**Benefits and Limitations of Using Spreadsheets for Data Analysis**

**Benefits**:

1. **User-Friendly Interface**: Spreadsheets provide an intuitive way to organize data in a table format, making it easy to view and analyze the top results, such as the first 100 rows.
2. **Data Comparison**: Functions like VLOOKUP and XLOOKUP allow for quick comparisons between fields, facilitating simple data matching tasks.
3. **SQL Table Emulation**: Custom formulas can simulate SQL table functionality within Excel, enabling relational data management on a small scale.
4. **Versatile Tools**: Features like pivot tables, charts, and formulas support basic data summarization, visualization, and calculation tasks.
5. **Ideal for Small Datasets**: Spreadsheets handle datasets of up to 20,000 records effectively, making them a great choice for light, straightforward analyses.
6. **Accessibility**: Widely available and easy to learn, spreadsheets are accessible for beginners and professionals alike.

**Limitations**:

1. **Lack of Reproducibility**: Actions like filtering, sorting, or applying formulas lack a built-in log, making it difficult to retrace steps or reproduce analysis accurately.
2. **Performance Issues**: Spreadsheets struggle with large datasets; handling records exceeding 100,000 rows may result in crashes or slow performance.
3. **Limited Scalability**: They are not suitable for complex analyses, such as advanced statistical modeling or machine learning, due to limited functionality and computational power.
4. **Error-Prone**: Manual data entry and formula adjustments increase the risk of errors, especially for large or intricate datasets.
5. **Collaboration Challenges**: Multi-user editing in real-time can be cumbersome, leading to version control issues unless using advanced tools like Google Sheets.
6. **Automation Constraints**: Spreadsheets lack the automation and repeatability offered by programming languages like Python or R, limiting efficiency in iterative tasks.
7. **Data Integration**: Combining data from multiple sources is possible but can become tedious without tools like Power Query or dedicated software.

**2. When should we use the Waterfall and funnel chart?**

**Waterfall-**

- Shows how values increase/decrease over time

Used in: -

Budget Tracking

**Funnel: -**

Shows drop-off across process stages

Used in: -

Customer Journey

* 1. **When should we use donut chart and pie chart?**

**When to Use Pie Chart: -**

To show the relative proportion of different categories in a dataset.

When there are few categories (ideally less than 6).

When exact percentage comparisons are important.

When labels and values are easy to distinguish.

**Donut Chart**

**When to Use?**

When you want a modern, cleaner look with a center space.

To show proportions while allowing space in the center for extra labels or text.

When comparing multiple datasets (e.g., side-by-side donut charts).

For dashboard visualizations (less cluttered than pie charts).

**1. General Shortcuts**

| **Shortcut** | **Function** |
| --- | --- |
| Ctrl + N | Create a new workbook |
| Ctrl + O | Open an existing workbook |
| Ctrl + S | Save the workbook |
| Ctrl + P | Print the worksheet |
| Ctrl + Z | Undo last action |
| Ctrl + Y | Redo last undone action |
| Ctrl + C | Copy selected cells |
| Ctrl + X | Cut selected cells |
| Ctrl + V | Paste copied/cut cells |
| Ctrl + F | Find in the worksheet |
| Ctrl + H | Find and replace |

**2. Navigation Shortcuts**

| **Shortcut** | **Function** |
| --- | --- |
| Arrow Keys | Move between cells |
| Ctrl + Arrow Key | Jump to the edge of the data range |
| Ctrl + Home | Go to the **A1** cell |
| Ctrl + End | Go to the last used cell |
| Page Up / Page Down | Scroll up/down one screen |
| Alt + Tab | Switch between open workbooks |

**3. Cell Editing Shortcuts**

| **Shortcut** | **Function** |
| --- | --- |
| F2 | Edit selected cell |
| Alt + Enter | Insert a line break inside a cell |
| Ctrl + D | Copy value from the cell above |
| Ctrl + R | Copy value from the left cell |
| Ctrl + Shift + "+" | Insert a new row or column |
| Ctrl + "-" | Delete a row or column |

**4. Formatting Shortcuts**

| **Shortcut** | **Function** |
| --- | --- |
| Ctrl + B | Bold text |
| Ctrl + I | Italic text |
| Ctrl + U | Underline text |
| Ctrl + 1 | Open Format Cells dialog box |
| Ctrl + Shift + $ | Apply currency format |
| Ctrl + Shift + % | Apply percentage format |
| Ctrl + Shift + # | Apply date format |

**5. Selecting Data**

| **Shortcut** | **Function** |
| --- | --- |
| Ctrl + A | Select the entire worksheet |
| Ctrl + Shift + Arrow Key | Select data in a direction |
| Ctrl + Space | Select entire column |
| Shift + Space | Select entire row |

**6. Working with Formulas**

| **Shortcut** | **Function** |
| --- | --- |
| = (Equals sign) | Start a formula |
| Alt + = | AutoSum (Σ) selected cells |
| Ctrl + Shift + Enter | Enter an array formula |
| F4 | Repeat last action / Toggle absolute & relative references in formulas |

**7. Pivot Table & Charts**

| **Shortcut** | **Function** |
| --- | --- |
| Alt + N + V | Insert a Pivot Table |
| Alt + F1 | Create a chart from selected data |
| F11 | Create a chart in a new sheet |

**What is GitHub?**

GitHub is a **web-based platform** for **version control, collaboration, and code hosting**. It is built on **Git**, a distributed version control system that helps developers manage code efficiently.

**Why Should We Use GitHub?**

🔹 **1. Version Control**

* Tracks changes to your code over time.
* Allows you to revert to previous versions if needed.
* Helps in debugging by seeing historical changes.

**What is Power BI?**

Power BI is a **business intelligence (BI) tool** developed by Microsoft that helps users **analyze data, create reports, and generate interactive visualizations**. It enables users to connect to various data sources, transform raw data, and gain insights through dashboards and reports.

**Why Should We Use Power BI?**

**🔹 1. Data Connectivity**

* Connects to **Excel, SQL Server, Oracle, AWS, Azure, Google Analytics, APIs, and more**.
* Supports **real-time data** updates.

**🔹 2. Data Transformation & Modeling**

* