes one or more dimension keys that provide context for the snapshot, such as date, product, or customer information.

Example:

Consider a bank that wants to track its customers' account balances on a monthly basis. In this example, the periodic snapshot fact table could be called 'monthly_account_balance_fact'. For each customer, the following information might be stored:

1. Snapshot Date (Date Dimension)
2. Customer information (Customer Dimension)
3. Account type (Account Type Dimension)
4. Account balance (Fact)

Here is a simplified version of what the 'monthly_account_balance_fact' table might look like:

snapshot_id	date_key	customer_key	account_type_key	account_balance
1	1	1	1	1000
2	1	2	1	2500
3	1	1	2	5000
4	2	1	1	1200
5	2	2	1	2300
6	2	1	2	5100

In this example, the periodic snapshot fact table contains one fact column (account_balance) and three foreign key columns that connect to the respective dimension tables (date_key, customer_key, and account_type_key). Each row represents the state of a customer's account balance at the end of a specific month.

Using a periodic snapshot fact table, the bank can analyze its customers' account balances in various ways, such as:

1. Average account balance per month
2. Changes in account balances over time
3. Distribution of account balances by account type
4. Customer segments based on account balances

Periodic snapshot fact tables are particularly useful for tracking and analyzing data that changes over time and may not be easily captured in a transactional fact table. A complication with periodic snapshot fact tables is the presence of semi-additive facts.

Periodic Snapshots and Semi-Additive Facts

A semi-additive fact is a type of measure that can be aggregated along some (but not all) dimensions. Typically, semi-additive facts are used in periodic snapshot fact tables.

The classic example of a semi-additive fact is a bank account balance. If you have daily snapshots of account balances, you can't add the balances across the time dimension because it wouldn't make sense - adding Monday's balance to Tuesday's balance doesn't give you any meaningful information. However, you could add up these balances across the account dimension (if you were to aggregate for a household with multiple accounts, for instance) or across a geographical dimension (if you wanted to see total balances for a particular branch or region).

Here's an example:

Date	AccountID	Balance
2023-05-01	1	100
2023-05-01	2	200
2023-05-02	1	150
2023-05-02	2	250
2023-05-03	1	120
2023-05-03	2	240

If we were to add the balances across the time dimension (for example, to try to get a total balance for account 1 for the period from May 1 to May 3), we would get $370, which doesn't represent any meaningful quantity in this context. This is why the balance is a semi-additive fact - it can be added across some dimensions (like the AccountID), but not across others (like the Date).

In contrast, fully additive facts, like a bank deposit or withdrawal amount, can be summed along any dimension. Non-additive facts, like an interest rate, cannot be meaningfully summed along any dimension.

When dealing with semi-additive facts, it's important to ensure that your analysis is using the appropriate aggregation methods for each dimension. Depending on the specific requirements of your analysis, you might need to use the maximum, minimum, first, last, or average value of the semi-additive fact for a given period, rather than the sum.

The Role of Accumulating Snapshot Fact Tables

These tables are used to track the progress of a business process (e.g. order fulfillment) with formally defined stages, measuring elapsed time spent in each stage or phase. They typically include both completed phases and phases that are in progress. Accumulating snapshot fact tables also introduce the concept of multiple relationships from a fact table back to a single dimension table.

These typically include multiple date columns representing different milestones in the life cycle of the event and one or more dimension keys providing context for the snapshot, such as product, customer, or project information. The fact columns represent the measures associated with each stage or milestone.

Example:

Consider a company that wants to track its sales orders from order placement to delivery. In this example, the accumulating snapshot fact table could be called 'sales_order_fact'. For each sales order, the following information might be stored:

- Sales Order ID (Primary key)
- Customer information (Customer Dimension)
- Product Information (Product Dimension)
- Order date (Date Dimension)
- Shipping date (Date Dimension)
- Delivery date (Date Dimension)
- Order amount (Fact)
- Shipping cost (Fact)

Here is a simplified version of what the 'sales_order_fact' table might look like:

order_id	customer_key	product_key	order_date_key	shipping_date_key	delivery_date_key	order_amount	shipping_c
1	1	1	1	2	4	100	10
2	2	1	1	3	5	200	15
3	1	2	2	4	6	150	12

In this example, the accumulating snapshot fact table contains two fact columns (order_amount and shipping_cost) and five foreign key columns that connect to the respective dimension tables (customer_key, product_key, order_date_key, shipping_date_key, and delivery_date_key). Each row represents the life cycle of a sales order, from order placement to delivery.

Using an accumulating snapshot fact table, the company can analyze its sales order data in various ways, such as:

1. Average time between order placement and shipping
2. Average time between shipping and delivery
3. Total shipping cost by product or customer
4. Comparison of actual delivery dates to estimated delivery dates

In the accumulating snapshot fact table, there are multiple relationships with the same dimension table, such as the date dimension. This is because the table needs to track various dates related to the business process, like the order date, shipping date, delivery date, and so on. Similarly, multiple relationships with the employee dimension may also be required to account for different employees responsible for different phases of the process.

Comparing the three main types of fact tables

Type	Transactional	Periodic Snapshot	Accumulating Snapshot
Grain	1 row = 1 transaction	1 row =1 defined period (plus other dimensions)	1 row = lifetime of process/event
Date Dimension	1 Transaction date	Snapshot date (end of period)	Multiple snapshot dates
No of Dimension	High	Low	Very High
Size	Largest (most detailed grain)	Middle (less detailed grain)	Lowest (highest aggregation)

Why a Factless Fact Table isn't a Contradiction in Terms

Fact tables are used in data warehouses to store facts or measurements that need to be tracked. A fact is different from a fact table, and there are various types of fact tables, each with a specific purpose and usage.

Factless fact tables are used in data warehousing to capture many-to-many relationships among dimensions. They are called "factless" because they have no measures or facts associated with transactions. Essentially, they contain only dimensional keys. It's the structure and use of the table, not the presence of numeric measures, that makes a table a fact table.

Factless fact tables are used in two main scenarios:

1. **Event Tracking**: In this case, the factless fact table records an event. For example, consider a table that records student attendance. The table might contain a student key, a date key, and a course key. There are no measures or facts in this table, only the keys of the students who attended classes on certain dates. The absence of facts is not a problem; simply capturing the occurrence of the event provides valuable information.
2. **Coverage or Bridge Table**: This kind of factless fact table is used to model conditions or coverage data. For example, in a health insurance data warehouse, a factless fact table could be used to track eligibility. The table might contain keys for patient, policy, and date, and it would show which patients are covered by which policies on which dates.

Here's an example of a factless fact table for an attendance system:

Student_ID	Course_ID	Date (FK)
1	101	2023-5-1
2	101	2023-5-1
1	102	2023-5-2
3	103	2023-5-2

This table does not contain any measures or facts. However, it provides valuable information about which student attended which course on which date. We can count the rows to know the number of attendances, join this with other tables to get more details, or even use this for many other analyses. The presence of a row in the factless fact table signifies that an event occurred.

Dimension Tables

Dimension tables in a data warehouse contain the descriptive attributes of the data, and they are used in conjunction with fact tables to provide a more complete view of the data. These tables are typically designed to have a denormalized structure for simplicity of data retrieval and efficiency in handling user queries, which is often essential in a business intelligence context.

Here are some key characteristics and components of dimension tables:

1. Descriptive Attributes: These are the core of a dimension table. These attributes provide context and descriptive characteristics of the dimensions. For example, in a "Customer" dimension table, attributes might include customer name, address, phone number, email, etc.

2. Dimension Keys: These are unique identifiers for each record in the dimension table. These keys are used to link the fact table to the corresponding dimension table. They can be either natural keys (e.g., a customer's email address) or surrogate keys (e.g., a system-generated customer ID).

3. Hierarchies: These are often present within dimensions, providing a way to aggregate data at different levels. For example, a "Time" dimension might have a hierarchy like Year > Quarter > Month > Day.

Example of a Dimension Table: Customer

CustomerID (PK)	CustomerName	Gender	DateOfBirth	City	State	Country
1	John Doe	Male	1980-01-20	NY	NY	USA
2	Jane Smith	Female	1990-07-15	LA	CA	USA
3	Tom Brown	Male	1985-10-30	SF	CA	USA
4	Emma Davis	Female	1995-05-25	TX	TX	USA

In this Customer dimension table, `CustomerID` is the surrogate key which uniquely identifies each customer. Other fields like `CustomerName`, `Gender`, `DateOfBirth`, `City`, `State`, and `Country` provide descriptive attributes for the customers.

The "Sales" fact table in this data warehouse might then have a foreign key that links to the CustomerID in the dimension table. This allows users to analyze sales data not just in terms of raw numbers, but also in terms of the customers.

In summary, dimension tables provide the "who, what, where, when, why, and how" context that surrounds the numerical metrics stored in the fact tables of a data warehouse. They are essential for enabling users to perform meaningful analysis of the data.

Date Dimension Table

The Date dimension table is a special type of dimension table that is usually found in most data warehouses. This table is used to represent all the possible dates that can be used in the other tables of the data warehouse, and it typically includes several attributes related to the date, such as the day, month, year, quarter, and even weekday/weekend flag, fiscal periods, etc.

This is useful because it allows us to avoid repeating this information in our fact tables, and it also makes it easier to perform time-based analyses and aggregations.

Here is an example of a date dimension table:

DateKey	FullDate	DayOfWeek	DayOfMonth	Month	Year	Quarter	IsWeekend
1	2023-01-01	Sunday	1	January	2023	Q1	True
2	2023-01-02	Monday	2	January	2023	Q1	False
3	2023-01-03	Tuesday	3	January	2023	Q1	False
...	...	...	...	...	...	...	...
365	2023-12-31	Sunday	31	December	2023	Q4	True

In this table, `DateKey` is a surrogate key, `FullDate` is the actual date, `DayOfWeek`, `DayOfMonth`, `Month`, `Year`, `Quarter`, and `IsWeekend` are attributes derived from the full date.

This Date dimension table would be linked to fact tables in the data warehouse using the `DateKey` field. This allows for easy filtering, grouping, and other date-based analysis of the data in the fact tables. For example, you might want to compare sales data by quarter or analyze trends on weekdays versus weekends. The Date dimension table makes these types of analyses possible.

▼ SQL Code for creating data dimension tables

```
DROP TABLE date_dim;

CREATE TABLE date_dim
(
  date_key              INT NOT NULL,
  date                    DATE NOT NULL,
  weekday               VARCHAR(9) NOT NULL,
  weekday_num           INT NOT NULL,
```

```
  day_month             INT NOT NULL,
  day_of_year           INT NOT NULL,
  week_of_year          INT NOT NULL,
  iso_week                CHAR(10) NOT NULL,
  month_num             INT NOT NULL,
  month_name            VARCHAR(9) NOT NULL,
  month_name_short        CHAR(3) NOT NULL,
  quarter                   INT NOT NULL,
  year                    INT NOT NULL,
  first_day_of_month    DATE NOT NULL,
  last_day_of_month     DATE NOT NULL,
  yyyymm                CHAR(7) NOT NULL,
  weekend_indr          CHAR(10) NOT NULL
);

ALTER TABLE public.date_dim ADD CONSTRAINT date_dim_pk PRIMARY KEY (date_key);

CREATE INDEX d_date_date_actual_idx
  ON date_dim(date);



INSERT INTO date_dim
SELECT TO_CHAR(datum, 'yyyymmdd')::INT AS date_key,
       datum AS date,
       TO_CHAR(datum, 'TMDay') AS weekday,
       EXTRACT(ISODOW FROM datum) AS weekday_num,
       EXTRACT(DAY FROM datum) AS day_month,
       EXTRACT(DOY FROM datum) AS day_of_year,
       EXTRACT(WEEK FROM datum) AS week_of_year,
       EXTRACT(ISOYEAR FROM datum) || TO_CHAR(datum, '"-W"IW-') || EXTRACT(ISODOW FROM datu
m) AS iso_week,
       EXTRACT(MONTH FROM datum) AS month,
       TO_CHAR(datum, 'TMMonth') AS month_name,
       TO_CHAR(datum, 'Mon') AS month_name_short,
       EXTRACT(QUARTER FROM datum) AS quarter,
       EXTRACT(YEAR FROM datum) AS year,
       datum + (1 - EXTRACT(DAY FROM datum))::INT AS first_day_of_month,
       (DATE_TRUNC('MONTH', datum) + INTERVAL '1 MONTH - 1 day')::DATE AS last_day_of_month,
       CONCAT(TO_CHAR(datum, 'yyyy'),'-',TO_CHAR(datum, 'mm')) AS mmyyyy,
       CASE
           WHEN EXTRACT(ISODOW FROM datum) IN (6, 7) THEN 'weekend'
           ELSE 'weekday'
           END AS weekend_indr
FROM (SELECT '2010-01-01'::DATE + SEQUENCE.DAY AS datum
      FROM GENERATE_SERIES(0, 7300) AS SEQUENCE (DAY)
      GROUP BY SEQUENCE.DAY) DQ
ORDER BY 1;


SELECT * FROM DATE_DIM
```

NULLs in Dimensions

The presence of NULLs in dimension tables can have several effects, some of which are:

1. **Loss of Information:** When a dimension attribute is NULL, it means that some information is missing. Depending on the attribute, this could result in significant losses of information. For example, if a Customer dimension has NULL values for the 'City' attribute, this could impact any analysis related to the geographical locations of customers.
2. **Complicates Query Writing:** NULL values can make query writing more complex. SQL treats NULLs in a special way. For instance, NULL is not equal to any value, not even to another NULL. Therefore, in order to properly handle NULLs, you

often need to use special SQL constructs like IS NULL or IS NOT NULL, or functions like COALESCE().

3. **Effects Join Operations:** If Foreign Keys with NULL values are used in a JOIN condition, it may lead to fewer results than expected. This is because a NULL does not equal anything, including another NULL. As a result, rows with NULL in the joining column of either table will not be matched.

Hierarchies in Dimension Tables

Hierarchies in dimensions are a way of organizing data that reflects defined levels of granularity and relationships among different levels. For instance, a date dimension might include a hierarchy that goes from year to quarter to month to day. An organization dimension might have a hierarchy that goes from company to division to department to team.

Modelling Hierarchies in Dimensions:

There are two main ways to model hierarchies in dimension tables:

1. Single Table Hierarchies (also known as "flat" hierarchies/ Denormalized Dimension):

In this approach, all levels of the hierarchy are stored in a single table, with each row containing all the hierarchy levels for a particular leaf-level member.

For instance, consider a simple Product dimension with a hierarchy that goes Category → Subcategory → Product. In a single table hierarchy, you might have a table that looks like this:

ProductID	Category	Subcategory	Product
1	Food	Fruit	Apple
2	Food	Fruit	Banana
3	Food	Vegetable	Broccoli
4	Beverage	Soda	Cola
5	Beverage	Juice	AppleJuice

This approach is simple and easy to query, but it can result in a lot of redundant data, especially for large hierarchies. This is normally the preferred approach when creating star-schemas

2. Multiple Table Hierarchies (also known as "snowflake" hierarchies/ Normalized Dimension):

In this approach, each level of the hierarchy is stored in a separate table, with each table having a foreign key that links to the level above it.

Using the same Product dimension example, in a multiple-table hierarchy, you might have three separate tables that look like this:

Category Table:

CategoryID	Category
1	Food
2	Beverage

Subcategory Table:

SubcategoryID	CategoryID	Subcategory
1	1	Fruit
2	1	Vegetable
3	2	Soda
4	2	Juice

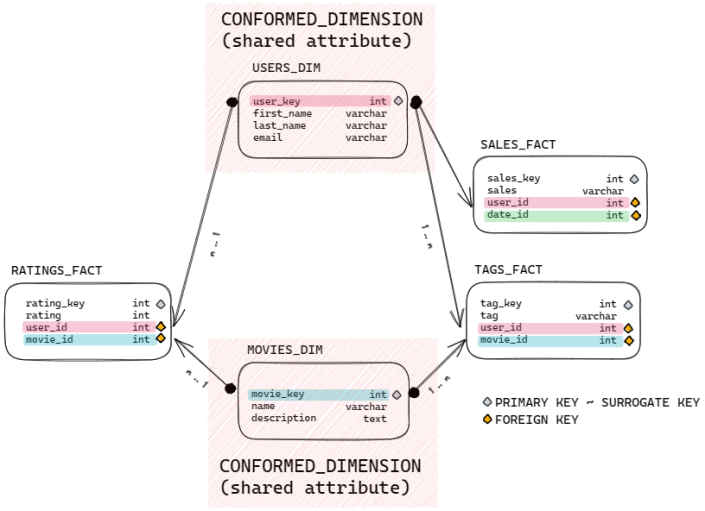
Product Table:

ProductID	SubcategoryID	Product
1	1	Apple
2	1	Banana
3	2	Broccoli
4	3	Cola
5	4	AppleJuice

This approach is more normalized and reduces data redundancy, but it can make querying more complex since it requires joining multiple tables.

Conformed Dimensions

A conformed dimension is a dimension that is shared by multiple fact tables/stars. Essentially, a conformed dimension is a dimension that has the same meaning to every fact table to which it is joined.



Characteristics of Conformed Dimensions:

1. **Consistent definition and understanding:** A conformed dimension has the same meaning and content when being referred to from different fact tables.
2. **Common keys:** Conformed dimensions **can** have the same keys available in each fact table. This enables joining the fact tables on these conformed dimensions, enabling cross-filtering.
3. **Same level of granularity:** Conformed dimensions are at the same level of granularity or detail across the fact tables.

Degenerate Dimension

A type of dimension that is derived from the fact table and does not have its own dimension table, because all the interesting attributes about them are contained in the fact table itself.

Typically, degenerate dimensions are identifiers or numbers that are used for operational or control purposes. For example, an order number, invoice number, or transaction id in a sales fact table. While these identifiers do not have any descriptive attributes of their own (which is why they don't need a separate dimension table), they are extremely useful for tracking and summarizing information.

Example:

Let's consider a retail company's sales fact table. It records data about each sales transaction, including the transaction id, product id, date id, store id, quantity sold, and sales amount.

Sales Fact Table:

TransactionID	ProductID	DateID	StoreID	QuantitySold	SalesAmount
1001	1	101	501	2	100
1002	2	102	502	1	50
1003	3	103	503	5	250
1004	1	104	504	3	150
1005	2	105	505	4	200

Here, the TransactionID is a degenerate dimension. It doesn't have a separate dimension table because there aren't any additional attributes about the transaction that we need to store. However, the TransactionID is important for tracking individual sales transactions and can be used for summarizing data, such as calculating the total sales for each transaction.

Junk Dimension

A junk dimension is a single table composed of a combination of unrelated attributes (flags, indicators, statuses, etc.) to avoid having a large number of foreign keys in the fact table. By grouping these low-cardinality attributes (attributes with very few unique values) into one dimension, we can simplify our model and improve query performance.

For example, let's say we have a retail sales fact table, and we're tracking several binary attributes for each sale:

1. Was the item on sale (Yes/No)?
2. Was a coupon used (Yes/No)?

3. Was the purchase made online (Yes/No)?
4. Was the item returned (Yes/No)?

Instead of adding four separate foreign key columns to our fact table (each corresponding to a dimension table with two rows: "Yes" and "No"), we can combine these attributes into a single junk dimension.

The junk dimension table would look something like this:

JunkKey	OnSale	CouponUsed	OnlinePurchase	ItemReturned
1	Yes	Yes	Yes	Yes
2	Yes	Yes	Yes	No
3	Yes	Yes	No	Yes
4	Yes	Yes	No	No
5	Yes	No	Yes	Yes
...	...	...	...	...

And so on, until all 16 (2^4) possible combinations of these four attributes are represented.

Then, in our fact table, we just include a single foreign key to the Junk Dimension:

TransactionID	ProductID	DateID	JunkKey	QuantitySold	SalesAmount
1001	1	101	2	2	100
1002	2	102	1	1	50
1003	3	103	3	5	250
1004	1	104	4	3	150
1005	2	105	5	4	200

By using this junk dimension, we're able to keep our fact table simpler, and our queries can become faster and more efficient.

Role Playing Dimension

A Role-Playing Dimension is a concept where a single dimension can be used in multiple roles in the same fact table. This happens when a dimension table has multiple relationships with the fact table. Each relationship represents a logically distinct role that the dimension plays in the fact table.

For example, let's take a date dimension. A single date dimension can be used for "Order Date", "Shipping Date", "Delivery Date", etc., in a sales fact table. Each of these dates could reference the same date dimension table but would represent different business concepts.

Here's how it could look:

Date Dimension Table

DateKey	Date	Day	Month	Year
1	2023-01-01	1	1	2023
2	2023-01-02	2	1	2023
3	2023-01-03	3	1	2023
...	...	...	...	...

Sales Fact Table

SalesID	ProductID	OrderDateKey	ShippingDateKey	DeliveryDateKey	Quantity	TotalPrice
1	1	1	2	3	2	100
2	2	2	3	4	1	50
3	1	3	4	5	3	150
4	2	4	5	6	2	100
5	1	5	6	7	1	50

In the above example, the DateKey is role-playing as OrderDateKey, ShippingDateKey, and DeliveryDateKey in the Sales fact table. Each of these roles represents a different business concept, but they all use the same Date dimension table for their values. This is an efficient way to reuse the same dimension in different contexts, avoiding redundancy and making the data model more manageable.

For analysis in SQL, we can create additional *views* for each role

Designing Fact and Dimension Tables

Creating a fact table in a data warehouse involves a series of steps. The exact process might vary slightly depending on the specifics of your project and the tools you're using, but the following steps provide a general outline.

1. **Identify the Business Process**: The first step is to identify the business process that you want to analyze. This could be sales transactions, inventory levels, or customer service calls, for example. The business process will guide the design of the fact table.
2. **Identify the Granularity**: Granularity refers to the level of detail or depth of a fact information. Granularity is the lowest level of information that is stored in a table. For example, if you are building a fact table to analyze sales, the granularity could be at the transaction level (each row represents a sale) or at the item level (each row represents an item within a sale).
3. **Identify the Facts**: Facts are typically numeric values that you wish to analyze. In a sales business process, examples of facts could be Sales Amount, Quantity Sold, Profit, etc. A fact table would typically store these facts along with keys to related dimension tables. The facts are usually the result of some business process event and are what you will be analyzing.
4. **Identify the Dimensions**: Dimensions are descriptive attributes related to the facts. They provide the context for the facts and are often what you group by when analyzing the facts. Typical dimensions include time, location, product, and customer. Each dimension would typically have its own table, which would include a primary key and additional attributes related to the dimension
5. **Design the Fact Table**: The fact table is usually designed next, with a column for each fact identified in Step 3 and a foreign key for each dimension identified in Step 4. The primary key of a fact table is usually a composite key composed of all its foreign keys.
6. **Create Dimension Tables**: Each dimension table should include a primary key that uniquely identifies each row and other descriptive attributes of the dimension. It should also contain other attributes that provide descriptive characteristics related to the dimension.
7. **Create Relationships**: Establish relationships between the fact table and the dimension tables. This is usually done by creating foreign keys in the fact table that correspond to the primary keys in the dimension tables.
8. **Load the Data**: Once the fact table and dimension tables have been created, the next step is to load the data. This typically involves extracting data from source systems, transforming it into the format of the fact and dimension tables (a process known as ETL), and then loading it into the data warehouse.
9. **Test and Validate**: After the data has been loaded, test and validate the fact table to ensure that it accurately represents the business process and supports the desired analysis. This might involve running sample queries, checking for data quality issues, and verifying that the relationships between the fact table and the dimension tables are working correctly.

Let's walk through an example scenario for creating a fact table in a retail business setting:

1. Identify the Business Process:
Let's say our business process is "Sales" - the selling of items to customers.

2. Identify the Granularity:
In this case, we decide that each row in the fact table will represent an individual item sold in a transaction.

3. Identify the Facts:
The facts we want to analyze could include the quantity of items sold, the price of each item, and the total amount for each item sold (quantity * price).

4. Identify the Dimensions:
Possible dimensions for this process include Time (when was the item sold), Product (which product was sold), Store (in which store the sale happened), and Customer (who purchased the item).

5. Design the Fact Table:
Our fact table could look something like this:

Sale_ID (PK)	Time_ID (FK)	Product_ID (FK)	Store_ID (FK)	Customer_ID (FK)	Quantity_Sold	Item_Price	Total_Amount
1	20230501	123	001	456	2	50	100

6. Create Dimension Tables:
For each of the dimensions identified, we'll create a separate table. For instance, the Product dimension table might look like this:

Product_ID (PK)	Product_Name	Product_Category	Product_Price
123	Shirt	Clothing	50

7. Create Relationships:
The relationships between the fact table and dimension tables would be established using the foreign keys in the fact table. For example, the Product_ID in the fact table would link to the Product_ID in the Product dimension table.

8. Load the Data:
We would then extract data from our various sources (maybe point of sale systems, CRM, etc.), transform the data to match the structure of our fact and dimension tables and load the data into these tables.

9. Test and Validate:
Finally, we would need to run queries to ensure our fact table works correctly. For example, we might want to find out the total sales for a specific product in a specific store. We would use our fact and dimension tables to answer this question and verify that the result is as expected.

Remember, this is a simplified example, and real-world scenarios can be more complex, but it should give you a general idea of how the process works.

SQL for Dimension and Fact Tables

Fact tables and dimension tables can be created in star and snowflake schemas using SQL in a data warehouse. The create table statements are used to define the structure of the tables, including columns, data types, primary keys, and foreign keys.

For a star schema dimension table, the primary key is typically a single column, even if the table has multiple hierarchical levels. In a snowflake schema, the non-terminal dimension tables have both primary keys and foreign keys. The terminal dimension table in a snowflake schema has only a primary key, as there is no higher level to reference.

For a transaction-grained fact table, the primary key is a combination of all foreign keys related to the dimension tables. Each foreign key explicitly points to a specific table and column within that table. The SQL syntax for periodic snapshot fact tables is almost identical to that of transaction-grained fact tables.

Regardless of the type of fact table being created, the SQL model follows the same structure: identify keys, combine necessary keys as the primary key, and have a foreign key clause for each key pointing back to the respective dimension tables. The only complication arises when there are multiple relationships with the same dimension, as in accumulating snapshot fact tables or some types of factless fact tables. In such cases, multiple foreign key clauses for each date reference the same date dimension and its key.

Section 6: Managing Data Warehouse History Through Slowly Changing Dimensions

Slowly changing dimensions (SCD) are tables in a dimensional model that handle changes to dimension values over time and not on a set schedule. SCDs present a challenge because they can alter historical data and, in the process, affect the outcome of current analyses.

Over time, it is possible that certain product name changes or maybe a customer changes phone number. This will lead to the case where we will have to change the dimension table to reflect these changes. There are various strategies to tackle the different cases. There are three main types of SCDs: Type 1, Type 2, and Type 3.

Type 1 (Overwrite)

A Type 1 SCD always reflects the latest values, and when changes in source data are detected, the dimension table data is overwritten. No history is kept.

E.g. When a customer's email address or phone number changes, the dimension table updates the customer row with the new values.

CustomerID	FirstName	LastName	EmailAddress	CompanyName	InsertedDate	ModifiedDate
2	Keith	Harris	keith0@aw.com	Progressive Sports	2021-03-20	2021-03-20
3	Donna	Carreras	donna0@aw.com	A Bike Store	2021-03-20	2021-03-20

CustomerID	FirstName	LastName	EmailAddress	CompanyName	InsertedDate	ModifiedDate
2	Keith	Harris	keith0@aw.com	Progressive Sports	2021-03-20	2021-03-20
3	Donna	Carreras	donna0@aw.com	Bikes, Bikes, Bikes	2021-03-20	2021-03-22

Type 2 (Add a new Row):

A Type 2 SCD supports the versioning of dimension members. It includes columns that define the date range validity of the version (for example, `StartDate` and `EndDate`) and possibly a flag column (for example, `IsCurrent`) to easily filter by current dimension members.

Current versions may define an empty end date (or 12/31/9999), which indicates that the row is the current version. The table must also define a **surrogate key** because the business key (in this instance, RepSourceID) won't be unique.

SalesRepID	RepSourceId	FirstName	LastName	Region	StartDate	EndDate	IsCurrent
1	312	Jun	Cao	Southwest	2021-03-20	9999-12-31	True
2	331	Susan	Eaton	Southcentral	2021-03-20	9999-12-31	True

SalesRepID	RepSourceId	FirstName	LastName	Region	StartDate	EndDate	IsCurrent
1	312	Jun	Cao	Southwest	2021-03-20	9999-12-31	True
2	331	Susan	Eaton	Southcentral	2021-03-20	2021-03-21	False
3	331	Susan	Eaton	Southeast	2021-03-22	9999-12-31	True

Instead of putting NULL for the End date, its better to put a future date

▼ **Code to apply type 1 and 2 logic**

```
/*Logic to implement Type 1 and Type 2 updates can be complex, and there are various techniq
ues you can use. For example, you could use a combination of UPDATE and INSERT statements as
shown in the following code example:*/

-- Insert new customers
INSERT INTO dbo.DimCustomer
SELECT stg.CustomerNo,
       stg.CustomerName,
       stg.EmailAddress,
       stg.Phone,
       stg.StreetAddress
FROM dbo.StageCustomers AS stg
WHERE NOT EXISTS
    (SELECT * FROM dbo.DimCustomer AS dim
     WHERE dim.CustomerAltKey = stg.CustomerNo);

-- Type 1 updates (name, email, phone)
UPDATE dbo.DimCustomer
SET CustomerName = stg.CustomerName,
    EmailAddress = stg.EmailAddress,
    Phone = stg.Phone
FROM dbo.StageCustomers AS stg
WHERE dbo.DimCustomer.CustomerAltKey = stg.CustomerNo;

-- Type 2 updates (geographic address)
INSERT INTO dbo.DimCustomer
SELECT stg.CustomerNo AS CustomerAltKey,
       stg.CustomerName,
       stg.EmailAddress,
       stg.Phone,
       stg.StreetAddress,
       stg.City,
       stg.PostalCode,
       stg.CountryRegion
```

```
FROM dbo.StageCustomers AS stg
JOIN dbo.DimCustomer AS dim
ON stg.CustomerNo = dim.CustomerAltKey
AND stg.StreetAddress <> dim.StreetAddress;

/*As an alternative to using multiple INSERT and UPDATE statement, you can use a single MERG
E statement to perform an "upsert" operation to insert new records and update existing ones,
as shown in the following example, which loads new product records and applies type 1 update
s to existing products*/

MERGE dbo.DimProduct AS tgt
    USING (SELECT * FROM dbo.StageProducts) AS src
    ON src.ProductID = tgt.ProductBusinessKey
WHEN MATCHED THEN
    UPDATE SET
        tgt.ProductName = src.ProductName,
        tgt.ProductCategory = src.ProductCategory
        tgt.Color = src.Color,
        tgt.Size = src.Size,
        tgt.ListPrice = src.ListPrice,
        tgt.Discontinued = src.Discontinued
WHEN NOT MATCHED THEN
    INSERT VALUES
        (src.ProductID,
         src.ProductName,
         src.ProductCategory,
         src.Color,
         src.Size,
         src.ListPrice,
         src.Discontinued);

/*Another way to load a combination of new and updated data into a dimension table is to use
a CREATE TABLE AS (CTAS) statement to create a new table that contains the existing rows fro
m the dimension table and the new and updated records from the staging table. After creating
the new table, you can delete or rename the current dimension table, and rename the new tabl
e to replace it.*/

CREATE TABLE dbo.DimProductUpsert
WITH
(
    DISTRIBUTION = REPLICATE,
    CLUSTERED COLUMNSTORE INDEX
)
AS
-- New or updated rows
SELECT  stg.ProductID AS ProductBusinessKey,
        stg.ProductName,
        stg.ProductCategory,
        stg.Color,
        stg.Size,
        stg.ListPrice,
        stg.Discontinued
FROM    dbo.StageProduct AS stg
UNION ALL
-- Existing rows
SELECT  dim.ProductBusinessKey,
        dim.ProductName,
        dim.ProductCategory,
        dim.Color,
        dim.Size,
        dim.ListPrice,
```

```
        dim.Discontinued
FROM    dbo.DimProduct AS dim
WHERE NOT EXISTS
(   SELECT  *
    FROM dbo.StageProduct AS stg
    WHERE stg.ProductId = dim.ProductBusinessKey
);

RENAME OBJECT dbo.DimProduct TO DimProductArchive;
RENAME OBJECT dbo.DimProductUpsert TO DimProduct;
```

Type 3 (Add a new Column/Attribute):

A Type 3 SCD supports storing two versions of a dimension member as separate columns.

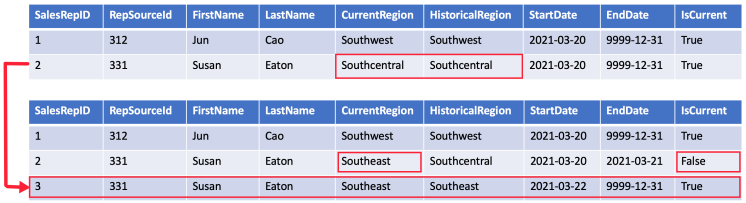
Here instead of having multiple rows to signify changes, we have multiple columns. We do have an effective/modified date column to show when the change took place.

CustomerID	FirstName	LastName	CurrentEmail	OriginalEmail	CompanyName	InsertedDate	ModifiedDate
2	Keith	Harris	keith0@aw.com	keith0@aw.com	Progressive Sports	2021-03-20	2021-03-20
3	Donna	Carreras	donna0@aw.com	donna0@aw.com	A Bike Store	2021-03-20	2021-03-20

CustomerID	FirstName	LastName	CurrentEmail	OriginalEmail	CompanyName	InsertedDate	ModifiedDate
2	Keith	Harris	keith0@aw.com	keith0@aw.com	Progressive Sports	2021-03-20	2021-03-20
3	Donna	Carreras	dc3@aw.com	donna0@aw.com	A Bike Store	2021-03-20	2021-03-22

Type 6:

A Type 6 SCD combines Type 1, 2, and 3. In Type 6 design we also store the current value in all versions of that entity so you can easily report the current value or the historical value.



SalesRepID	RepSourceId	FirstName	LastName	CurrentRegion	HistoricalRegion	StartDate	EndDate	IsCurrent
1	312	Jun	Cao	Southwest	Southwest	2021-03-20	9999-12-31	True
2	331	Susan	Eaton	Southcentral	Southcentral	2021-03-20	9999-12-31	True

SalesRepID	RepSourceId	FirstName	LastName	CurrentRegion	HistoricalRegion	StartDate	EndDate	IsCurrent
1	312	Jun	Cao	Southwest	Southwest	2021-03-20	9999-12-31	True
2	331	Susan	Eaton	Southeast	Southcentral	2021-03-20	2021-03-21	False
3	331	Susan	Eaton	Southeast	Southeast	2021-03-22	9999-12-31	True

---

Section 7: Designing Your ETL

Compare ETL to ELT

ETL and ELT are methodologies for handling the flow of data from source systems to a data warehouse or data lake for analytical purposes.

ETL (Extract, Transform, Load):

1. **Extract**: Data is extracted from various source systems (such as databases, applications, or external data sources) in raw form, often with errors.

2. **Transform**: The data is transformed into a consistent and uniform format. This may involve cleaning, validating, and converting data types, as well as aggregating, filtering, and sorting data to make it suitable for the target data warehouse or data mart.
3. **Load**: The transformed data is loaded into the user access layer (such as a data warehouse or data mart) where it becomes available for business intelligence (BI) and analytics.

ETL requires significant upfront work, including business analysis, data modelling, and data structure design, to ensure that the data is ready for analytical use when it is loaded into the user access layer.

ELT (Extract, Load, Transform):

1. **Extract**: Data is extracted from source systems, just like in the ETL process.
2. **Load**: Instead of transforming the data immediately, it is loaded into a big data environment, such as Hadoop Distributed File System (HDFS) or cloud-based storage like Amazon S3, in its raw form. This is usually done for both structured and unstructured data.
3. **Transform**: When the data is needed for analytical purposes, it is transformed using the computing power of the big data environment. This approach allows for more flexible and scalable data processing, as **transformations can be performed on demand**.

ELT defers the data modelling and analysis until the data is required for analytical use, following a schema-on-read approach. This allows for more agile data handling and reduces the need for upfront data modelling and analysis.

In summary, ETL is more suitable for traditional data warehousing with predefined data structures and strict data quality requirements, while ELT is more appropriate for big data environments, data lakes, or data warehouse-data lake hybrids, where data can be stored in its raw form and transformed later when needed.

	ETL	ELT
Process Order	Data is Extracted, Transformed, and then Loaded into the target database or data warehouse.	Data is Extracted, Loaded into the target system, and then Transformed.
Data Transformation	Transformation occurs before loading, typically in an intermediate staging area.	Transformation occurs after loading, directly in the target system.
Performance	May be slower for large data volumes, as transformations are performed before loading.	May be faster for large data volumes, as raw data is loaded first and transformations are performed within the target system.
Hardware Requirement	Can be less demanding, as transformation is done before loading the data, which can be distributed across multiple servers.	Can be more demanding, especially for cloud-based data warehouses that are designed for heavy computation.
Data Storage	Only transformed data is stored in the target system, which can be more efficient in terms of storage.	Both raw and transformed data are often stored, which can require more storage space.
Flexibility	Less flexible for changes in transformation logic, as changes may require re-extracting and re-loading data.	More flexible for changes in transformation logic, as raw data is already loaded and can be re-transformed.
Complexity	Can be more complex, as transformation logic needs to be defined before loading data.	Can be less complex, as raw data is loaded first and transformations can be performed using SQL or other data manipulation languages.
Use Case	Suitable for smaller data volumes, or when transformations are complex and need to be carefully controlled.	Suitable for large data volumes, or when using modern cloud-based data warehouses that can handle complex transformations.

Design the Initial Load ETL

Next, we focus on two forms of ETL (Extract, Transform, Load) processes in data warehousing environments: initial ETL and incremental ETL.

Initial ETL:

This is a one-time process, typically performed right before the data warehouse goes live. The goal is to gather all the relevant data needed for business intelligence (BI) and analytics, transform it, and load it into the user access layer of the data

warehouse.

- **Relevance is key**: Only data that is essential or likely to be needed for BI and analytics should be included instead of importing all data from the source systems.
- Historical data may also be imported to provide a basis for trend analysis and other historical reporting.
- Initial ETL might be repeated in cases of data corruption or re-platforming, but this is not common.

Incremental/Delta ETL:

This process is used to keep the data warehouse up-to-date after the initial ETL is completed. It is done regularly to refresh and update the data warehouse with new, modified, or deleted data.

- Regularly updates the data warehouse, ensuring it remains up-to-date.
- Adds new data, modifies existing data, and handles deleted data without purging it.
- Makes the data warehouse non-volatile and static between ETL runs.

Four major incremental ETL patterns:

1. **Append pattern**: New information is added to the existing data warehouse content.
2. **In-place update**: Existing rows are updated with changes, rather than appending new data.
3. **Complete replacement**: A portion of the data warehouse is entirely overwritten, even if only a small part of the data has changed.
4. **Rolling append**: A fixed duration of history is maintained, with new updates appending data and removing the oldest equivalent data.

Modern incremental ETL primarily uses the append and in-place update patterns, which align well with dimensional data handling.

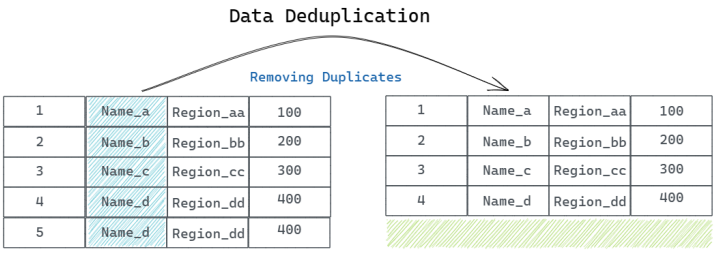
Why do we require Data Transformation?

Data transformation, a key aspect of ETL focuses on two main goals: uniformity and restructuring of data.

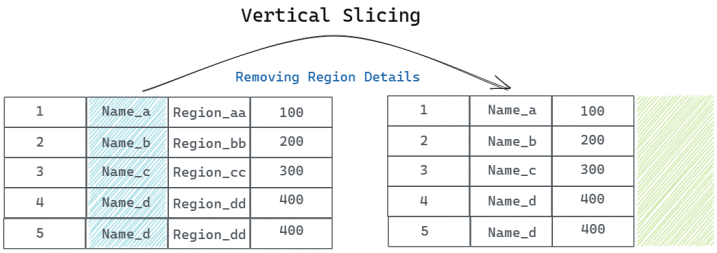
Uniformity ensures that data from different systems is transformed to allow for apples-to-apples comparisons. Restructuring involves organizing raw data from the staging layer into well-engineered data structures.

Some of the data transformation models are explained:

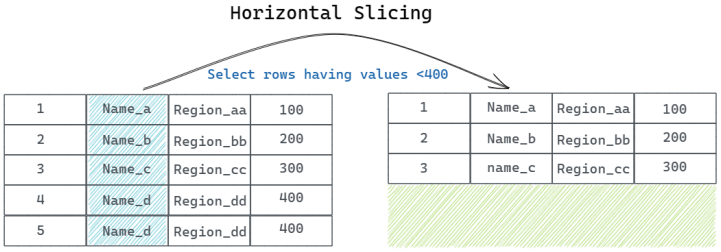
1. **Data value unification**: This involves unifying data values from different systems to present a consistent format to users. E.g. Daily sales data from two different regions being transformed to use a uniform abbreviation format in the data warehouse.
2. **Data type and size unification**: This involves unifying data types and sizes from different systems into a single representation in the data warehouse. E.g. Salesperson names from different regions have different character lengths, which need to be standardized in the data warehouse.
3. **Data deduplication**: This involves identifying and removing duplicate data to ensure accuracy in analytics and reporting. E.g. A particular salesperson's sales data is duplicated across different systems. The data warehouse should store only one copy of the information to avoid double counting.



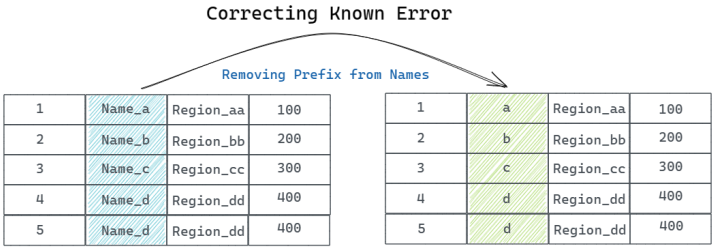
4. **Dropping columns (vertical slicing)**: This involves removing unnecessary columns from the data warehouse. E.g. Region data from a particular source system may be redundant as it has already been captured by another source. Thus, unnecessary columns need to be removed to maintain data quality in the warehouse.



5. **Row filtering based on values (horizontal slicing)**: This model filters out rows based on certain values in specific fields. E.g. Data warehouse is being built to analyze only particular sales values thus any rows not satisfying the value data need to be dropped.



6. **Correcting known errors**: This model involves fixing errors in the source data during the ETL process. E.g. names of salespersons have an unnecessary prefix that needs to be removed.



Some of the more advanced transformations may include joining, splitting (by length/position or by delimiter), aggregating (sum, count, average), deriving new values etc

Implement Mix-and-Match Incremental ETL

Let's discuss the flexibility and customization of incremental ETL (Extract, Transform, Load) processes in data warehousing. Although ETL feeds might appear similar, they can vary in terms of frequency and patterns, depending on the data and requirements.

Different ETL feeds might update the data warehouse at different frequencies, such as daily, hourly, or weekly, depending on the data volatility and criticality. Furthermore, incremental ETL patterns, like append, complete replacement, and in-place updates, can also vary across feeds.

The frequency and patterns are not solely determined by the ETL feed but can also depend on the table level within the source systems. Each source system can have a set of tables with different frequencies and ETL patterns. Thus it's important to implement a mix-and-match approach to cater to the specific needs of each table and source system.

Section 8: Selecting and Using a Data Warehouse

ETL Tools

Extract, Transform, Load (ETL) tools are used in the process of extracting data from different source systems, transforming it into a standard format, and loading it into a destination system, typically a data warehouse or data lake. There are a variety of ETL tools available, which cater to different environments and needs. Some of these tools are:

1. **Enterprise ETL Tools:**
   - **Informatica PowerCenter:** This is a widely used ETL tool that supports all the steps of the ETL process and is known for its high performance, intuitive interface, and wide support for different data sources and targets.
   - **IBM InfoSphere DataStage:** It is a part of IBM's Information Platforms Solutions suite and also its InfoSphere Platform. It uses a graphical notation to construct data integration solutions.
   - **Oracle Data Integrator (ODI):** Oracle's ETL tool provides a fully unified solution for building, deploying, and managing real-time data-centric architectures in an SOA, BI, and data warehouse environment.
   - **SAP Business Objects Data Services (BODS):** This ETL tool by SAP provides comprehensive data integration, data quality, and data processing.
   - **Alteryx, Microsoft SSIS** etc.
2. **Open Source ETL Tools:**
   - **Apache NiFi:** Apache NiFi supports powerful and scalable directed graphs of data routing, transformation, and system mediation logic.
   - **Talend Open Studio:** It's a robust open-source ETL tool that supports a wide array of source and target systems. Talend also offers a commercial version with additional features.
   - **Pentaho Data Integration:** Also known as Kettle, this tool offers data integration and transformation, including ETL capabilities.
3. **Cloud-based ETL Tools:**
   - **AWS Glue:** This is a fully managed ETL service provided by Amazon that makes it easy to prepare and load data for analytics.
   - **Google Cloud Dataflow:** This is a cloud-based data processing service for both batch and real-time data streaming applications.
   - **Azure Data Factory:** This is a cloud-based data integration service provided by Microsoft that orchestrates and automates the movement and transformation of data.
4. **Python-based ETL Tools:** For those who prefer to write their own ETL scripts, there are Python libraries such as **Pandas** and **PySpark** that can be used to create custom ETL processes.

The choice of an ETL tool depends on various factors: such as the nature of the source and target systems, the complexity of the ETL process, the required performance, the available budget, and the skills of the available staff.

1. **Data Source and Target Compatibility:** The ETL tool should be compatible with your data sources (databases, APIs, file formats, etc.) and target systems (data warehouse, data lake, etc.).
2. **Performance:** The efficiency and speed of the ETL tool are important, especially if you need to process large volumes of data.
3. **Scalability:** The ETL tool should be able to scale and handle increasing data volumes as your business grows.
4. **Ease of Use:** A tool with a user-friendly interface can lower the learning curve and increase productivity. Some ETL tools provide a graphical interface to design ETL flows, which can be easier to use than writing code.
5. **Data Transformation Capabilities:** The ETL tool should support the types of data transformations you need, such as filtering, aggregation, joining, splitting, data type conversion, etc.
6. **Data Quality Features:** Some ETL tools provide built-in features to ensure data quality, such as data profiling, data validation, and error handling capabilities.
7. **Scheduling and Automation:** The ETL tool should provide capabilities to schedule ETL jobs and automate workflows.
8. **Real-time Processing:** If you need real-time or near real-time data integration, choose an ETL tool that supports streaming data and real-time processing.
9. **Security:** The ETL tool should provide strong security features, including data encryption, user authentication, access controls, and audit logs.

10. **Cost:** The cost of ETL tools can vary widely, from free open-source tools to expensive commercial solutions. Consider your budget and the total cost of ownership, including license costs, hardware costs, support costs, and training costs.

Optimizing Data Warehouse

A performance improvement strategy should be in place whenever a DWH is delivered. This ensures that DWH stays relevant and meets the business end-user requirements on time.

Indexing

Adding some search functionality or indexes can drastically improve your chances of finding your item quicker. Indexes store the value from the given column in a searchable structure, which allows the query to read fewer data to find the information. However, they have an overhead, and having too many indexes slows down the insert and update operations on your DWH. Bitmap indexes, B-tree indexes, and columnstore indexes are some of the types that can be used in a data warehouse.

1. **Bitmap Indexing**

   Bitmap indexing is a special kind of database indexing that uses bitmaps or bit arrays. It is particularly effective for queries on large tables that return a small percentage of rows, and it's very useful for low-cardinality fields, i.e., fields that have a small number of distinct values.

   For example, let's consider a table `Customer` in a database, where we have a column `Gender` with two distinct values: Male and Female.

   ```
   Customer Table:
   +----+------+--------+
   | ID | Name | Gender |
   +----+------+--------+
   | 1  | Alex | Male   |
   | 2  | Sam  | Female |
   | 3  | John | Male   |
   | 4  | Anna | Female |
   | 5  | Mike | Male   |
   +----+------+--------+
   ```

   A bitmap index on the `Gender` column would look something like this:

   ```
   Bitmap Index:
   +--------+---------+
   | Gender | Bitmap  |
   +--------+---------+
   | Male   | 10101   |
   | Female | 01010   |
   +--------+---------+
   ```

   In this bitmap representation, each bit position corresponds to a row in the table. A `1` indicates that the row has that value for the indexed column, and a `0` indicates that it does not.

   For example, the bitmap for `Male` is `10101`. This means that rows 1, 3, and 5 in the table have the `Gender` value `Male`. Similarly, the bitmap for `Female` is `01010`, which corresponds to rows 2 and 4.

   The advantage of a bitmap index is that it can be very space-efficient for low-cardinality data, and it allows for very fast Boolean operations (AND, OR, NOT) between different bitmaps. For instance, if you want to find all customers who are Male, the database engine can quickly find this information from the bitmap index without scanning the whole table. However, it performs poorly when it comes to high-cardinality data, where the number of distinct values is high, and the bitmaps become sparse and take up more space.

2. **B-tree Indexing**

   B-tree indexing is a commonly used indexing method in databases. B-tree stands for "balanced tree", and it's a sorted data structure that maintains sorted data and allows for efficient insertion, deletion, and search operations.

   B-tree indexes are beneficial when dealing with large amounts of data, as they keep data sorted and allow searches, sequential access, insertions, and deletions in logarithmic time. They are particularly useful for databases stored on disk, as they minimize disk I/O operations.

   Let's consider a simplified example of a B-tree index on an `Employee` table.

```
Employee Table:
+----+-------+
| ID | Name  |
+----+-------+
| 1  | Alex  |
| 2  | Bob   |
| 3  | Carol |
| 4  | Dave  |
| 5  | Ed    |
+----+-------+
```

We might create a B-tree index on the `ID` column. The B-tree index structure might look something like this:

```
           [3]
       /   \
    [2]     [5]
   /   \   /   \
 [1] [3] [4]  [5]
```

In this tree, each node represents a page or block of the index, and each page contains one or more keys and pointers. The keys act as separation values which divide its subtrees.

For instance, in the root node of the tree above, the key `3` separates two subtrees. The left subtree contains values less than `3`, and the right subtree contains values greater than `3`.

If we wanted to find the data associated with `ID = 4`, we would:

1. Start at the root and see the value `3`.
2. `4` is greater than `3`, so we follow the right pointer.
3. In the right child node, we find `5`. Since `4` is less than `5`, we follow the left pointer.
4. We reach the leaf node with the key `4`, where we can find the pointer to the actual data record in the table.

This efficient structure of the B-tree index allows the database to quickly find data without scanning the whole table. It's important to note that in actual databases, B-trees can have many more child nodes per parent, and the tree can be much deeper, allowing for efficient indexing of large amounts of data.

3. **Columnstore Indexing**

   Columnstore indexing is a technology used to enhance the processing speed of database queries and is especially efficient for large data warehouse queries. The key difference between a columnstore index and traditional row-based indexes (like B-tree) is that data is stored column-wise rather than row-wise. This columnar storage allows for high compression rates and significantly improves query performance.

   Columnstore indexes work best on fact tables in the data warehouse schema that are loaded through a single ETL process and used for read-only queries. Let's consider an example:

   Imagine we have a Sales table with millions of rows and the following structure:

```
Sales Table:
+--------+-------+----------+--------+
| SaleID | Item  | Quantity | Price  |
+--------+-------+----------+--------+
| 1      | Apple | 10       | 2.00   |
| 2      | Pear  | 15       | 2.50   |
| 3      | Grape | 20       | 3.00   |
| ...    | ...   | ...      | ...    |
+--------+-------+----------+--------+
```

   If we create a columnstore index on this table, the data would be stored something like this:

```
SaleID Column:     | 1 | 2 | 3 | ...
Item Column:       | Apple | Pear | Grape | ...
```

```
Quantity Column:   | 10 | 15 | 20 | ...
Price Column:      | 2.00 | 2.50 | 3.00 | ...
```

Each column is stored separately, which enables high compression rates because the redundancy within a column is typically higher than within a row. This structure also improves the performance of queries that only need a few columns from the table because only the columns needed for the query are fetched from storage.

For example, if we wanted to calculate the total revenue from all sales, we would need only the `Quantity` and `Price` columns. A database with a columnstore index could perform this operation much faster than traditional row-based storage because it only needs to read two columns instead of the entire table.

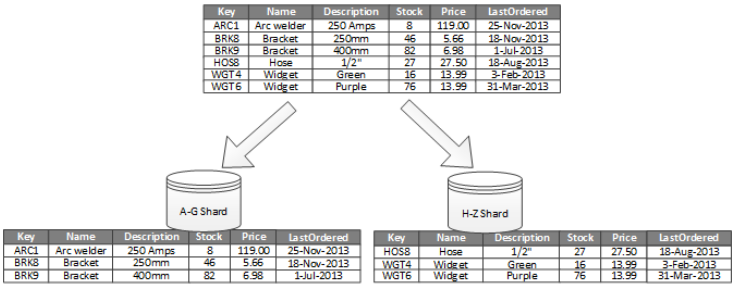
It's also worth mentioning that columnstore indexes allow for batch processing, which is another aspect that enhances their performance compared to traditional B-tree indexes.

Partitioning

In many large-scale solutions, data is divided into *partitions* that can be managed and accessed separately. Partitioning can improve scalability, reduce contention, and optimize performance. It can also provide a mechanism for dividing data by usage pattern. There are three typical strategies for partitioning data:

1. **Horizontal partitioning** (often called *sharding*)

   In this strategy, each partition is a separate data store, but all partitions have the same schema. Each partition is known as a *shard* and holds a specific subset of the data, such as all the orders for a specific set of customers.



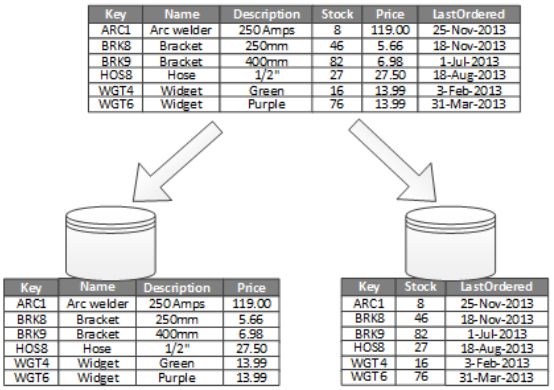
Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013

Key	Name	Description	Stock	Price	LastOrdered
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

2. **Vertical partitioning**

   In this strategy, each partition holds a subset of the fields for items in the data store. The fields are divided according to their pattern of use. For example, frequently accessed fields might be placed in one vertical partition and less frequently accessed fields in another.



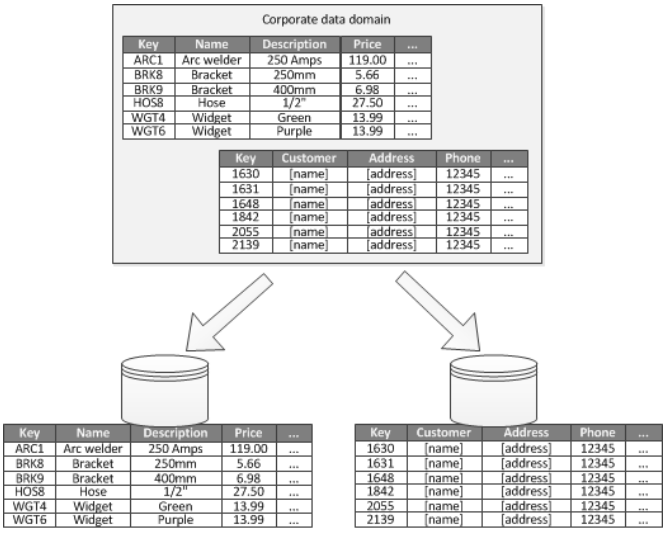
Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Key	Name	Description	Price
ARC1	Arc welder	250 Amps	119.00
BRK8	Bracket	250mm	5.66
BRK9	Bracket	400mm	6.98
HOS8	Hose	1/2"	27.50
WGT4	Widget	Green	13.99
WGT6	Widget	Purple	13.99

Key	Stock	LastOrdered
ARC1	8	25-Nov-2013
BRK8	46	18-Nov-2013
BRK9	82	1-Jul-2013
HOS8	27	18-Aug-2013
WGT4	16	3-Feb-2013
WGT6	76	31-Mar-2013

3. **Functional partitioning**

   In this strategy, data is aggregated according to how it is used by each bounded context in the system. For example, an e-commerce system might store invoice data in one partition and product inventory data in another.



Materialized Views:

Materialized views are a type of view in databases but with a twist. While a standard view is a virtual table that dynamically retrieves data from the underlying tables, a materialized view stores the result of the query physically, similar to a table. Because of this, they can significantly speed up query execution times as the database does not need to compute the result set every time the view is queried - it simply accesses the pre-computed results.

Let's consider an example where a materialized view can be beneficial. Suppose we have two tables, `Orders` and `Customers`, in a retail database.

The `Orders` table might look something like this:

```
Orders:
+---------+------------+-------+
| OrderID | CustomerID | Total |
+---------+------------+-------+
| 1       | 101        | 50.00 |
| 2       | 102        | 75.00 |
| 3       | 103        | 25.00 |
| ...     | ...        | ...   |
+---------+------------+-------+
```

And the `Customers` table might look like this:

```
Customers:
+------------+-------+
| CustomerID | Name  |
+------------+-------+
| 101        | Alice |
| 102        | Bob   |
| 103        | Carol |
| ...        | ...   |
+------------+-------+
```

Now, suppose we frequently need to calculate the total amount spent by each customer. We could create a view that aggregates this data:

```
CREATE VIEW TotalSpent AS
SELECT Customers.Name, SUM(Orders.Total) as TotalSpent
FROM Customers
JOIN Orders ON Customers.CustomerID = Orders.CustomerID
GROUP BY Customers.Name;
```

Each time we query this view, the database would need to perform the join and aggregation operation, which can be costly on large tables.

However, if we were to create a materialized view:

```
CREATE MATERIALIZED VIEW TotalSpent AS
SELECT Customers.Name, SUM(Orders.Total) as TotalSpent
FROM Customers
JOIN Orders ON Customers.CustomerID = Orders.CustomerID
GROUP BY Customers.Name;
```

The database would store the result of the query in a table-like structure. When we query the `TotalSpent` view, the database would simply return the pre-computed results, which can be significantly faster.

However, there is a tradeoff. The data in a materialized view can become stale when the underlying data changes. Depending on the database system, you may need to manually refresh the materialized view, or it might be possible to set it to refresh automatically at certain intervals or in response to certain events.

Compression

Compression reduces the amount of storage space needed and can also improve query performance, as less data needs to be read from the disk. Many modern DBMSs offer data compression features.

Parallel Processing

Many data warehouse systems support parallel processing, which can greatly improve the performance of large queries. This involves dividing a task into smaller subtasks that are executed concurrently.

DATA WAREHOUSING - QUICK GUIDE

DATA WAREHOUSING - OVERVIEW

The term "Data Warehouse" was first coined by Bill Inmon in 1990. According to Inmon, a data warehouse is a subject oriented, integrated, time-variant, and non-volatile collection of data. This data helps analysts to take informed decisions in an organization.

An operational database undergoes frequent changes on a daily basis on account of the transactions that take place. Suppose a business executive wants to analyze previous feedback on any data such as a product, a supplier, or any consumer data, then the executive will have no data available to analyze because the previous data has been updated due to transactions.

A data warehouses provides us generalized and consolidated data in multidimensional view. Along with generalized and consolidated view of data, a data warehouses also provides us Online Analytical Processing *OLAP* tools. These tools help us in interactive and effective analysis of data in a multidimensional space. This analysis results in data generalization and data mining.

Data mining functions such as association, clustering, classification, prediction can be integrated with OLAP operations to enhance the interactive mining of knowledge at multiple level of abstraction. That's why data warehouse has now become an important platform for data analysis and online analytical processing.

Understanding a Data Warehouse

- A data warehouse is a database, which is kept separate from the organization's operational database.
- There is no frequent updating done in a data warehouse.
- It possesses consolidated historical data, which helps the organization to analyze its business.
- A data warehouse helps executives to organize, understand, and use their data to take strategic decisions.
- Data warehouse systems help in the integration of diversity of application systems.
- A data warehouse system helps in consolidated historical data analysis.

Why a Data Warehouse is Separated from Operational Databases

A data warehouses is kept separate from operational databases due to the following reasons:

- An operational database is constructed for well-known tasks and workloads such as searching particular records, indexing, etc. In contract, data warehouse queries are often complex and they present a general form of data.
- Operational databases support concurrent processing of multiple transactions. Concurrency control and recovery mechanisms are required for operational databases to ensure robustness and consistency of the database.
- An operational database query allows to read and modify operations, while an OLAP query needs only **read only** access of stored data.
- An operational database maintains current data. On the other hand, a data warehouse maintains historical data.

Data Warehouse Features

The key features of a data warehouse are discussed below:

- **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.

Note: A data warehouse does not require transaction processing, recovery, and concurrency controls, because it is physically stored and separate from the operational database.

Data Warehouse Applications

As discussed before, a data warehouse helps business executives to organize, analyze, and use their data for decision making. A data warehouse serves as a sole part of a plan-execute-assess "closed-loop" feedback system for the enterprise management. Data warehouses are widely used in the following fields:

- Financial services
- Banking services
- Consumer goods
- Retail sectors
- Controlled manufacturing

Types of Data Warehouse

Information processing, analytical processing, and data mining are the three types of data warehouse applications that are discussed below:

- **Information Processing** - A data warehouse allows to process the data stored in it. The data can be processed by means of querying, basic statistical analysis, reporting using crosstabs, tables, charts, or graphs.
- **Analytical Processing** - A data warehouse supports analytical processing of the information stored in it. The data can be analyzed by means of basic OLAP operations, including slice-and-dice, drill down, drill up, and pivoting.
- **Data Mining** - Data mining supports knowledge discovery by finding hidden patterns and associations, constructing analytical models, performing classification and prediction. These mining results can be presented using the visualization tools.

Sr.No.	Data Warehouse *OLAP*	Operational Database*OLTP*
1	It involves historical processing of information.	It 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	It is used to analyze the business.	It is used to run the business.
4	It focuses on Information out.	It focuses on Data in.
5	It is based on Star Schema, Snowflake Schema, and Fact Constellation Schema.	It is based on Entity Relationship Model.

6	It focuses on Information out.	It is application oriented.
7	It contains historical data.	It contains current data.
8	It provides summarized and consolidated data.	It provides primitive and highly detailed data.
9	It provides summarized and multidimensional view of data.	It provides detailed and flat relational view of data.
10	The number of users is in hundreds.	The number of users is in thousands.
11	The number of records accessed is in millions.	The number of records accessed is in tens.
12	The database size is from 100GB to 100 TB.	The database size is from 100 MB to 100 GB.
13	These are highly flexible.	It provides high performance.

DATA WAREHOUSING - CONCEPTS

What is Data Warehousing?

Data warehousing is the process of constructing and using a data warehouse. A data warehouse is constructed by integrating data from multiple heterogeneous sources that support analytical reporting, structured and/or ad hoc queries, and decision making. Data warehousing involves data cleaning, data integration, and data consolidations.

Using Data Warehouse Information

There are decision support technologies that help utilize the data available in a data warehouse. These technologies help executives to use the warehouse quickly and effectively. They can gather data, analyze it, and take decisions based on the information present in the warehouse. The information gathered in a warehouse can be used in any of the following domains:

- **Tuning Production Strategies** - The product strategies can be well tuned by repositioning the products and managing the product portfolios by comparing the sales quarterly or yearly.
- **Customer Analysis** - Customer analysis is done by analyzing the customer's buying preferences, buying time, budget cycles, etc.
- **Operations Analysis** - Data warehousing also helps in customer relationship management, and making environmental corrections. The information also allows us to analyze business operations.

Integrating Heterogeneous Databases

To integrate heterogeneous databases, we have two approaches:

- Query-driven Approach
- Update-driven Approach

Query-Driven Approach

This is the traditional approach to integrate heterogeneous databases. This approach was used to build wrappers and integrators on top of multiple heterogeneous databases. These integrators are also known as mediators.

Process of Query-Driven Approach

- When a query is issued to a client side, a metadata dictionary translates the query into an

appropriate from for individual heterogeneous sites involved.

- Now these queries are mapped and sent to the local query processor.
- The results from heterogeneous sites are integrated into a global answer set.

Disadvantages

- Query-driven approach needs complex integration and filtering processes.
- This approach is very inefficient.
- It is very expensive for frequent queries.
- This approach is also very expensive for queries that require aggregations.

Update-Driven Approach

This is an alternative to the traditional approach. Today's data warehouse systems follow update-driven approach rather than the traditional approach discussed earlier. In update-driven approach, the information from multiple heterogeneous sources are integrated in advance and are stored in a warehouse. This information is available for direct querying and analysis.

Advantages

This approach has the following advantages:

- This approach provide high performance.
- The data is copied, processed, integrated, annotated, summarized and restructured in semantic data store in advance.
- Query processing does not require an interface to process data at local sources.

Functions of Data Warehouse Tools and Utilities

The following are the functions 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.

Note: Data cleaning and data transformation are important steps in improving the quality of data and data mining results.

DATA WAREHOUSING - TERMINOLOGIES

In this chapter, we will discuss some of the most commonly used terms in data warehousing.

Metadata

Metadata is simply defined as data about data. The data that are used to represent other data is known as metadata. For example, the index of a book serves as a metadata for the contents in the book. In other words, we can say that metadata is the summarized data that leads us to the detailed data.

In terms of data warehouse, we can define metadata as following:

- Metadata is a road-map to data warehouse.
- Metadata in data warehouse defines the warehouse objects.
- Metadata acts as a directory. This directory helps the decision support system to locate the contents of a data warehouse.

Metadata Repository

Metadata repository is an integral part of a data warehouse system. It contains the following metadata:

- **Business metadata** - It contains the data ownership information, business definition, and changing policies.
- **Operational metadata** - It includes currency of data and data lineage. Currency of data refers to the data being active, archived, or purged. Lineage of data means history of data migrated and transformation applied on it.
- **Data for mapping from operational environment to data warehouse** - It metadata includes source databases and their contents, data extraction, data partition, cleaning, transformation rules, data refresh and purging rules.
- **The algorithms for summarization** - It includes dimension algorithms, data on granularity, aggregation, summarizing, etc.

Data Cube

A data cube helps us represent data in multiple dimensions. It is defined by dimensions and facts. The dimensions are the entities with respect to which an enterprise preserves the records.

Illustration of Data Cube

Suppose a company wants to keep track of sales records with the help of sales data warehouse with respect to time, item, branch, and location. These dimensions allow to keep track of monthly sales and at which branch the items were sold. There is a table associated with each dimension. This table is known as dimension table. For example, "item" dimension table may have attributes such as item_name, item_type, and item_brand.

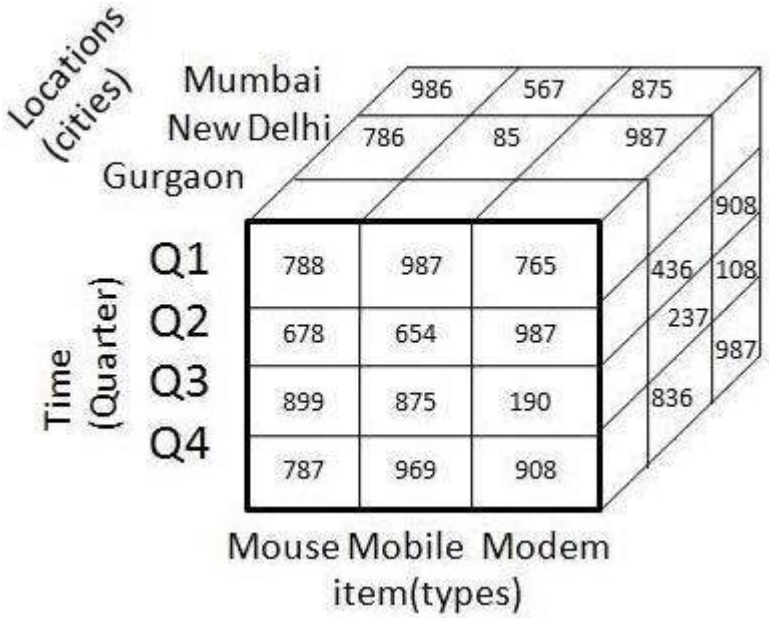
The following table represents the 2-D view of Sales Data for a company with respect to time, item, and location dimensions.

Location="New Delhi"				
Time(quarter)	Item(type)			
	Entertainment	Keyboard	Mobile	Locks
Q1	500	700	10	300
Q2	769	765	30	476
Q3	987	489	18	659
Q4	666	976	40	539

But here in this 2-D table, we have records with respect to time and item only. The sales for New Delhi are shown with respect to time, and item dimensions according to type of items sold. If we want to view the sales data with one more dimension, say, the location dimension, then the 3-D view would be useful. The 3-D view of the sales data with respect to time, item, and location is shown in the table below:

Time	Location="Gurgaon"			Location="New Delhi"			Location="Mumbai"		
	Item			Item			Item		
	Mouse	Mobile	Modem	Mouse	Mobile	Modem	Mouse	Mobile	Modem
Q1	788	987	765	786	85	987	986	567	875
Q2	678	654	987	659	786	436	980	876	908
Q3	899	875	190	983	909	237	987	100	1089
Q4	787	969	908	537	567	836	837	926	987

The above 3-D table can be represented as 3-D data cube as shown in the following figure:



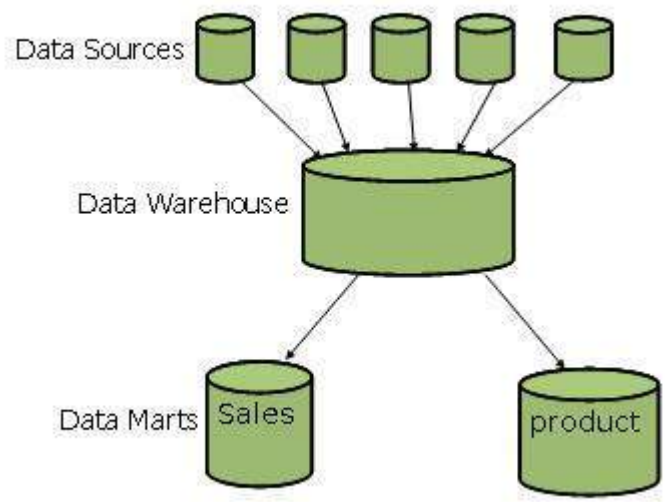
Data Mart

Data marts contain a subset of organization-wide data that is valuable to specific groups of people in an organization. In other words, a data mart contains only those data that is specific to a particular group. For example, the marketing data mart may contain only data related to items, customers, and sales. Data marts are confined to subjects.

Points to Remember About Data Marts

- Windows-based or Unix/Linux-based servers are used to implement data marts. They are implemented on low-cost servers.
- The implementation cycle of a data mart is measured in short periods of time, i.e., in weeks rather than months or years.
- The life cycle of data marts may be complex in the long run, if their planning and design are not organization-wide.
- Data marts are small in size.
- Data marts are customized by department.
- The source of a data mart is departmentally structured data warehouse.
- Data marts are flexible.

The following figure shows a graphical representation of data marts.



Virtual Warehouse

The view over an operational data warehouse is known as virtual warehouse. It is easy to build a virtual warehouse. Building a virtual warehouse requires excess capacity on operational database servers.

DATA WAREHOUSING - DELIVERY PROCESS

A data warehouse is never static; it evolves as the business expands. As the business evolves, its requirements keep changing and therefore a data warehouse must be designed to ride with these changes. Hence a data warehouse system needs to be flexible.

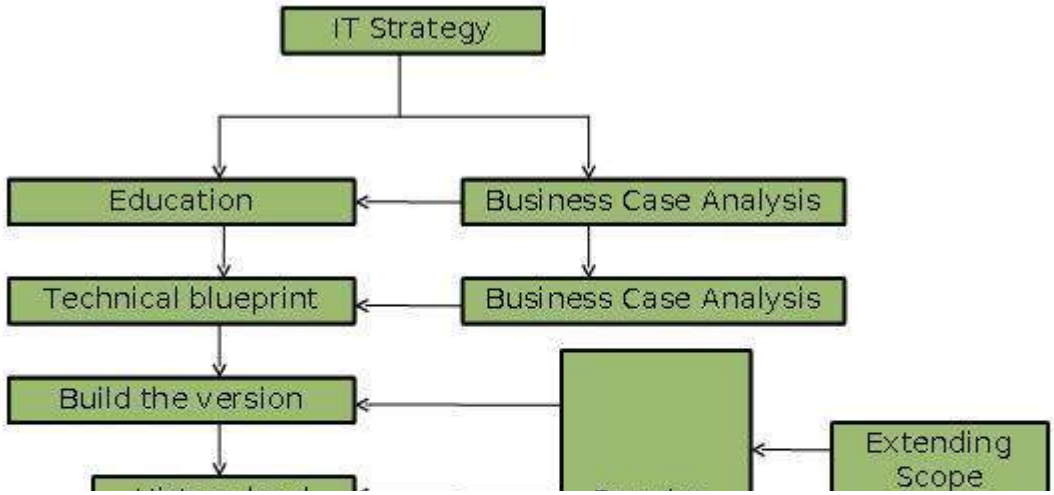
Ideally there should be a delivery process to deliver a data warehouse. However data warehouse projects normally suffer from various issues that make it difficult to complete tasks and deliverables in the strict and ordered fashion demanded by the waterfall method. Most of the times, the requirements are not understood completely. The architectures, designs, and build components can be completed only after gathering and studying all the requirements.

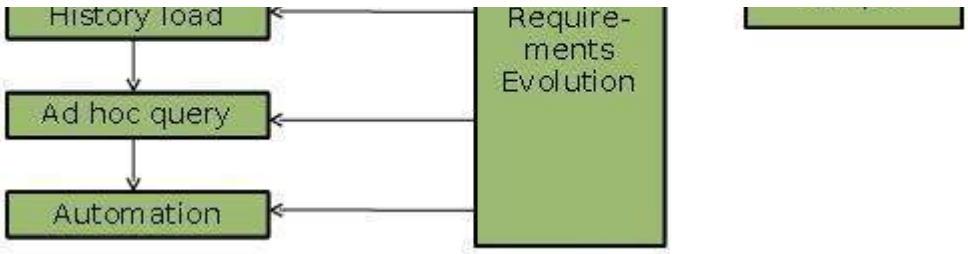
Delivery Method

The delivery method is a variant of the joint application development approach adopted for the delivery of a data warehouse. We have staged the data warehouse delivery process to minimize risks. The approach that we will discuss here does not reduce the overall delivery time-scales but ensures the business benefits are delivered incrementally through the development process.

Note: The delivery process is broken into phases to reduce the project and delivery risk.

The following diagram explains the stages in the delivery process:





IT Strategy

Data warehouse are strategic investments that require a business process to generate benefits. IT Strategy is required to procure and retain funding for the project.

Business Case

The objective of business case is to estimate business benefits that should be derived from using a data warehouse. These benefits may not be quantifiable but the projected benefits need to be clearly stated. If a data warehouse does not have a clear business case, then the business tends to suffer from credibility problems at some stage during the delivery process. Therefore in data warehouse projects, we need to understand the business case for investment.

Education and Prototyping

Organizations experiment with the concept of data analysis and educate themselves on the value of having a data warehouse before settling for a solution. This is addressed by prototyping. It helps in understanding the feasibility and benefits of a data warehouse. The prototyping activity on a small scale can promote educational process as long as:

- The prototype addresses a defined technical objective.
- The prototype can be thrown away after the feasibility concept has been shown.
- The activity addresses a small subset of eventual data content of the data warehouse.
- The activity timescale is non-critical.

The following points are to be kept in mind to produce an early release and deliver business benefits.

- Identify the architecture that is capable of evolving.
- Focus on business requirements and technical blueprint phases.
- Limit the scope of the first build phase to the minimum that delivers business benefits.
- Understand the short-term and medium-term requirements of the data warehouse.

Business Requirements

To provide quality deliverables, we should make sure the overall requirements are understood. If we understand the business requirements for both short-term and medium-term, then we can design a solution to fulfil short-term requirements. The short-term solution can then be grown to a full solution.

The following aspects are determined in this stage:

Things to determine in this stage are following.

- The business rule to be applied on data.
- The logical model for information within the data warehouse.
- The query profiles for the immediate requirement.
- The source systems that provide this data.

Technical Blueprint

This phase need to deliver an overall architecture satisfying the long term requirements. This phase also deliver the components that must be implemented in a short term to derive any business benefit. The blueprint need to identify the followings.

- The overall system architecture.
- The data retention policy.
- The backup and recovery strategy.
- The server and data mart architecture.
- The capacity plan for hardware and infrastructure.
- The components of database design.

Building the version

In this stage, the first production deliverable is produced. This production deliverable is the smallest component of a data warehouse. This smallest component adds business benefit.

History Load

This is the phase where the remainder of the required history is loaded into the data warehouse. In this phase, we do not add new entities, but additional physical tables would probably be created to store increased data volumes.

Let us take an example. Suppose the build version phase has delivered a retail sales analysis data warehouse with 2 months' worth of history. This information will allow the user to analyze only the recent trends and address the short-term issues. The user in this case cannot identify annual and seasonal trends. To help him do so, last 2 years' sales history could be loaded from the archive. Now the 40GB data is extended to 400GB.

Note: The backup and recovery procedures may become complex, therefore it is recommended to perform this activity within a separate phase.

Ad hoc Query

In this phase, we configure an ad hoc query tool that is used to operate a data warehouse. These tools can generate the database query.

Note: It is recommended not to use these access tools when the database is being substantially modified.

Automation

In this phase, operational management processes are fully automated. These would include:

- Transforming the data into a form suitable for analysis.
- Monitoring query profiles and determining appropriate aggregations to maintain system performance.
- Extracting and loading data from different source systems.
- Generating aggregations from predefined definitions within the data warehouse.
- Backing up, restoring, and archiving the data.

Extending Scope

In this phase, the data warehouse is extended to address a new set of business requirements. The scope can be extended in two ways:

- By loading additional data into the data warehouse.

- By introducing new data marts using the existing information.

Note: This phase should be performed separately, since it involves substantial efforts and complexity.

Requirements Evolution

From the perspective of delivery process, the requirements are always changeable. They are not static. The delivery process must support this and allow these changes to be reflected within the system.

This issue is addressed by designing the data warehouse around the use of data within business processes, as opposed to the data requirements of existing queries.

The architecture is designed to change and grow to match the business needs, the process operates as a pseudo-application development process, where the new requirements are continually fed into the development activities and the partial deliverables are produced. These partial deliverables are fed back to the users and then reworked ensuring that the overall system is continually updated to meet the business needs.

DATA WAREHOUSING - SYSTEM PROCESSES

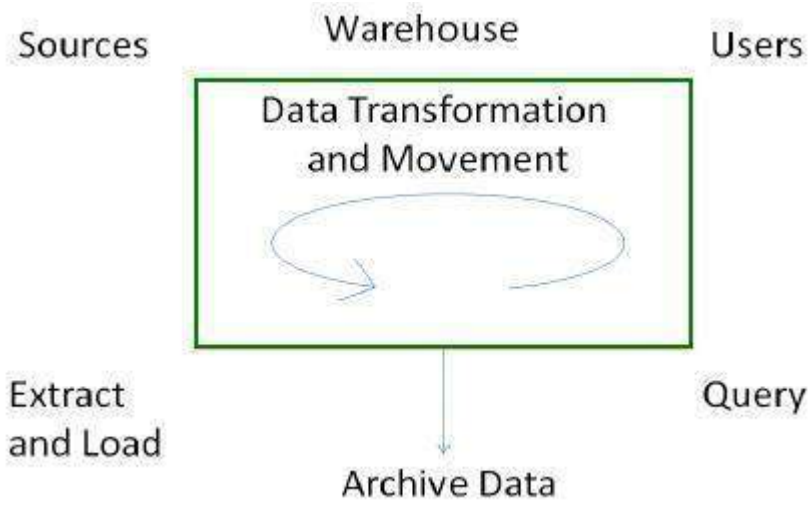
We have a fixed number of operations to be applied on the operational databases and we have well-defined techniques such as **use normalized data**, **keep table small**, etc. These techniques are suitable for delivering a solution. But in case of decision-support systems, we do not know what query and operation needs to be executed in future. Therefore techniques applied on operational databases are not suitable for data warehouses.

In this chapter, we will discuss how to build data warehousing solutions on top open-system technologies like Unix and relational databases.

Process Flow in Data Warehouse

There are four major processes that contribute to a data warehouse:

- Extract and load the data.
- Cleaning and transforming the data.
- Backup and archive the data.
- Managing queries and directing them to the appropriate data sources.



Extract and Load Process

Data extraction takes data from the source systems. Data load takes the extracted data and loads it into the data warehouse.

Note: Before loading the data into the data warehouse, the information extracted from the

external sources must be reconstructed.

Controlling the Process

Controlling the process involves determining when to start data extraction and the consistency check on data. Controlling process ensures that the tools, the logic modules, and the programs are executed in correct sequence and at correct time.

When to Initiate Extract

Data needs to be in a consistent state when it is extracted, i.e., the data warehouse should represent a single, consistent version of the information to the user.

For example, in a customer profiling data warehouse in telecommunication sector, it is illogical to merge the list of customers at 8 pm on Wednesday from a customer database with the customer subscription events up to 8 pm on Tuesday. This would mean that we are finding the customers for whom there are no associated subscriptions.

Loading the Data

After extracting the data, it is loaded into a temporary data store where it is cleaned up and made consistent.

Note: Consistency checks are executed only when all the data sources have been loaded into the temporary data store.

Clean and Transform Process

Once the data is extracted and loaded into the temporary data store, it is time to perform Cleaning and Transforming. Here is the list of steps involved in Cleaning and Transforming:

- Clean and transform the loaded data into a structure
- Partition the data
- Aggregation

Clean and Transform the Loaded Data into a Structure

Cleaning and transforming the loaded data helps speed up the queries. It can be done by making the data consistent:

- within itself.
- with other data within the same data source.
- with the data in other source systems.
- with the existing data present in the warehouse.

Transforming involves converting the source data into a structure. Structuring the data increases the query performance and decreases the operational cost. The data contained in a data warehouse must be transformed to support performance requirements and control the ongoing operational costs.

Partition the Data

It will optimize the hardware performance and simplify the management of data warehouse. Here we partition each fact table into multiple separate partitions.

Aggregation

Aggregation is required to speed up common queries. Aggregation relies on the fact that most common queries will analyze a subset or an aggregation of the detailed data.

Backup and Archive the Data

In order to recover the data in the event of data loss, software failure, or hardware failure, it is necessary to keep regular back ups. Archiving involves removing the old data from the system in a format that allow it to be quickly restored whenever required.

For example, in a retail sales analysis data warehouse, it may be required to keep data for 3 years with the latest 6 months data being kept online. In such as scenario, there is often a requirement to be able to do month-on-month comparisons for this year and last year. In this case, we require some data to be restored from the archive.

Query Management Process

This process performs the following functions:

- manages the queries.
- helps speed up the execution time of queris.
- directs the queries to their most effective data sources.
- ensures that all the system sources are used in the most effective way.
- monitors actual query profiles.

The information generated in this process is used by the warehouse management process to determine which aggregations to generate. This process does not generally operate during the regular load of information into data warehouse.

DATA WAREHOUSING - ARCHITECTURE

In this chapter, we will discuss the business analysis framework for the data warehouse design and architecture of a data warehouse.

Business Analysis Framework

The business analyst get the information from the data warehouses to measure the performance and make critical adjustments in order to win over other business holders in the market. Having a data warehouse offers the following advantages:

- Since a data warehouse can gather information quickly and efficiently, it can enhance business productivity.
- A data warehouse provides us a consistent view of customers and items, hence, it helps us manage customer relationship.
- A data warehouse also helps in bringing down the costs by tracking trends, patterns over a long period in a consistent and reliable manner.

To design an effective and efficient data warehouse, we need to understand and analyze the business needs and construct a **business analysis framework**. Each person has different views regarding the design of a data warehouse. These views are as follows:

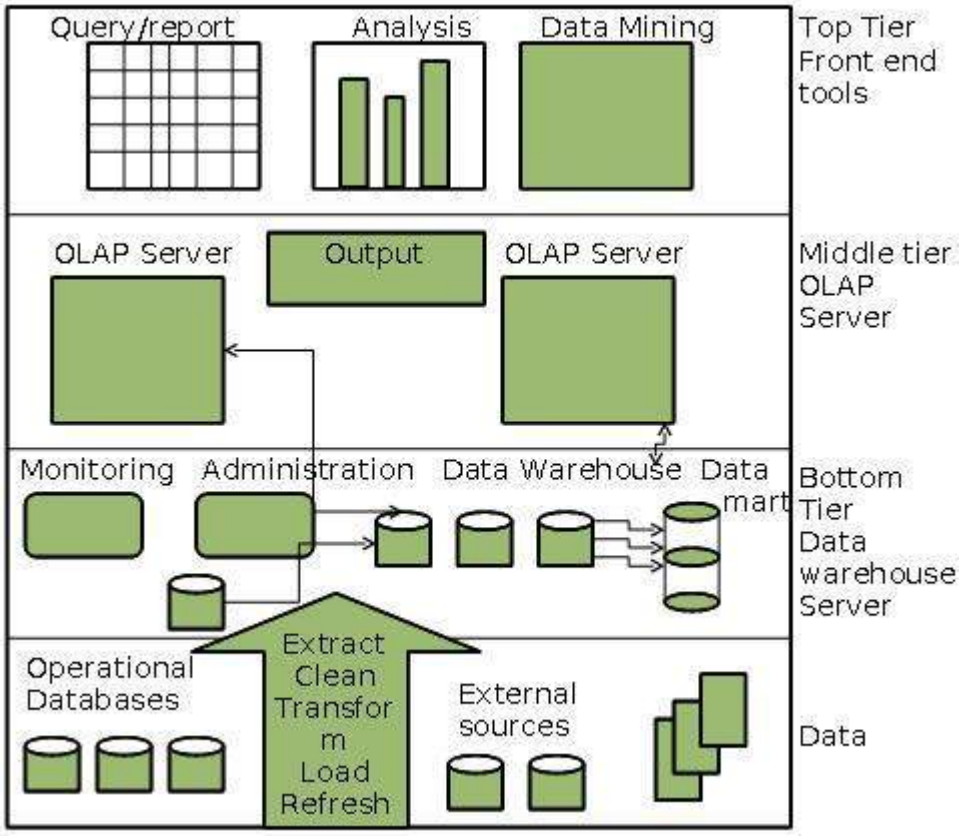
- **The top-down view** - This view allows the selection of relevant information needed for a data warehouse.
- **The data source view** - This view presents the information being captured, stored, and managed by the operational system.
- **The data warehouse view** - This view includes the fact tables and dimension tables. It represents the information stored inside the data warehouse.
- **The business query view** - It is the view of the data from the viewpoint of the end-user.

Three-Tier Data Warehouse Architecture

Generally a data warehouses adopts a three-tier architecture. Following are the three tiers of the data warehouse architecture.

- **Bottom Tier** - The bottom tier of the architecture is the data warehouse database server. It is the relational database system. We use the back end tools and utilities to feed data into the bottom tier. These back end tools and utilities perform the Extract, Clean, Load, and refresh functions.
- **Middle Tier** - In the middle tier, we have the OLAP Server that can be implemented in either of the following ways.
    - By Relational OLAP *ROLAP*, which is an extended relational database management system. The ROLAP maps the operations on multidimensional data to standard relational operations.
    - By Multidimensional OLAP *MOLAP* model, which directly implements the multidimensional data and operations.
- **Top-Tier** - This tier is the front-end client layer. This layer holds the query tools and reporting tools, analysis tools and data mining tools.

The following diagram depicts the three-tier architecture of data warehouse:



Data Warehouse Models

From the perspective of data warehouse architecture, we have the following data warehouse models:

- Virtual Warehouse
- Data mart
- Enterprise Warehouse

Virtual Warehouse

The view over an operational data warehouse is known as a virtual warehouse. It is easy to build a virtual warehouse. Building a virtual warehouse requires excess capacity on operational database servers.

Data Mart

Data mart contains a subset of organization-wide data. This subset of data is valuable to specific groups of an organization.

In other words, we can claim that data marts contain data specific to a particular group. For example, the marketing data mart may contain data related to items, customers, and sales. Data marts are confined to subjects.

Points to remember about data marts:

- Window-based or Unix/Linux-based servers are used to implement data marts. They are implemented on low-cost servers.
- The implementation data mart cycles is measured in short periods of time, i.e., in weeks rather than months or years.
- The life cycle of a data mart may be complex in long run, if its planning and design are not organization-wide.
- Data marts are small in size.
- Data marts are customized by department.
- The source of a data mart is departmentally structured data warehouse.
- Data mart are flexible.

Enterprise Warehouse

- An enterprise warehouse collects all the information and the subjects spanning an entire organization
- It provides us enterprise-wide data integration.
- The data is integrated from operational systems and external information providers.
- This information can vary from a few gigabytes to hundreds of gigabytes, terabytes or beyond.

Load Manager

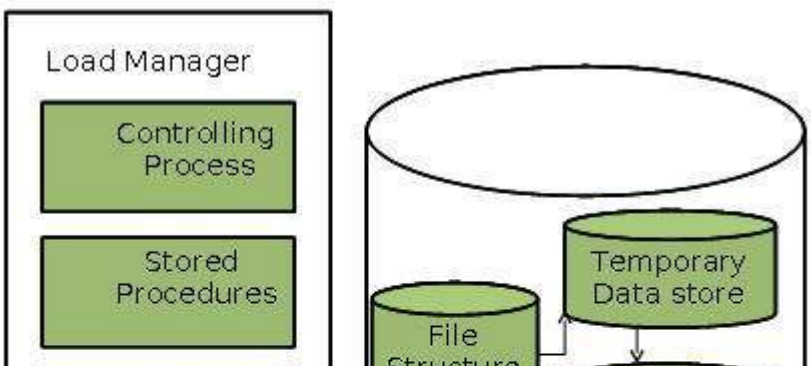
This component performs the operations required to extract and load process.

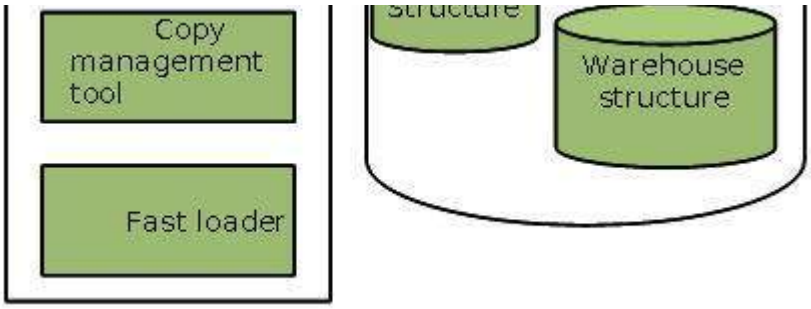
The size and complexity of the load manager varies between specific solutions from one data warehouse to other.

Load Manager Architecture

The load manager performs the following functions:

- Extract the data from source system.
- Fast Load the extracted data into temporary data store.
- Perform simple transformations into structure similar to the one in the data warehouse.





Extract Data from Source

The data is extracted from the operational databases or the external information providers. Gateways is the application programs that are used to extract data. It is supported by underlying DBMS and allows client program to generate SQL to be executed at a server. Open Database Connection*ODBC*, Java Database Connection *JDBC*, are examples of gateway.

Fast Load

- In order to minimize the total load window the data need to be loaded into the warehouse in the fastest possible time.
- The transformations affects the speed of data processing.
- It is more effective to load the data into relational database prior to applying transformations and checks.
- Gateway technology proves to be not suitable, since they tend not be performant when large data volumes are involved.

Simple Transformations

While loading it may be required to perform simple transformations. After this has been completed we are in position to do the complex checks. Suppose we are loading the EPOS sales transaction we need to perform the following checks:

- Strip out all the columns that are not required within the warehouse.
- Convert all the values to required data types.

Warehouse Manager

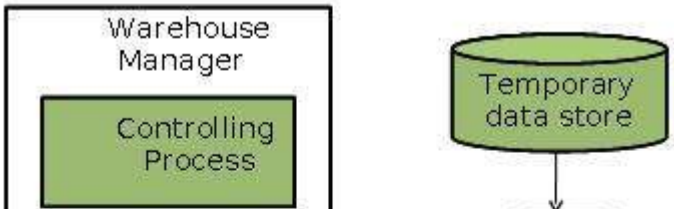
A warehouse manager is responsible for the warehouse management process. It consists of third-party system software, C programs, and shell scripts.

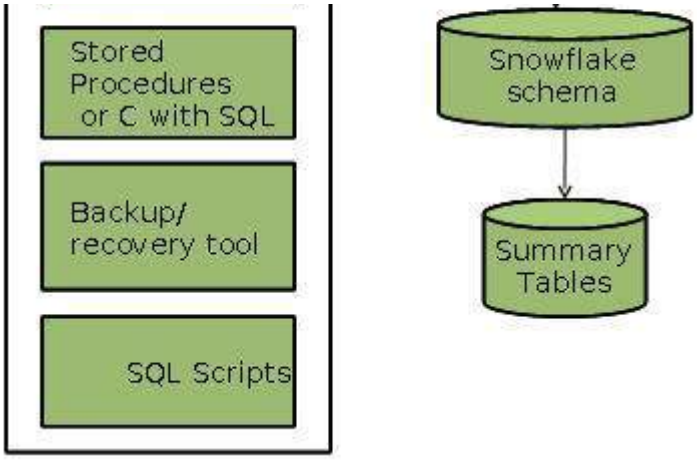
The size and complexity of warehouse managers varies between specific solutions.

Warehouse Manager Architecture

A warehouse manager includes the following:

- The controlling process
- Stored procedures or C with SQL
- Backup/Recovery tool
- SQL Scripts





Operations Performed by Warehouse Manager

- A warehouse manager analyzes the data to perform consistency and referential integrity checks.
- Creates indexes, business views, partition views against the base data.
- Generates new aggregations and updates existing aggregations. Generates normalizations.
- Transforms and merges the source data into the published data warehouse.
- Backup the data in the data warehouse.
- Archives the data that has reached the end of its captured life.

Note: A warehouse Manager also analyzes query profiles to determine index and aggregations are appropriate.

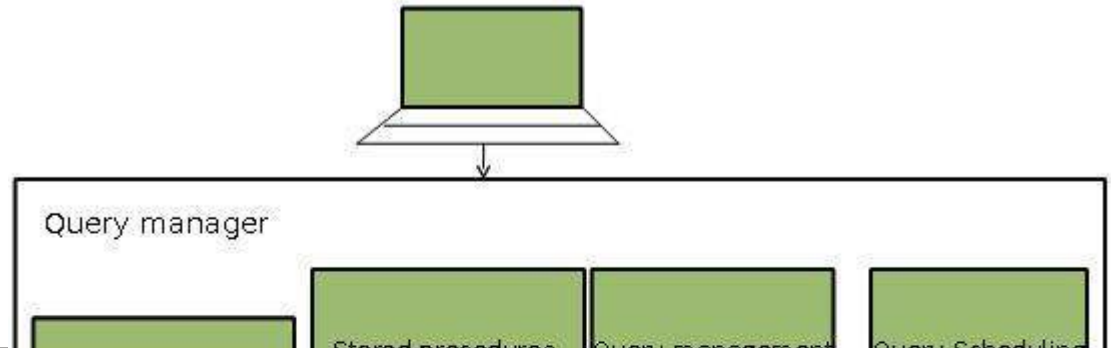
Query Manager

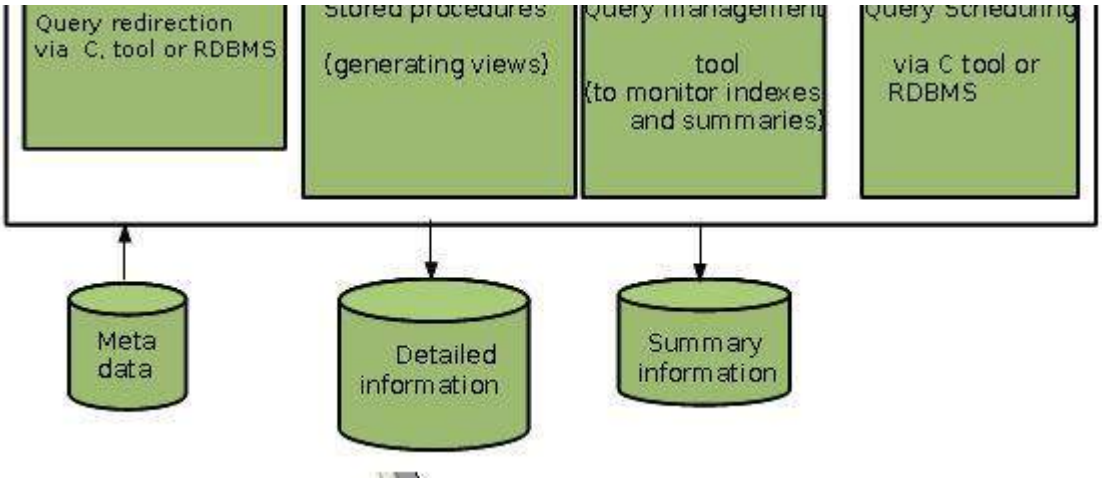
- Query manager is responsible for directing the queries to the suitable tables.
- By directing the queries to appropriate tables, the speed of querying and response generation can be increased.
- Query manager is responsible for scheduling the execution of the queries posed by the user.

Query Manager Architecture

The following screenshot shows the architecture of a query manager. It includes the following:

- Query redirection via C tool or RDBMS
- Stored procedures
- Query management tool
- Query scheduling via C tool or RDBMS
- Query scheduling via third-party software

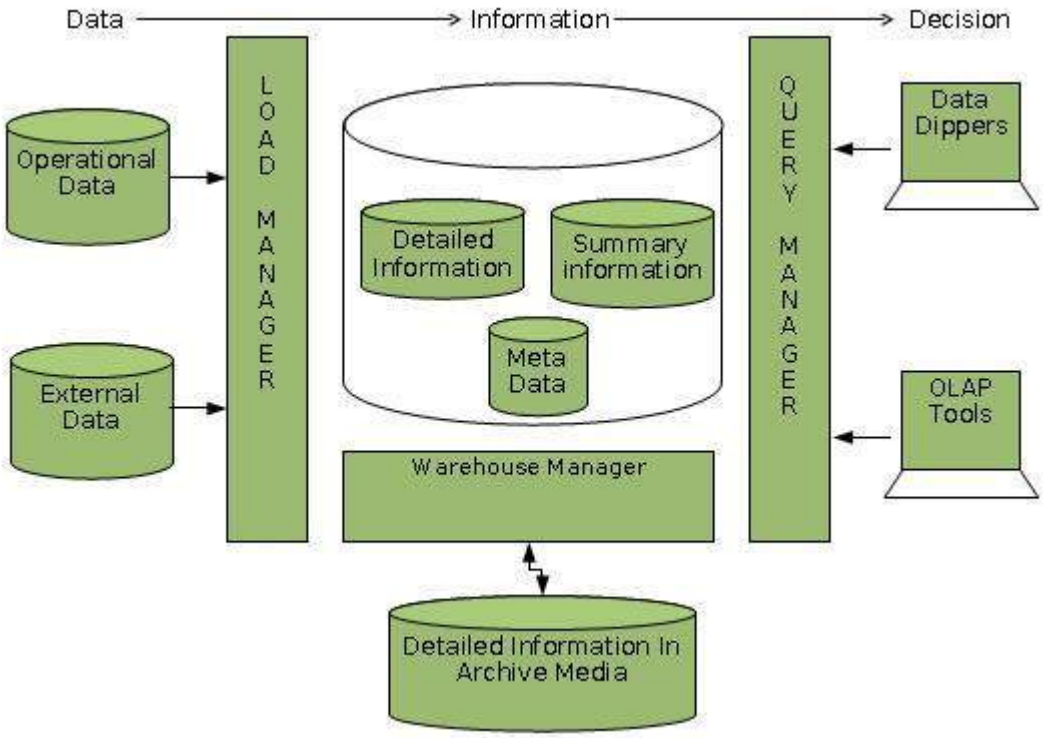




Detailed Information

Detailed information is not kept online, rather it is aggregated to the next level of detail and then archived to tape. The detailed information part of data warehouse keeps the detailed information in the starflake schema. Detailed information is loaded into the data warehouse to supplement the aggregated data.

The following diagram shows a pictorial impression of where detailed information is stored and how it is used.



Note: If detailed information is held offline to minimize disk storage, we should make sure that the data has been extracted, cleaned up, and transformed into starflake schema before it is archived.

Summary Information

Summary Information is a part of data warehouse that stores predefined aggregations. These aggregations are generated by the warehouse manager. Summary Information must be treated as transient. It changes on-the-go in order to respond to the changing query profiles.

Points to remember about summary information.

- Summary information speeds up the performance of common queries.
- It increases the operational cost.

- It needs to be updated whenever new data is loaded into the data warehouse.
- It may not have been backed up, since it can be generated fresh from the detailed information.

DATA WAREHOUSING - OLAP

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 *HOLAP*

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

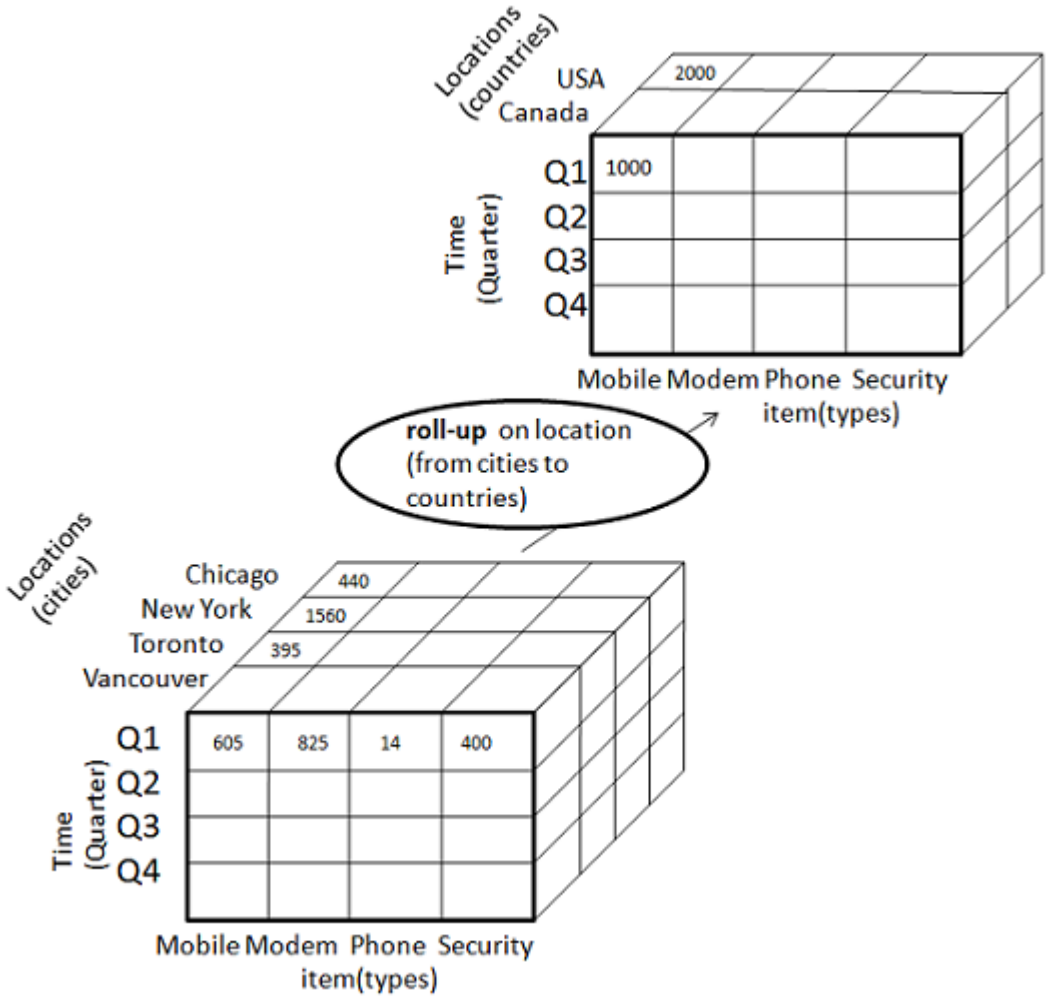
- Pivot *rotate*

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.



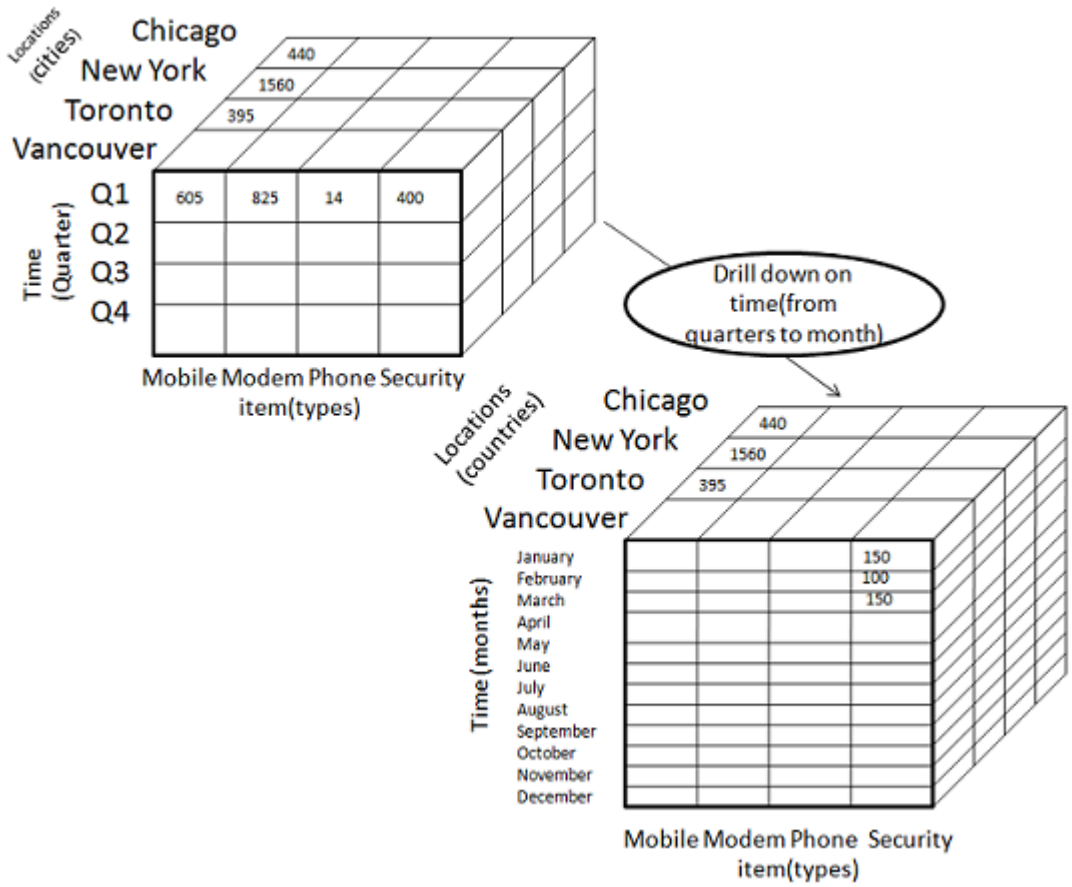
- Roll-up is performed by climbing up a concept hierarchy for the dimension location.
- Initially the concept hierarchy was "street < city < province < country".
- On rolling up, the data is aggregated by ascending the location hierarchy from the level of city to the level of country.
- The data is grouped 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.

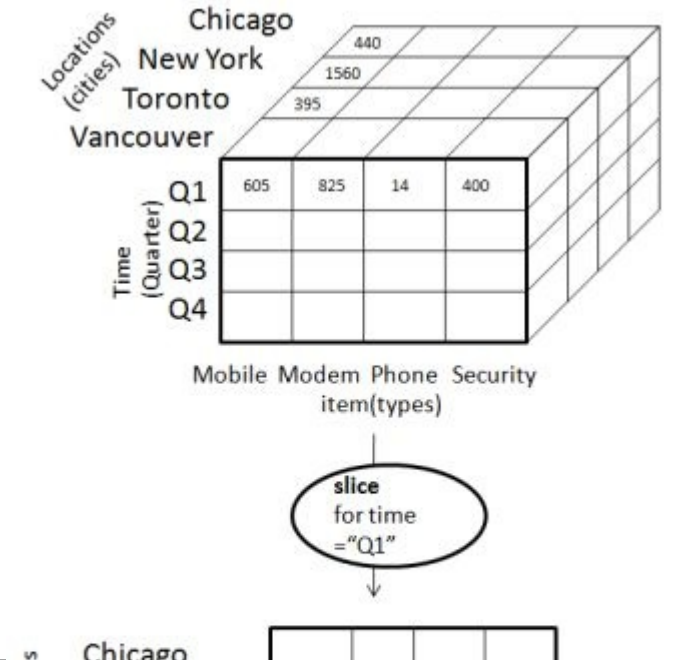
The following diagram illustrates how drill-down works:

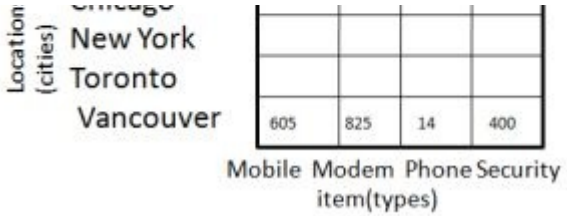


- Drill-down is performed by stepping down a concept hierarchy for the dimension time.
- Initially the concept hierarchy was "day < month < quarter < year."
- On drilling down, the time dimension is descended from the level of quarter to the level of month.
- When drill-down is performed, one or more 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.

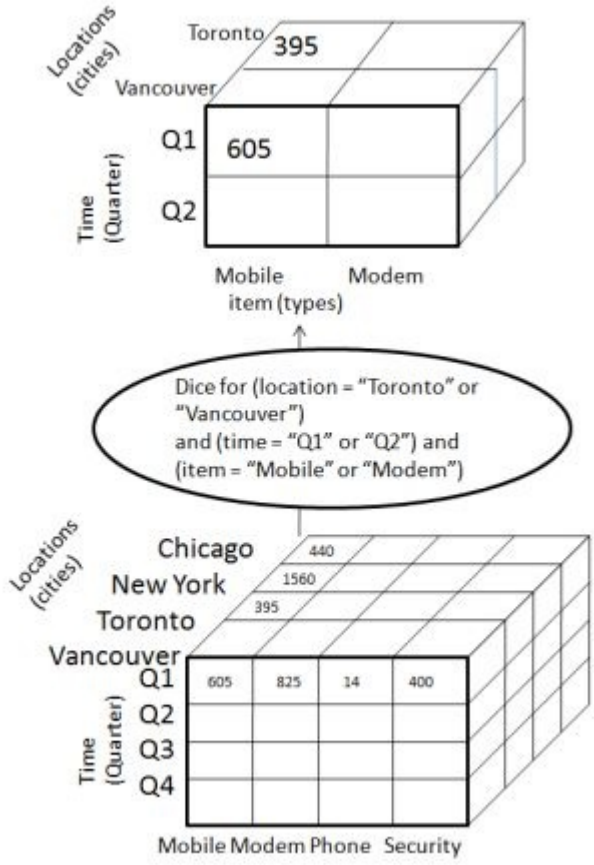




- Here Slice is performed for the dimension "time" using the criterion time = "Q1".
- It will form a new sub-cube by selecting one or more dimensions.

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.

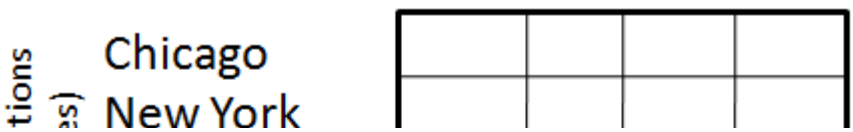


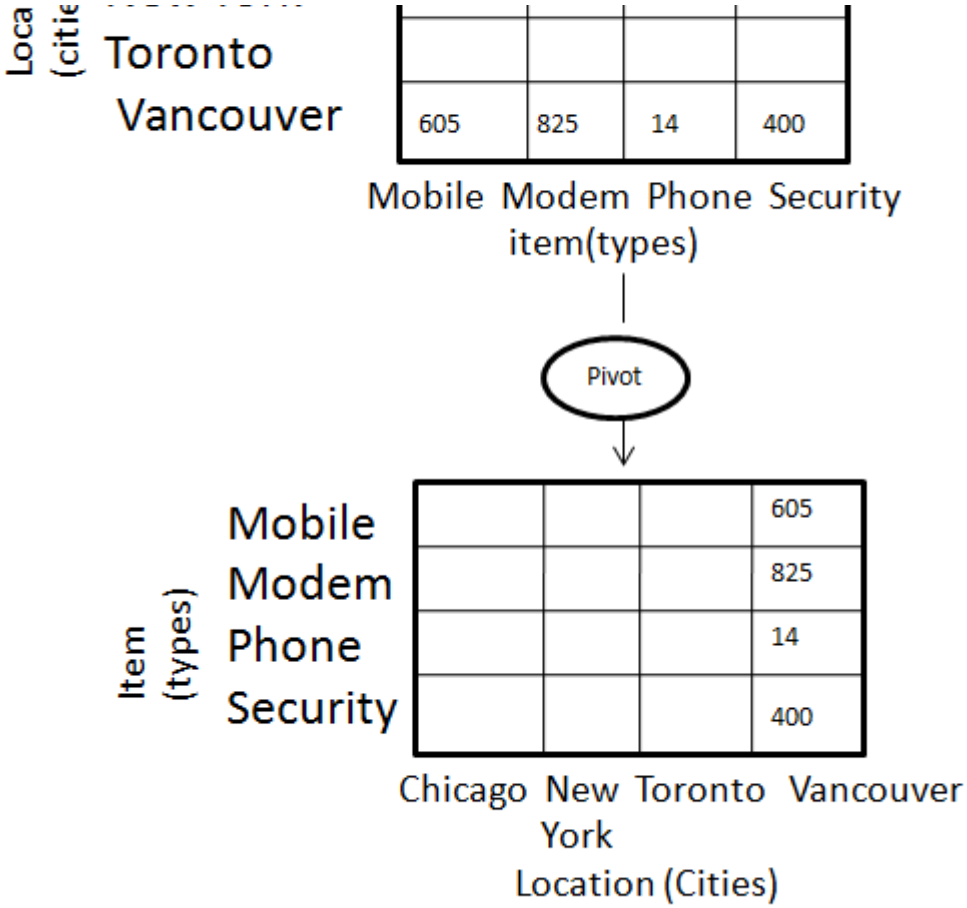
The dice operation on the cube based on the following selection criteria involves three dimensions.

- 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.





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

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.

Highly flexible.	Provides high performance.

DATA WAREHOUSING - RELATIONAL OLAP

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

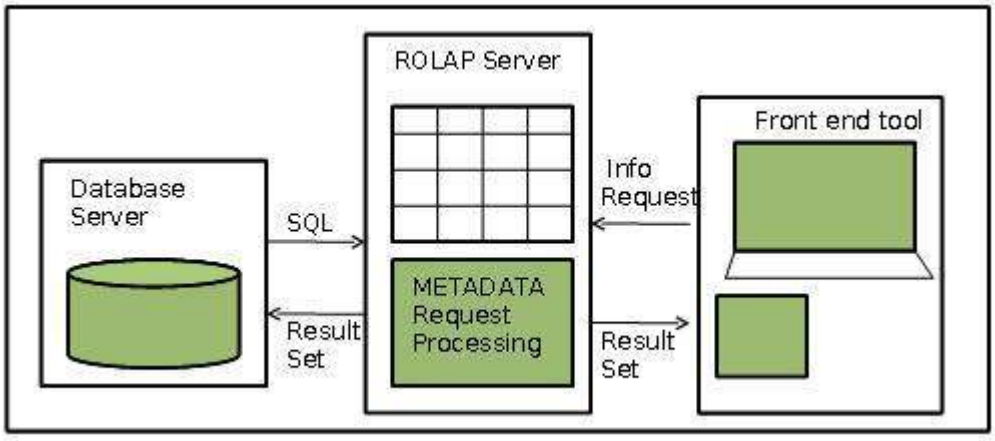
Points to Remember

- 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.

DATA WAREHOUSING - MULTIDIMENSIONAL OLAP

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 servers use two levels of data storage representation to handle dense and sparse data-sets.

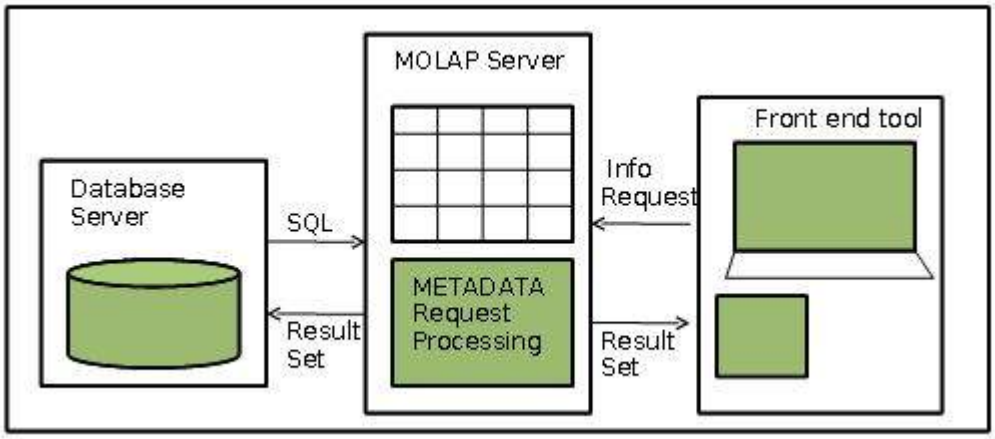
Points to Remember:

- 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

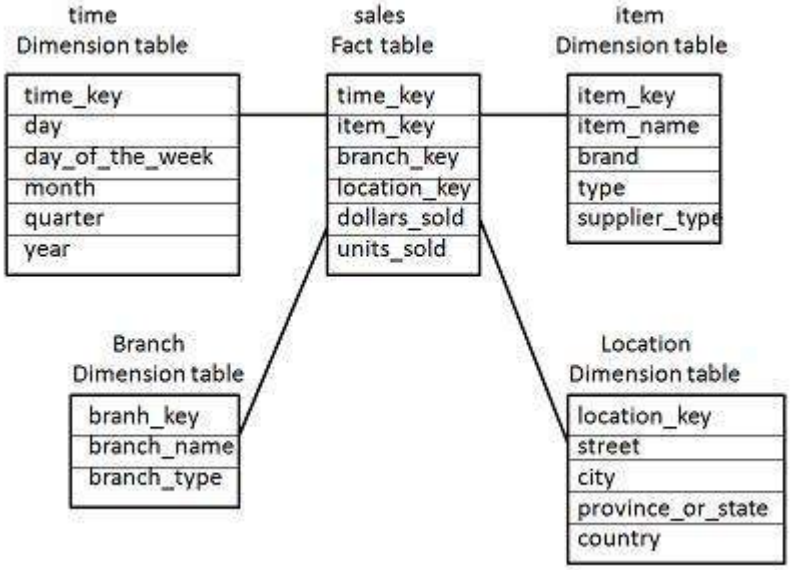
1	Information retrieval is fast.	Information retrieval is comparatively slow.
2	Uses sparse array to store data-sets.	Uses relational table.
3	MOLAP is best suited for inexperienced users, since it is very easy to use.	ROLAP is best suited for experienced users.
4	Maintains a separate database for data cubes.	It may not require space other than available in the Data warehouse.
5	DBMS facility is weak.	DBMS facility is strong.

DATA WAREHOUSING - SCHEMAS

Schema is a logical description of the entire database. It includes the name and description of records of all record types including all associated data-items and aggregates. Much like a database, a data warehouse also requires to maintain a schema. A database uses relational model, while a data warehouse uses Star, Snowflake, and Fact Constellation schema. In this chapter, we will discuss the schemas used in a data warehouse.

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.



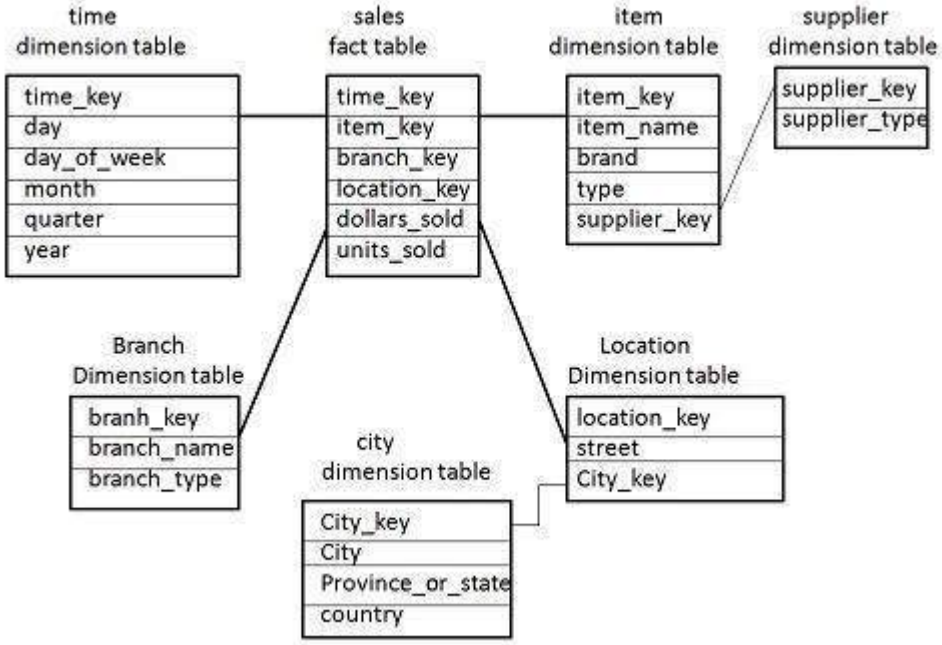
- There is a fact table at the center. It contains the keys to each of four dimensions.
- The fact table also contains the attributes, namely dollars sold and units sold.

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.

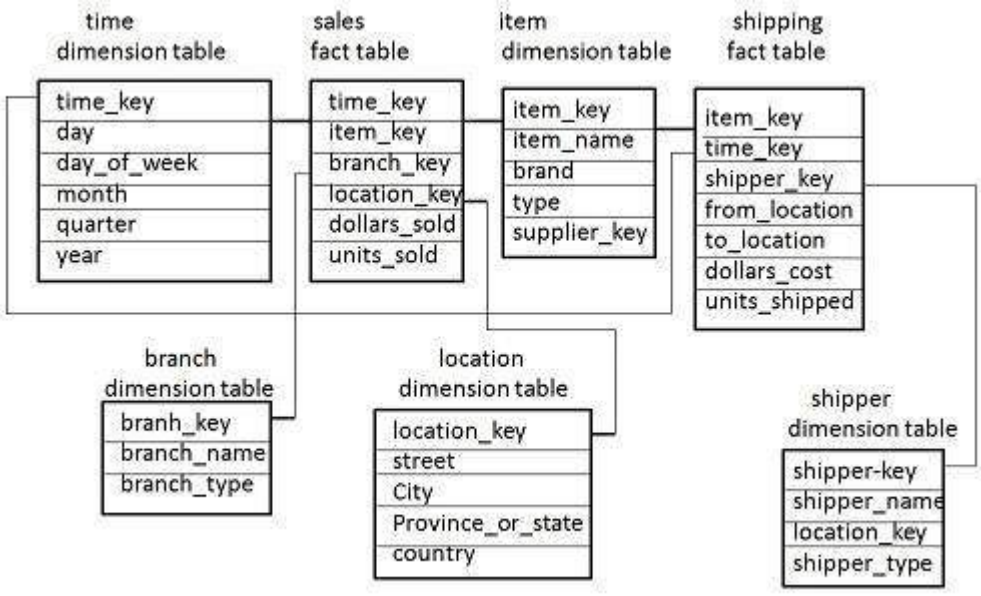


- Now the item dimension table contains the attributes item_key, item_name, type, brand, and supplier-key.
- The supplier key is linked to the supplier dimension table. The supplier dimension table contains the attributes supplier_key and supplier_type.

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

Fact Constellation Schema

- A fact constellation has multiple fact tables. It is also known as galaxy schema.
- The following diagram shows two fact tables, namely sales and shipping.



- The sales fact table is same as that in the star schema.
- The shipping fact table has the 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.

Schema Definition

Multidimensional schema is defined using Data Mining Query Language DMQL. The two primitives, cube definition and dimension definition, can be used for defining the data warehouses and data marts.

Syntax for Cube Definition

```
define cube < cube_name > [ < dimension-list > }: < measure_list >
```

Syntax for Dimension Definition

```
define dimension < dimension_name > as ( < attribute_or_dimension_list > )
```

Star Schema Definition

The star schema that we have discussed can be defined using Data Mining Query Language DMQL as follows:

```
define cube sales star [time, item, branch, location]:

dollars sold = sum(sales in dollars), units sold = count(*)

define dimension time as (time key, day, day of week, month, quarter, year)
define dimension item as (item key, item name, brand, type, supplier type)
define dimension branch as (branch key, branch name, branch type)
define dimension location as (location key, street, city, province or state, country)
```

Snowflake Schema Definition

Snowflake schema can be defined using DMQL as follows:

```
define cube sales snowflake [time, item, branch, location]:

dollars sold = sum(sales in dollars), units sold = count(*)

define dimension time as (time key, day, day of week, month, quarter, year)
define dimension item as (item key, item name, brand, type, supplier (supplier key,
supplier type))
define dimension branch as (branch key, branch name, branch type)
define dimension location as (location key, street, city (city key, city, province or
state, country))
```

Fact Constellation Schema Definition

Fact constellation schema can be defined using DMQL as follows:

```
define cube sales [time, item, branch, location]:

dollars sold = sum(sales in dollars), units sold = count(*)

define dimension time as (time key, day, day of week, month, quarter, year)
define dimension item as (item key, item name, brand, type, supplier type)
define dimension branch as (branch key, branch name, branch type)
define dimension location as (location key, street, city, province or state,country)
define cube shipping [time, item, shipper, from location, to location]:
```

```
dollars cost = sum(cost in dollars), units shipped = count(*)

define dimension time as time in cube sales
define dimension item as item in cube sales
define dimension shipper as (shipper key, shipper name, location as location in cube
sales, shipper type)
define dimension from location as location in cube sales
define dimension to location as location in cube sales
```

DATA WAREHOUSING - PARTITIONING STRATEGY

Partitioning is done to enhance performance and facilitate easy management of data. Partitioning also helps in balancing the various requirements of the system. It optimizes the hardware performance and simplifies the management of data warehouse by partitioning each fact table into multiple separate partitions. In this chapter, we will discuss different partitioning strategies.

Why is it Necessary to Partition?

Partitioning is important for the following reasons:

- For easy management,
- To assist backup/recovery,
- To enhance performance.

For Easy Management

The fact table in a data warehouse can grow up to hundreds of gigabytes in size. This huge size of fact table is very hard to manage as a single entity. Therefore it needs partitioning.

To Assist Backup/Recovery

If we do not partition the fact table, then we have to load the complete fact table with all the data. Partitioning allows us to load only as much data as is required on a regular basis. It reduces the time to load and also enhances the performance of the system.

Note: To cut down on the backup size, all partitions other than the current partition can be marked as read-only. We can then put these partitions into a state where they cannot be modified. Then they can be backed up. It means only the current partition is to be backed up.

To Enhance Performance

By partitioning the fact table into sets of data, the query procedures can be enhanced. Query performance is enhanced because now the query scans only those partitions that are relevant. It does not have to scan the whole data.

Horizontal Partitioning

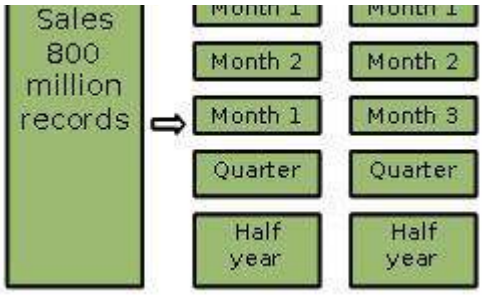
There are various ways in which a fact table can be partitioned. In horizontal partitioning, we have to keep in mind the requirements for manageability of the data warehouse.

Partitioning by Time into Equal Segments

In this partitioning strategy, the fact table is partitioned on the basis of time period. Here each time period represents a significant retention period within the business. For example, if the user queries for **month to date data** then it is appropriate to partition the data into monthly segments. We can reuse the partitioned tables by removing the data in them.

Partition by Time into Different-sized Segments

This kind of partition is done where the aged data is accessed infrequently. It is implemented as a set of small partitions for relatively current data, larger partition for inactive data.



Points to Note

- The detailed information remains available online.
- The number of physical tables is kept relatively small, which reduces the operating cost.
- This technique is suitable where a mix of data dipping recent history and data mining through entire history is required.
- This technique is not useful where the partitioning profile changes on a regular basis, because repartitioning will increase the operation cost of data warehouse.

Partition on a Different Dimension

The fact table can also be partitioned on the basis of dimensions other than time such as product group, region, supplier, or any other dimension. Let's have an example.

Suppose a market function has been structured into distinct regional departments like on a **state by state** basis. If each region wants to query on information captured within its region, it would prove to be more effective to partition the fact table into regional partitions. This will cause the queries to speed up because it does not require to scan information that is not relevant.

Points to Note

- The query does not have to scan irrelevant data which speeds up the query process.
- This technique is not appropriate where the dimensions are unlikely to change in future. So, it is worth determining that the dimension does not change in future.
- If the dimension changes, then the entire fact table would have to be repartitioned.

Note: We recommend to perform the partition only on the basis of time dimension, unless you are certain that the suggested dimension grouping will not change within the life of the data warehouse.

Partition by Size of Table

When there are no clear basis for partitioning the fact table on any dimension, then we should **partition the fact table on the basis of their size.** We can set the predetermined size as a critical point. When the table exceeds the predetermined size, a new table partition is created.

Points to Note

- This partitioning is complex to manage.

It requires metadata to identify what data is stored in each partition.

Partitioning Dimensions

If a dimension contains large number of entries, then it is required to partition the dimensions. Here we have to check the size of a dimension.

Consider a large design that changes over time. If we need to store all the variations in order to apply comparisons, that dimension may be very large. This would definitely affect the response time.

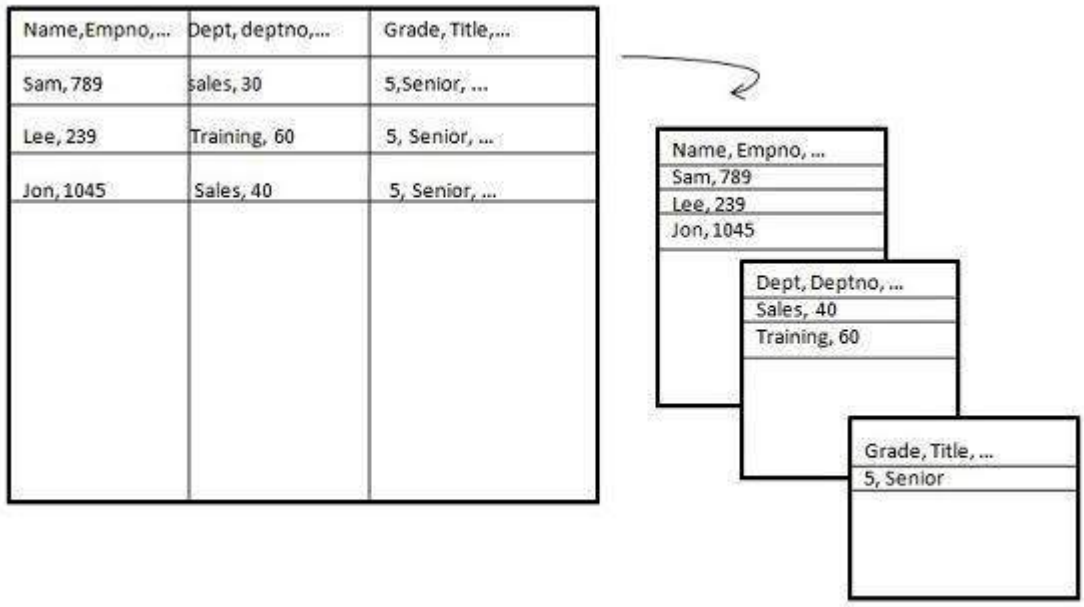
Round Robin Partitions

In the round robin technique, when a new partition is needed, the old one is archived. It uses metadata to allow user access tool to refer to the correct table partition.

This technique makes it easy to automate table management facilities within the data warehouse.

Vertical Partition

Vertical partitioning, splits the data vertically. The following images depicts how vertical partitioning is done.



Vertical partitioning can be performed in the following two ways:

- Normalization
- Row Splitting

Normalization

Normalization is the standard relational method of database organization. In this method, the rows are collapsed into a single row, hence it reduce space. Take a look at the following tables that show how normalization is performed.

Table before Normalization

Product_id	Qty	Value	sales_date	Store_id	Store_name	Location	Region
30	5	3.67	3-Aug-13	16	sunny	Bangalore	S
35	4	5.33	3-Sep-13	16	sunny	Bangalore	S
40	5	2.50	3-Sep-13	64	san	Mumbai	W
45	7	5.66	3-Sep-13	16	sunny	Bangalore	S

Table after Normalization

Store_id	Store_name	Location	Region

16	sunny	Bangalore	W
64	san	Mumbai	S

Product_id	Quantity	Value	sales_date	Store_id
30	5	3.67	3-Aug-13	16
35	4	5.33	3-Sep-13	16
40	5	2.50	3-Sep-13	64
45	7	5.66	3-Sep-13	16

Row Splitting

Row splitting tends to leave a one-to-one map between partitions. The motive of row splitting is to speed up the access to large table by reducing its size.

Note: While using vertical partitioning, make sure that there is no requirement to perform a major join operation between two partitions.

Identify Key to Partition

It is very crucial to choose the right partition key. Choosing a wrong partition key will lead to reorganizing the fact table. Let's have an example. Suppose we want to partition the following table.

```
Account_Txn_Table
transaction_id
account_id
transaction_type
value
transaction_date
region
branch_name
```

We can choose to partition on any key. The two possible keys could be

- region
- transaction_date

Suppose the business is organized in 30 geographical regions and each region has different number of branches. That will give us 30 partitions, which is reasonable. This partitioning is good enough because our requirements capture has shown that a vast majority of queries are restricted to the user's own business region.

If we partition by transaction_date instead of region, then the latest transaction from every region will be in one partition. Now the user who wants to look at data within his own region has to query across multiple partitions.

Hence it is worth determining the right partitioning key.

DATA WAREHOUSING - METADATA CONCEPTS

What is Metadata?

Metadata is simply defined as data about data. The data that is used to represent other data is known as metadata. For example, the index of a book serves as a metadata for the contents in the book. In other words, we can say that metadata is the summarized data that leads us to detailed data. In terms of data warehouse, we can define metadata as follows.

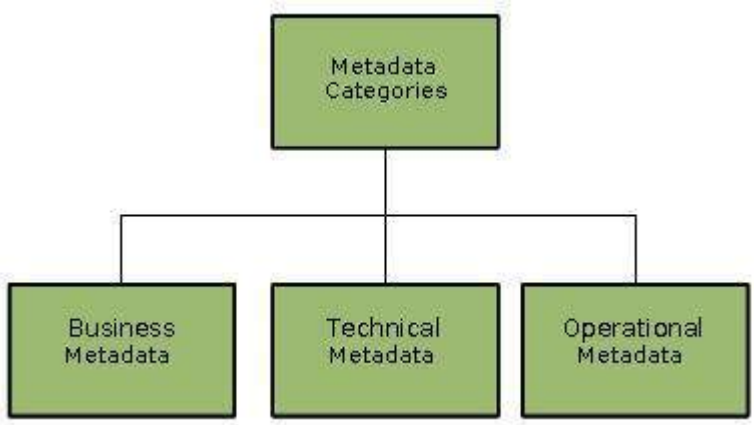
- Metadata is the road-map to a data warehouse.
- Metadata in a data warehouse defines the warehouse objects.
- Metadata acts as a directory. This directory helps the decision support system to locate the contents of a data warehouse.

Note: In a data warehouse, we create metadata for the data names and definitions of a given data warehouse. Along with this metadata, additional metadata is also created for time-stamping any extracted data, the source of extracted data.

Categories of Metadata

Metadata can be broadly categorized into three categories:

- **Business Metadata** - It has the data ownership information, business definition, and changing policies.
- **Technical Metadata** - It includes database system names, table and column names and sizes, data types and allowed values. Technical metadata also includes structural information such as primary and foreign key attributes and indices.
- **Operational Metadata** - It includes currency of data and data lineage. Currency of data means whether the data is active, archived, or purged. Lineage of data means the history of data migrated and transformation applied on it.



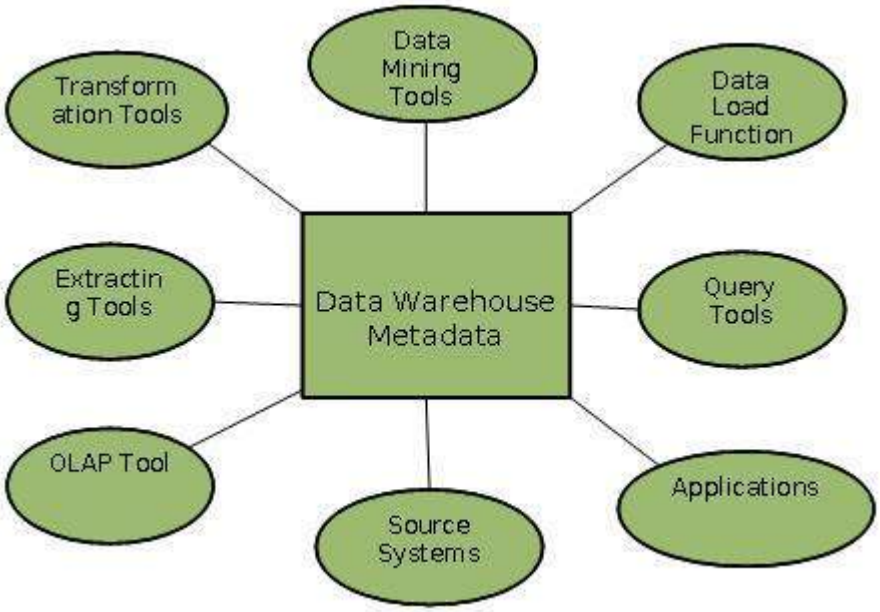
Role of Metadata

Metadata has a very important role in a data warehouse. The role of metadata in a warehouse is different from the warehouse data, yet it plays an important role. The various roles of metadata are explained below.

- Metadata acts as a directory.
- This directory helps the decision support system to locate the contents of the data warehouse.
- Metadata helps in decision support system for mapping of data when data is transformed from operational environment to data warehouse environment.
- Metadata helps in summarization between current detailed data and highly summarized data.
- Metadata also helps in summarization between lightly detailed data and highly summarized data.
- Metadata is used for query tools.
- Metadata is used in extraction and cleansing tools.
- Metadata is used in reporting tools.
- Metadata is used in transformation tools.

- Metadata plays an important role in loading functions.

The following diagram shows the roles of metadata.



Metadata Respiratory

Metadata respiratory is an integral part of a data warehouse system. It has the following metadata:

- **Definition of data warehouse** - It includes the description of structure of data warehouse. The description is defined by schema, view, hierarchies, derived data definitions, and data mart locations and contents.
- **Business metadata** - It contains has the data ownership information, business definition, and changing policies.
- **Operational Metadata** - It includes currency of data and data lineage. Currency of data means whether the data is active, archived, or purged. Lineage of data means the history of data migrated and transformation applied on it.
- **Data for mapping from operational environment to data warehouse** - It includes the source databases and their contents, data extraction, data partition cleaning, transformation rules, data refresh and purging rules.
- **Algorithms for summarization** - It includes dimension algorithms, data on granularity, aggregation, summarizing, etc.

Challenges for Metadata Management

The importance of metadata can not be overstated. Metadata helps in driving the accuracy of reports, validates data transformation, and ensures the accuracy of calculations. Metadata also enforces the definition of business terms to business end-users. With all these uses of metadata, it also has its challenges. Some of the challenges are discussed below.

- Metadata in a big organization is scattered across the organization. This metadata is spread in spreadsheets, databases, and applications.
- Metadata could be present in text files or multimedia files. To use this data for information management solutions, it has to be correctly defined.
- There are no industry-wide accepted standards. Data management solution vendors have narrow focus.
- There are no easy and accepted methods of passing metadata.

DATA WAREHOUSING - DATA MARTING

Why Do We Need a Data Mart?

Listed below are the reasons to create a data mart:

- To partition data in order to impose **access control strategies.**
- To speed up the queries by reducing the volume of data to be scanned.
- To segment data into different hardware platforms.
- To structure data in a form suitable for a user access tool.

Note: Do not data mart for any other reason since the operation cost of data marting could be very high. Before data marting, make sure that data marting strategy is appropriate for your particular solution.

Cost-effective Data Marting

Follow the steps given below to make data marting cost-effective:

- Identify the Functional Splits
- Identify User Access Tool Requirements
- Identify Access Control Issues

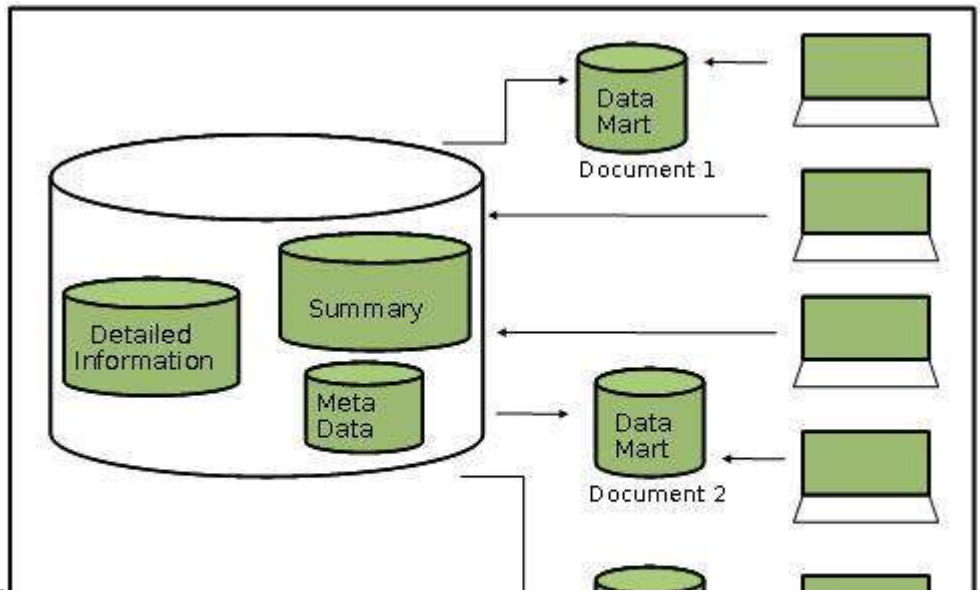
Identify the Functional Splits

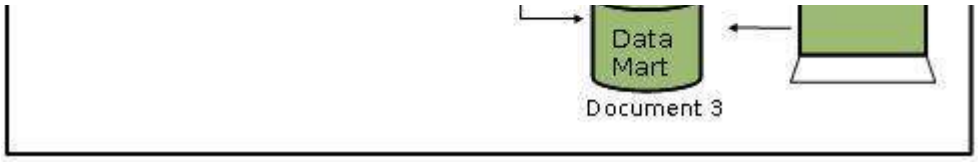
In this step, we determine if the organization has natural functional splits. We look for departmental splits, and we determine whether the way in which departments use information tend to be in isolation from the rest of the organization. Let's have an example.

Consider a retail organization, where each merchant is accountable for maximizing the sales of a group of products. For this, the following are the valuable information:

- sales transaction on a daily basis
- sales forecast on a weekly basis
- stock position on a daily basis
- stock movements on a daily basis

As the merchant is not interested in the products they are not dealing with, the data marting is a subset of the data dealing which the product group of interest. The following diagram shows data marting for different users.





Given below are the issues to be taken into account while determining the functional split:

- The structure of the department may change.
- The products might switch from one department to other.
- The merchant could query the sales trend of other products to analyze what is happening to the sales.

Note: We need to determine the business benefits and technical feasibility of using a data mart.

Identify User Access Tool Requirements

We need data marts to support **user access tools** that require internal data structures. The data in such structures are outside the control of data warehouse but need to be populated and updated on a regular basis.

There are some tools that populate directly from the source system but some cannot. Therefore additional requirements outside the scope of the tool are needed to be identified for future.

Note: In order to ensure consistency of data across all access tools, the data should not be directly populated from the data warehouse, rather each tool must have its own data mart.

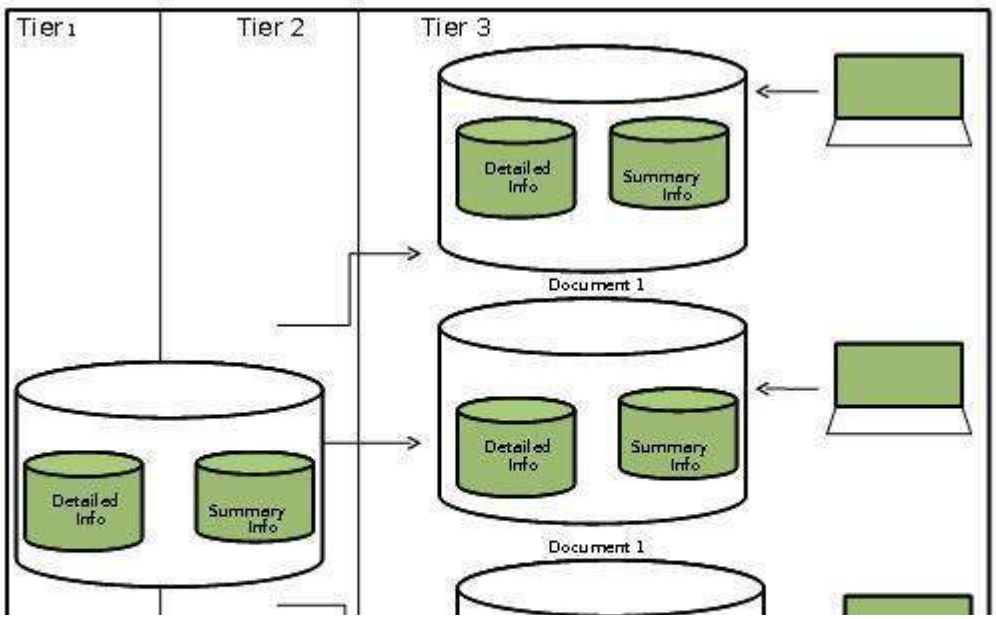
Identify Access Control Issues

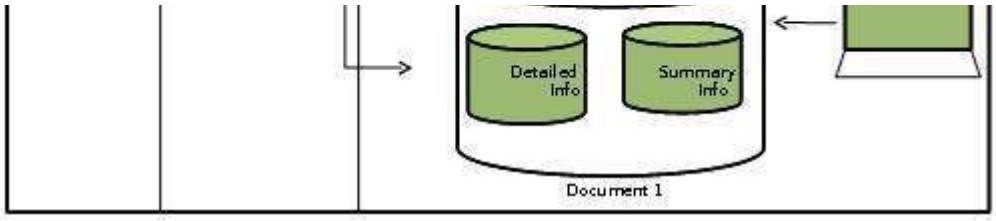
There should to be privacy rules to ensure the data is accessed by authorized users only. For example a data warehouse for retail banking institution ensures that all the accounts belong to the same legal entity. Privacy laws can force you to totally prevent access to information that is not owned by the specific bank.

Data marts allow us to build a complete wall by physically separating data segments within the data warehouse. To avoid possible privacy problems, the detailed data can be removed from the data warehouse. We can create data mart for each legal entity and load it via data warehouse, with detailed account data.

Designing Data Marts

Data marts should be designed as a smaller version of starflake schema within the data warehouse and should match with the database design of the data warehouse. It helps in maintaining control over database instances.





The summaries are data marted in the same way as they would have been designed within the data warehouse. Summary tables help to utilize all dimension data in the starflake schema.

Cost of Data Marting

The cost measures for data marting are as follows:

- Hardware and Software Cost
- Network Access
- Time Window Constraints

Hardware and Software Cost

Although data marts are created on the same hardware, they require some additional hardware and software. To handle user queries, it requires additional processing power and disk storage. If detailed data and the data mart exist within the data warehouse, then we would face additional cost to store and manage replicated data.

Note: Data marting is more expensive than aggregations, therefore it should be used as an additional strategy and not as an alternative strategy.

Network Access

A data mart could be on a different location from the data warehouse, so we should ensure that the LAN or WAN has the capacity to handle the data volumes being transferred within the **data mart load process.**

Time Window Constraints

The extent to which a data mart loading process will eat into the available time window depends on the complexity of the transformations and the data volumes being shipped. The determination of how many data marts are possible depends on:

- Network capacity.
- Time window available
- Volume of data being transferred
- Mechanisms being used to insert data into a data mart

DATA WAREHOUSING - SYSTEM MANAGERS

System management is mandatory for the successful implementation of a data warehouse. The most important system managers are:

- System configuration manager
- System scheduling manager
- System event manager
- System database manager
- System backup recovery manager

System Configuration Manager

- The system configuration manager is responsible for the management of the setup and configuration of data warehouse.
- The structure of configuration manager varies from one operating system to another.
- In Unix structure of configuration, the manager varies from vendor to vendor.
- Configuration managers have single user interface.
- The interface of configuration manager allows us to control all aspects of the system.

Note: The most important configuration tool is the I/O manager.

System Scheduling Manager

System Scheduling Manager is responsible for the successful implementation of the data warehouse. Its purpose is to schedule ad hoc queries. Every operating system has its own scheduler with some form of batch control mechanism. The list of features a system scheduling manager must have is as follows:

- Work across cluster or MPP boundaries
- Deal with international time differences
- Handle job failure
- Handle multiple queries
- Support job priorities
- Restart or re-queue the failed jobs
- Notify the user or a process when job is completed
- Maintain the job schedules across system outages
- Re-queue jobs to other queues
- Support the stopping and starting of queues
- Log Queued jobs
- Deal with inter-queue processing

Note: The above list can be used as evaluation parameters for the evaluation of a good scheduler.

Some important jobs that a scheduler must be able to handle are as follows:

- Daily and ad hoc query scheduling
- Execution of regular report requirements
- Data load
- Data processing
- Index creation
- Backup
- Aggregation creation
- Data transformation

Note: If the data warehouse is running on a cluster or MPP architecture, then the system scheduling manager must be capable of running across the architecture.

System Event Manager

The event manager is a kind of a software. The event manager manages the events that are defined on the data warehouse system. We cannot manage the data warehouse manually because the structure of data warehouse is very complex. Therefore we need a tool that automatically handles all the events without any intervention of the user.

Note: The Event manager monitors the events occurrences and deals with them. The event manager also tracks the myriad of things that can go wrong on this complex data warehouse system.

Events

Events are the actions that are generated by the user or the system itself. It may be noted that the event is a measurable, observable, occurrence of a defined action.

Given below is a list of common events that are required to be tracked.

- Hardware failure
- Running out of space on certain key disks
- A process dying
- A process returning an error
- CPU usage exceeding an 805 threshold
- Internal contention on database serialization points
- Buffer cache hit ratios exceeding or failure below threshold
- A table reaching to maximum of its size
- Excessive memory swapping
- A table failing to extend due to lack of space
- Disk exhibiting I/O bottlenecks
- Usage of temporary or sort area reaching a certain thresholds
- Any other database shared memory usage

The most important thing about events is that they should be capable of executing on their own. Event packages define the procedures for the predefined events. The code associated with each event is known as event handler. This code is executed whenever an event occurs.

System and Database Manager

System and database manager may be two separate pieces of software, but they do the same job. The objective of these tools is to automate certain processes and to simplify the execution of others. The criteria for choosing a system and the database manager are as follows:

- increase user's quota.
- assign and de-assign roles to the users
- assign and de-assign the profiles to the users
- perform database space management
- monitor and report on space usage
- tidy up fragmented and unused space
- add and expand the space
- add and remove users
- manage user password
- manage summary or temporary tables
- assign or deassign temporary space to and from the user
- reclaim the space form old or out-of-date temporary tables
- manage error and trace logs
- to browse log and trace files

- redirect error or trace information
- switch on and off error and trace logging
- perform system space management
- monitor and report on space usage
- clean up old and unused file directories
- add or expand space.

System Backup Recovery Manager

The backup and recovery tool makes it easy for operations and management staff to back-up the data. Note that the system backup manager must be integrated with the schedule manager software being used. The important features that are required for the management of backups are as follows:

- Scheduling
- Backup data tracking
- Database awareness

Backups are taken only to protect against data loss. Following are the important points to remember.

- The backup software will keep some form of database of where and when the piece of data was backed up.
- The backup recovery manager must have a good front-end to that database.
- The backup recovery software should be database aware.
- Being aware of the database, the software then can be addressed in database terms, and will not perform backups that would not be viable.

DATA WAREHOUSING - PROCESS MANAGERS

Process managers are responsible for maintaining the flow of data both into and out of the data warehouse. There are three different types of process managers:

- Load manager
- Warehouse manager
- Query manager

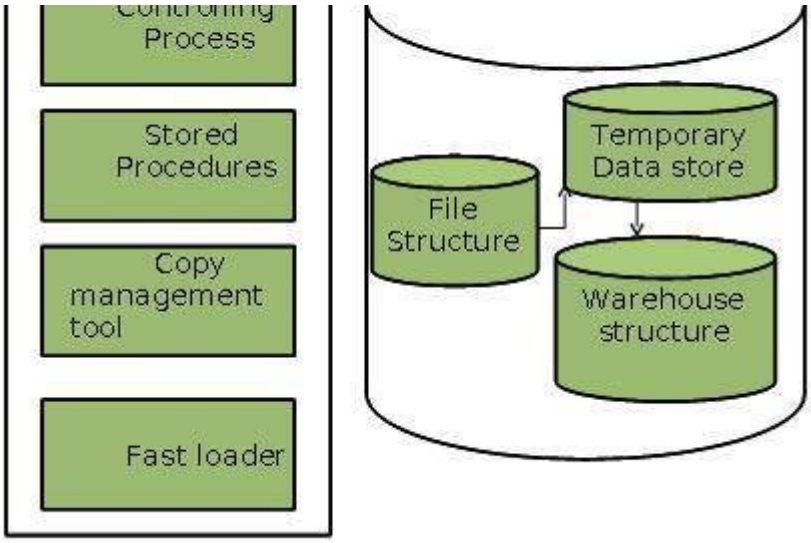
Data Warehouse Load Manager

Load manager performs the operations required to extract and load the data into the database. The size and complexity of a load manager varies between specific solutions from one data warehouse to another.

Load Manager Architecture

The load manager does performs the following functions:

- Extract data from the source system.
- Fast load the extracted data into temporary data store.
- Perform simple transformations into structure similar to the one in the data warehouse.



Extract Data from Source

The data is extracted from the operational databases or the external information providers. Gateways are the application programs that are used to extract data. It is supported by underlying DBMS and allows the client program to generate SQL to be executed at a server. Open Database Connection ODBC and Java Database Connection JDBC are examples of gateway.

Fast Load

- In order to minimize the total load window, the data needs to be loaded into the warehouse in the fastest possible time.
- Transformations affect the speed of data processing.
- It is more effective to load the data into a relational database prior to applying transformations and checks.
- Gateway technology is not suitable, since they are inefficient when large data volumes are involved.

Simple Transformations

While loading, it may be required to perform simple transformations. After completing simple transformations, we can do complex checks. Suppose we are loading the EPOS sales transaction, we need to perform the following checks:

- Strip out all the columns that are not required within the warehouse.
- Convert all the values to required data types.

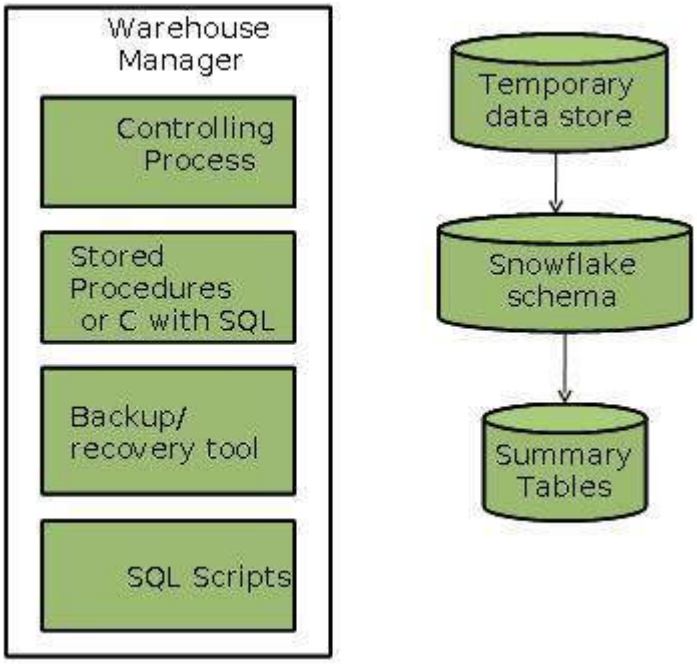
Warehouse Manager

The warehouse manager is responsible for the warehouse management process. It consists of a third-party system software, C programs, and shell scripts. The size and complexity of a warehouse manager varies between specific solutions.

Warehouse Manager Architecture

A warehouse manager includes the following:

- The controlling process
- Stored procedures or C with SQL
- Backup/Recovery tool
- SQL scripts



Functions of Warehouse Manager

A warehouse manager performs the following functions:

- Analyzes the data to perform consistency and referential integrity checks.
- Creates indexes, business views, partition views against the base data.
- Generates new aggregations and updates the existing aggregations.
- Generates normalizations.
- Transforms and merges the source data of the temporary store into the published data warehouse.
- Backs up the data in the data warehouse.
- Archives the data that has reached the end of its captured life.

Note: A warehouse Manager analyzes query profiles to determine whether the index and aggregations are appropriate.

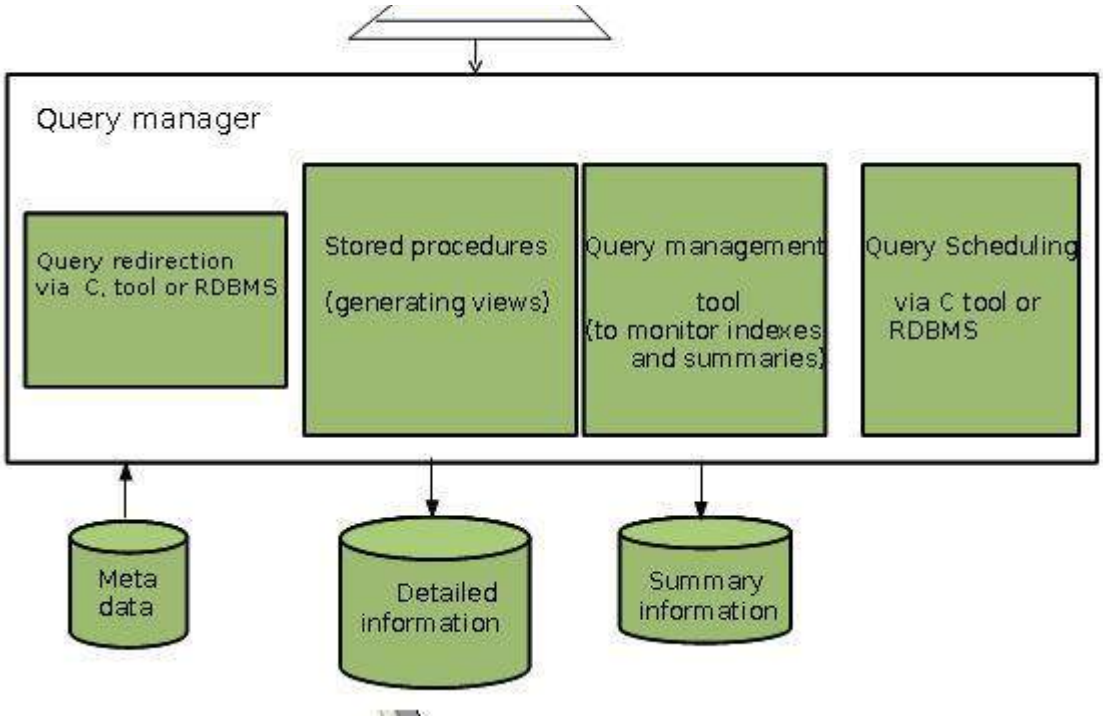
Query Manager

The query manager is responsible for directing the queries to suitable tables. By directing the queries to appropriate tables, it speeds up the query request and response process. In addition, the query manager is responsible for scheduling the execution of the queries posted by the user.

Query Manager Architecture

A query manager includes the following components:

- Query redirection via C tool or RDBMS
- Stored procedures
- Query management tool
- Query scheduling via C tool or RDBMS
- Query scheduling via third-party software



Functions of Query Manager

- It presents the data to the user in a form they understand.
- It schedules the execution of the queries posted by the end-user.
- It stores query profiles to allow the warehouse manager to determine which indexes and aggregations are appropriate.

DATA WAREHOUSING - SECURITY

The objective of a data warehouse is to make large amounts of data easily accessible to the users, hence allowing the users to extract information about the business as a whole. But we know that there could be some security restrictions applied on the data that can be an obstacle for accessing the information. If the analyst has a restricted view of data, then it is impossible to capture a complete picture of the trends within the business.

The data from each analyst can be summarized and passed on to management where the different summaries can be aggregated. As the aggregations of summaries cannot be the same as that of the aggregation as a whole, it is possible to miss some information trends in the data unless someone is analyzing the data as a whole.

Security Requirements

Adding security features affect the performance of the data warehouse, therefore it is important to determine the security requirements as early as possible. It is difficult to add security features after the data warehouse has gone live.

During the design phase of the data warehouse, we should keep in mind what data sources may be added later and what would be the impact of adding those data sources. We should consider the following possibilities during the design phase.

- Whether the new data sources will require new security and/or audit restrictions to be implemented?
- Whether the new users added who have restricted access to data that is already generally available?

This situation arises when the future users and the data sources are not well known. In such a situation, we need to use the knowledge of business and the objective of data warehouse to know likely requirements.

The following activities get affected by security measures:

- User access
- Data load
- Data movement
- Query generation

User Access

We need to first classify the data and then classify the users on the basis of the data they can access. In other words, the users are classified according to the data they can access.

Data Classification

The following two approaches can be used to classify the data:

- Data can be classified according to its sensitivity. Highly-sensitive data is classified as highly restricted and less-sensitive data is classified as less restrictive.
- Data can also be classified according to the job function. This restriction allows only specific users to view particular data. Here we restrict the users to view only that part of the data in which they are interested and are responsible for.

There are some issues in the second approach. To understand, let's have an example. Suppose you are building the data warehouse for a bank. Consider that the data being stored in the data warehouse is the transaction data for all the accounts. The question here is, who is allowed to see the transaction data. The solution lies in classifying the data according to the function.

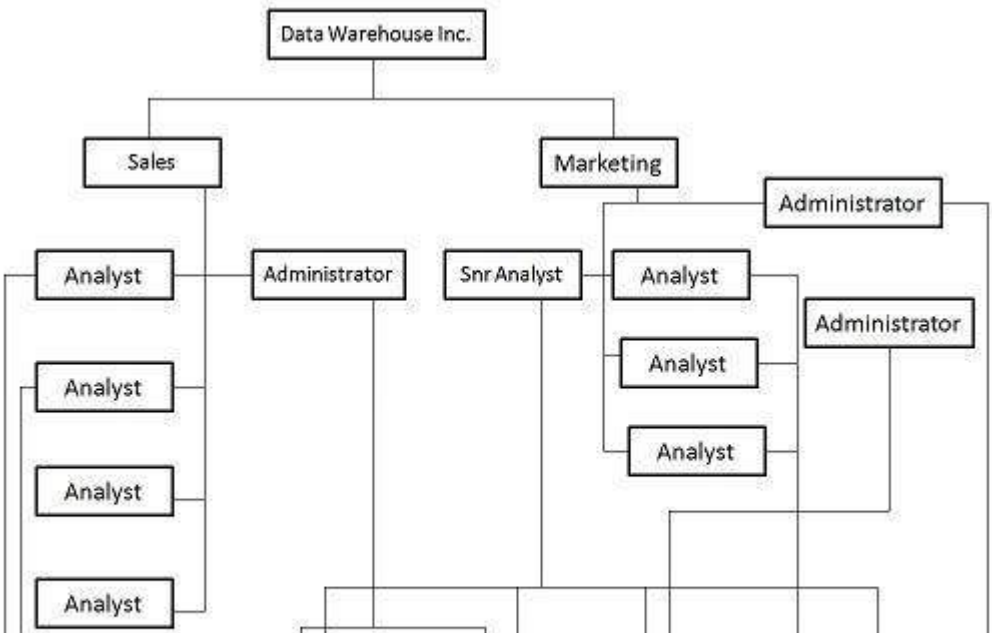
User classification

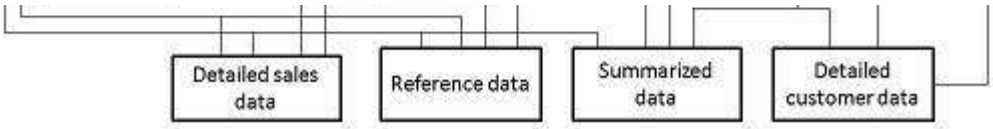
The following approaches can be used to classify the users:

- Users can be classified as per the hierarchy of users in an organization, i.e., users can be classified by departments, sections, groups, and so on.
- Users can also be classified according to their role, with people grouped across departments based on their role.

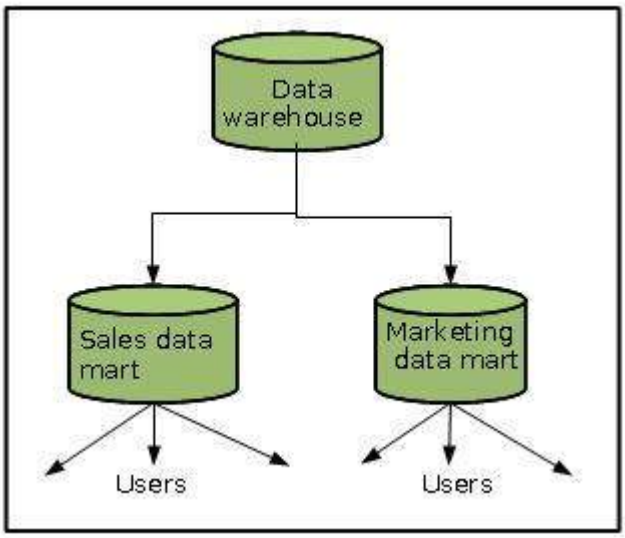
Classification on basis of Department

Let's have an example of a data warehouse where the users are from sales and marketing department. We can have security by top-to-down company view, with access centered on the different departments. But there could be some restrictions on users at different levels. This structure is shown in the following diagram.



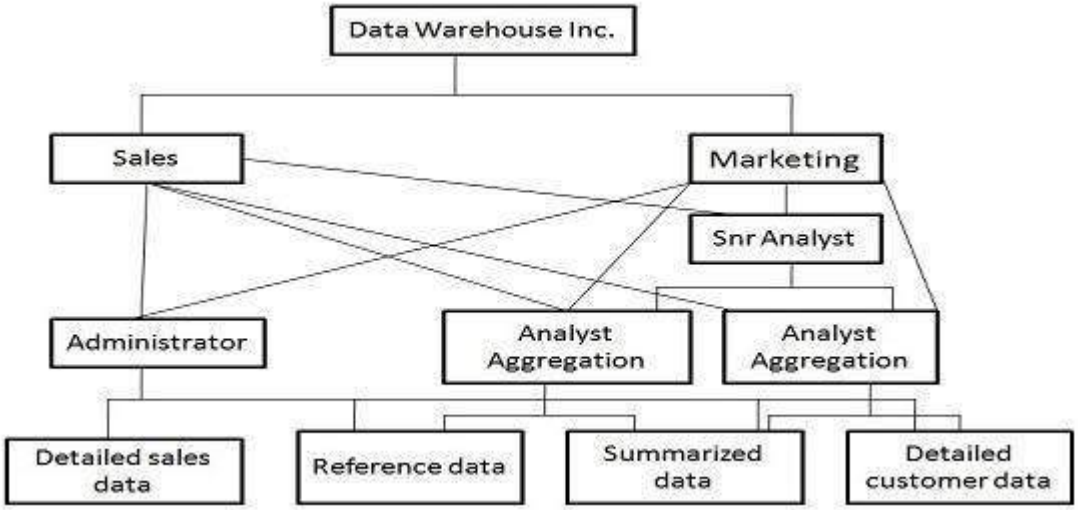


But if each department accesses different data, then we should design the security access for each department separately. This can be achieved by departmental data marts. Since these data marts are separated from the data warehouse, we can enforce separate security restrictions on each data mart. This approach is shown in the following figure.



Classification on basis of Role

If the data is generally available to all the departments, then it is useful to follow the role access hierarchy. In other words, if the data is generally accessed by all If the data is generally available to all the departments, then it is useful to follow the role access hierarchy. In other words, if the data is generally accessed by all



Audit Requirements

Auditing is a subset of security, a costly activity. Auditing can cause heavy overheads on the system. To complete an audit in time, we require more hardware and therefore, it is recommended that wherever possible, auditing should be switched off. Audit requirements can be categorized as follows:

- Connections
- Disconnections
- Data access

- Data change

Note : For each of the above-mentioned categories, it is necessary to audit success, failure, or both. From the perspective of security reasons, the auditing of failures are very important. Auditing of failure is important because they can highlight unauthorized or fraudulent access.

Network Requirements

Network security is as important as other securities. We cannot ignore the network security requirement. We need to consider the following issues:

- Is it necessary to encrypt data before transferring it to the data warehouse?
- Are there restrictions on which network routes the data can take?

These restrictions need to be considered carefully. Following are the points to remember:

- The process of encryption and decryption will increase overheads. It would require more processing power and processing time.
- The cost of encryption can be high if the system is already a loaded system because the encryption is borne by the source system.

Data Movement

There exist potential security implications while moving the data. Suppose we need to transfer some restricted data as a flat file to be loaded. When the data is loaded into the data warehouse, the following questions are raised:

- Where is the flat file stored?
- Who has access to that disk space?

If we talk about the backup of these flat files, the following questions are raised:

- Do you backup encrypted or decrypted versions?
- Do these backups need to be made to special tapes that are stored separately?
- Who has access to these tapes?

Some other forms of data movement like query result sets also need to be considered. The questions raised while creating the temporary table are as follows:

- Where is that temporary table to be held?
- How do you make such table visible?

We should avoid the accidental flouting of security restrictions. If a user with access to the restricted data can generate accessible temporary tables, data can be visible to non-authorized users. We can overcome this problem by having a separate temporary area for users with access to restricted data.

Documentation

The audit and security requirements need to be properly documented. This will be treated as a part of justification. This document can contain all the information gathered from:

- Data classification
- User classification
- Network requirements
- Data movement and storage requirements
- All auditable actions

Impact of Security on Design

Security affects the application code and the development timescales. Security affects the following area.

- Application development
- Database design
- Testing

Application Development

Security affects the overall application development and it also affects the design of the important components of the data warehouse such as load manager, warehouse manager, and query manager. The load manager may require checking code to filter record and place them in different locations. More transformation rules may also be required to hide certain data. Also there may be requirements of extra metadata to handle any extra objects.

To create and maintain extra views, the warehouse manager may require extra codes to enforce security. Extra checks may have to be coded into the data warehouse to prevent it from being fooled into moving data into a location where it should not be available. The query manager requires the changes to handle any access restrictions. The query manager will need to be aware of all extra views and aggregations.

Database design

The database layout is also affected because when security measures are implemented, there is an increase in the number of views and tables. Adding security increases the size of the database and hence increases the complexity of the database design and management. It will also add complexity to the backup management and recovery plan.

Testing

Testing the data warehouse is a complex and lengthy process. Adding security to the data warehouse also affects the testing time complexity. It affects the testing in the following two ways:

- It will increase the time required for integration and system testing.
- There is added functionality to be tested which will increase the size of the testing suite.

DATA WAREHOUSING - BACKUP

A data warehouse is a complex system and it contains a huge volume of data. Therefore it is important to back up all the data so that it becomes available for recovery in future as per requirement. In this chapter, we will discuss the issues in designing the backup strategy.

Backup Terminologies

Before proceeding further, you should know some of the backup terminologies discussed below.

- **Complete backup** - It backs up the entire database at the same time. This backup includes all the database files, control files, and journal files.
- **Partial backup** - As the name suggests, it does not create a complete backup of the database. Partial backup is very useful in large databases because they allow a strategy whereby various parts of the database are backed up in a round-robin fashion on a day-to-day basis, so that the whole database is backed up effectively once a week.
- **Cold backup** - Cold backup is taken while the database is completely shut down. In multi-instance environment, all the instances should be shut down.
- **Hot backup** - Hot backup is taken when the database engine is up and running. The requirements of hot backup varies from RDBMS to RDBMS.
- **Online backup** - It is quite similar to hot backup.

Hardware Backup

It is important to decide which hardware to use for the backup. The speed of processing the backup and restore depends on the hardware being used, how the hardware is connected, bandwidth of the network, backup software, and the speed of server's I/O system. Here we will discuss some of the hardware choices that are available and their pros and cons. These choices are as follows:

- Tape Technology
- Disk Backups

Tape Technology

The tape choice can be categorized as follows:

- Tape media
- Standalone tape drives
- Tape stackers
- Tape silos

Tape Media

There exists several varieties of tape media. Some tape media standards are listed in the table below:

Tape Media	Capacity	I/O rates
DLT	40 GB	3 MB/s
3490e	1.6 GB	3 MB/s
8 mm	14 GB	1 MB/s

Other factors that need to be considered are as follows:

- Reliability of the tape medium
- Cost of tape medium per unit
- Scalability
- Cost of upgrades to tape system
- Cost of tape medium per unit
- Shelf life of tape medium

Standalone tape drives

The tape drives can be connected in the following ways:

- Direct to the server
- As network available devices
- Remotely to other machine

There could be issues in connecting the tape drives to a data warehouse.

- Consider the server is a 48node MPP machine. We do not know the node to connect the tape drive and we do not know how to spread them over the server nodes to get the optimal performance with least disruption of the server and least internal I/O latency.
- Connecting the tape drive as a network available device requires the network to be up to the

job of the huge data transfer rates. Make sure that sufficient bandwidth is available during the time you require it.

- Connecting the tape drives remotely also require high bandwidth.

Tape Stackers

The method of loading multiple tapes into a single tape drive is known as tape stackers. The stacker dismounts the current tape when it has finished with it and loads the next tape, hence only one tape is available at a time to be accessed. The price and the capabilities may vary, but the common ability is that they can perform unattended backups.

Tape Silos

Tape silos provide large store capacities. Tape silos can store and manage thousands of tapes. They can integrate multiple tape drives. They have the software and hardware to label and store the tapes they store. It is very common for the silo to be connected remotely over a network or a dedicated link. We should ensure that the bandwidth of the connection is up to the job.

Disk Backups

Methods of disk backups are:

- Disk-to-disk backups
- Mirror breaking

These methods are used in the OLTP system. These methods minimize the database downtime and maximize the availability.

Disk-to-disk backups

Here backup is taken on the disk rather on the tape. Disk-to-disk backups are done for the following reasons:

- Speed of initial backups
- Speed of restore

Backing up the data from disk to disk is much faster than to the tape. However it is the intermediate step of backup. Later the data is backed up on the tape. The other advantage of disk-to-disk backups is that it gives you an online copy of the latest backup.

Mirror Breaking

The idea is to have disks mirrored for resilience during the working day. When backup is required, one of the mirror sets can be broken out. This technique is a variant of disk-to-disk backups.

Note: The database may need to be shutdown to guarantee consistency of the backup.

Optical Jukeboxes

Optical jukeboxes allow the data to be stored near line. This technique allows a large number of optical disks to be managed in the same way as a tape stacker or a tape silo. The drawback of this technique is that it has slow write speed than disks. But the optical media provides long-life and reliability that makes them a good choice of medium for archiving.

Software Backups

There are software tools available that help in the backup process. These software tools come as a package. These tools not only take backup, they can effectively manage and control the backup strategies. There are many software packages available in the market. Some of them are listed in the following table:

Package Name	Vendor

Networker	Legato
ADSM	IBM
Epoch	Epoch Systems
Omniback II	HP
Alexandria	Sequent

Criteria for Choosing Software Packages

The criteria for choosing the best software package are listed below:

- How scalable is the product as tape drives are added?
- Does the package have client-server option, or must it run on the database server itself?
- Will it work in cluster and MPP environments?
- What degree of parallelism is required?
- What platforms are supported by the package?
- Does the package support easy access to information about tape contents?
- Is the package database aware?
- What tape drive and tape media are supported by the package?

DATA WAREHOUSING - TUNING

A data warehouse keeps evolving and it is unpredictable what query the user is going to post in the future. Therefore it becomes more difficult to tune a data warehouse system. In this chapter, we will discuss how to tune the different aspects of a data warehouse such as performance, data load, queries, etc.

Difficulties in Data Warehouse Tuning

Tuning a data warehouse is a difficult procedure due to following reasons:

- Data warehouse is dynamic; it never remains constant.
- It is very difficult to predict what query the user is going to post in the future.
- Business requirements change with time.
- Users and their profiles keep changing.
- The user can switch from one group to another.
- The data load on the warehouse also changes with time.

Note: It is very important to have a complete knowledge of data warehouse.

Performance Assessment

Here is a list of objective measures of performance:

- Average query response time
- Scan rates
- Time used per day query
- Memory usage per process
- I/O throughput rates

Following are the points to remember.

- It is necessary to specify the measures in service level agreement SLA.
- It is of no use trying to tune response time, if they are already better than those required.
- It is essential to have realistic expectations while making performance assessment.
- It is also essential that the users have feasible expectations.
- To hide the complexity of the system from the user, aggregations and views should be used.
- It is also possible that the user can write a query you had not tuned for.

Data Load Tuning

Data load is a critical part of overnight processing. Nothing else can run until data load is complete. This is the entry point into the system.

Note: If there is a delay in transferring the data, or in arrival of data then the entire system is affected badly. Therefore it is very important to tune the data load first.

There are various approaches of tuning data load that are discussed below:

- The very common approach is to insert data using the **SQL Layer**. In this approach, normal checks and constraints need to be performed. When the data is inserted into the table, the code will run to check for enough space to insert the data. If sufficient space is not available, then more space may have to be allocated to these tables. These checks take time to perform and are costly to CPU.
- The second approach is to bypass all these checks and constraints and place the data directly into the preformatted blocks. These blocks are later written to the database. It is faster than the first approach, but it can work only with whole blocks of data. This can lead to some space wastage.
- The third approach is that while loading the data into the table that already contains the table, we can maintain indexes.
- The fourth approach says that to load the data in tables that already contain data, **drop the indexes & recreate them** when the data load is complete. The choice between the third and the fourth approach depends on how much data is already loaded and how many indexes need to be rebuilt.

Integrity Checks

Integrity checking highly affects the performance of the load. Following are the points to remember.

- Integrity checks need to be limited because they require heavy processing power.
- Integrity checks should be applied on the source system to avoid performance degrade of data load.

Tuning Queries

We have two kinds of queries in data warehouse:

- Fixed queries
- Ad hoc queries

Fixed Queries

Fixed queries are well defined. Following are the examples of fixed queries:

- regular reports

- Canned queries
- Common aggregations

Tuning the fixed queries in a data warehouse is same as in a relational database system. The only difference is that the amount of data to be queried may be different. It is good to store the most successful execution plan while testing fixed queries. Storing these executing plan will allow us to spot changing data size and data skew, as it will cause the execution plan to change.

Note: We cannot do more on fact table but while dealing with dimension tables or the aggregations, the usual collection of SQL tweaking, storage mechanism, and access methods can be used to tune these queries.

Ad hoc Queries

To understand ad hoc queries, it is important to know the ad hoc users of the data warehouse. For each user or group of users, you need to know the following:

- The number of users in the group
- Whether they use ad hoc queries at regular intervals of time
- Whether they use ad hoc queries frequently
- Whether they use ad hoc queries occasionally at unknown intervals.
- The maximum size of query they tend to run
- The average size of query they tend to run
- Whether they require drill-down access to the base data
- The elapsed login time per day
- The peak time of daily usage
- The number of queries they run per peak hour

Points to Note

- It is important to track the user's profiles and identify the queries that are run on a regular basis.
- It is also important that the tuning performed does not affect the performance.
- Identify similar and ad hoc queries that are frequently run.
- If these queries are identified, then the database will change and new indexes can be added for those queries.
- If these queries are identified, then new aggregations can be created specifically for those queries that would result in their efficient execution.

DATA WAREHOUSING - TESTING

Testing is very important for data warehouse systems to make them work correctly and efficiently. There are three basic levels of testing performed on a data warehouse:

- Unit testing
- Integration testing
- System testing

Unit Testing

- In unit testing, each component is separately tested.
- Each module, i.e., procedure, program, SQL Script, Unix shell is tested.

- This test is performed by the developer.

Integration Testing

- In integration testing, the various modules of the application are brought together and then tested against the number of inputs.
- It is performed to test whether the various components do well after integration.

System Testing

- In system testing, the whole data warehouse application is tested together.
- The purpose of system testing is to check whether the entire system works correctly together or not.
- System testing is performed by the testing team.
- Since the size of the whole data warehouse is very large, it is usually possible to perform minimal system testing before the test plan can be enacted.

Test Schedule

First of all, the test schedule is created in the process of developing the test plan. In this schedule, we predict the estimated time required for the testing of the entire data warehouse system.

There are different methodologies available to create a test schedule, but none of them are perfect because the data warehouse is very complex and large. Also the data warehouse system is evolving in nature. One may face the following issues while creating a test schedule:

- A simple problem may have a large size of query that can take a day or more to complete, i.e., the query does not complete in a desired time scale.
- There may be hardware failures such as losing a disk or human errors such as accidentally deleting a table or overwriting a large table.

Note: Due to the above-mentioned difficulties, it is recommended to always double the amount of time you would normally allow for testing.

Testing Backup Recovery

Testing the backup recovery strategy is extremely important. Here is the list of scenarios for which this testing is needed:

- Media failure
- Loss or damage of table space or data file
- Loss or damage of redo log file
- Loss or damage of control file
- Instance failure
- Loss or damage of archive file
- Loss or damage of table
- Failure during data failure

Testing Operational Environment

There are a number of aspects that need to be tested. These aspects are listed below.

- **Security** - A separate security document is required for security testing. This document contains a list of disallowed operations and devising tests for each.
- **Scheduler** - Scheduling software is required to control the daily operations of a data

warehouse. It needs to be tested during system testing. The scheduling software requires an interface with the data warehouse, which will need the scheduler to control overnight processing and the management of aggregations.

- **Disk Configuration.** - Disk configuration also needs to be tested to identify I/O bottlenecks. The test should be performed with multiple times with different settings.
- **Management Tools.** - It is required to test all the management tools during system testing. Here is the list of tools that need to be tested.
    - Event manager
    - System manager
    - Database manager
    - Configuration manager
    - Backup recovery manager

Testing the Database

The database is tested in the following three ways:

- **Testing the database manager and monitoring tools** - To test the database manager and the monitoring tools, they should be used in the creation, running, and management of test database.
- **Testing database features** - Here is the list of features that we have to test:
    - Querying in parallel
    - Create index in parallel
    - Data load in parallel
- **Testing database performance** - Query execution plays a very important role in data warehouse performance measures. There are sets of fixed queries that need to be run regularly and they should be tested. To test ad hoc queries, one should go through the user requirement document and understand the business completely. Take time to test the most awkward queries that the business is likely to ask against different index and aggregation strategies.

Testing the Application

- All the managers should be integrated correctly and work in order to ensure that the end-to-end load, index, aggregate and queries work as per the expectations.
- Each function of each manager should work correctly
- It is also necessary to test the application over a period of time.
- Week end and month-end tasks should also be tested.

Logistic of the Test

The aim of system test is to test all of the following areas.

- Scheduling software
- Day-to-day operational procedures
- Backup recovery strategy
- Management and scheduling tools
- Overnight processing
- Query performance

Note: The most important point is to test the scalability. Failure to do so will leave us a system

design that does not work when the system grows.

DATA WAREHOUSING - FUTURE ASPECTS

Following are the future aspects of data warehousing.

- As we have seen that the size of the open database has grown approximately double its magnitude in the last few years, it shows the significant value that it contains.
- As the size of the databases grow, the estimates of what constitutes a very large database continues to grow.
- The hardware and software that are available today do not allow to keep a large amount of data online. For example, a Telco call record requires 10TB of data to be kept online, which is just a size of one month's record. If it requires to keep records of sales, marketing customer, employees, etc., then the size will be more than 100 TB.
- The record contains textual information and some multimedia data. Multimedia data cannot be easily manipulated as text data. Searching the multimedia data is not an easy task, whereas textual information can be retrieved by the relational software available today.
- Apart from size planning, it is complex to build and run data warehouse systems that are ever increasing in size. As the number of users increases, the size of the data warehouse also increases. These users will also require to access the system.
- With the growth of the Internet, there is a requirement of users to access data online.

Hence the future shape of data warehouse will be very different from what is being created today.

DATA WAREHOUSING - INTERVIEW QUESTIONS

Dear readers, these **Data Warehousing Interview Questions** have been designed especially to get you acquainted with the nature of questions you may encounter during your interview for the subject of **Data Warehousing**.

Q: Define data warehouse?

A : Data warehouse is a subject oriented, integrated, time-variant, and nonvolatile collection of data that supports management's decision-making process.

Q: What does subject-oriented data warehouse signify?

A : Subject oriented signifies that the data warehouse stores the information around a particular subject such as product, customer, sales, etc.

Q: List any five applications of data warehouse.

A : Some applications include financial services, banking services, customer goods, retail sectors, controlled manufacturing.

Q: What do OLAP and OLTP stand for?

A : OLAP is an acronym for **Online Analytical Processing** and OLTP is an acronym of Online Transactional Processing.

Q: What is the very basic difference between data warehouse and operational databases?

A : A data warehouse contains historical information that is made available for analysis of the business whereas an operational database contains current information that is required to run the business.

Q: List the Schema that a data warehouse system can implements.

A : A data Warehouse can implement star schema, snowflake schema, and fact constellation schema.

Q: What is Data Warehousing?

A : Data Warehousing is the process of constructing and using the data warehouse.

Q: List the process that are involved in Data Warehousing.

A : Data Warehousing involves data cleaning, data integration and data consolidations.

Q: List the functions of data warehouse tools and utilities.

A : The functions performed by Data warehouse tool and utilities are Data Extraction, Data Cleaning, Data Transformation, Data Loading and Refreshing.

Q: What do you mean by Data Extraction?

A : Data extraction means gathering data from multiple heterogeneous sources.

Q: Define metadata?

A : Metadata is simply defined as data about data. In other words, we can say that metadata is the summarized data that leads us to the detailed data.

Q: What does Metadata Respiratory contain?

A : Metadata respiratory contains definition of data warehouse, business metadata, operational metadata, data for mapping from operational environment to data warehouse, and the algorithms for summarization.

Q: How does a Data Cube help?

A : Data cube helps us to represent the data in multiple dimensions. The data cube is defined by dimensions and facts.

Q: Define dimension?

A : The dimensions are the entities with respect to which an enterprise keeps the records.

Q: Explain data mart.

A : Data mart contains the subset of organization-wide data. This subset of data is valuable to specific groups of an organization. In other words, we can say that a data mart contains data specific to a particular group.

Q: What is Virtual Warehouse?

A : The view over an operational data warehouse is known as virtual warehouse.

Q: List the phases involved in the data warehouse delivery process.

A : The stages are IT strategy, Education, Business Case Analysis, technical Blueprint, Build the version, History Load, Ad hoc query, Requirement Evolution, Automation, and Extending Scope.

Q: Define load manager.

A : A load manager performs the operations required to extract and load the process. The size and complexity of load manager varies between specific solutions from data warehouse to data warehouse.

Q: Define the functions of a load manager.

A : A load manager extracts data from the source system. Fast load the extracted data into temporary data store. Perform simple transformations into structure similar to the one in the data warehouse.

Q: Define a warehouse manager.

A : Warehouse manager is responsible for the warehouse management process. The warehouse manager consist of third party system software, C programs and shell scripts. The size and

complexity of warehouse manager varies between specific solutions.

Q: Define the functions of a warehouse manager.

A : The warehouse manager performs consistency and referential integrity checks, creates the indexes, business views, partition views against the base data, transforms and merge the source data into the temporary store into the published data warehouse, backs up the data in the data warehouse, and archives the data that has reached the end of its captured life.

Q: What is Summary Information?

A : Summary Information is the area in data warehouse where the predefined aggregations are kept.

Q: What does the Query Manager responsible for?

A : Query Manager is responsible for directing the queries to the suitable tables.

Q: List the types of OLAP server

A : There are four types of OLAP servers, namely Relational OLAP, Multidimensional OLAP, Hybrid OLAP, and Specialized SQL Servers.

Q: Which one is faster, Multidimensional OLAP or Relational OLAP?

A : Multidimensional OLAP is faster than Relational OLAP.

Q: List the functions performed by OLAP.

A : OLAP performs functions such as roll-up, drill-down, slice, dice, and pivot.

Q: How many dimensions are selected in Slice operation?

A : Only one dimension is selected for the slice operation.

Q: How many dimensions are selected in dice operation?

A : For dice operation two or more dimensions are selected for a given cube.

Q: How many fact tables are there in a star schema?

A : There is only one fact table in a star Schema.

Q: What is Normalization?

A : Normalization splits up the data into additional tables.

Q: Out of star schema and snowflake schema, whose dimension table is normalized?

A : Snowflake schema uses the concept of normalization.

Q: What is the benefit of normalization?

A : Normalization helps in reducing data redundancy.

Q: Which language is used for defining Schema Definition?

A : Data Mining Query Language DMQL is used for Schema Definition.

Q: What language is the base of DMQL?

A : DMQL is based on Structured Query Language SQL.

Q: What are the reasons for partitioning?

A : Partitioning is done for various reasons such as easy management, to assist backup recovery, to enhance performance.

Q: What kind of costs are involved in Data Marting?

A : Data Marting involves hardware & software cost, network access cost, and time cost.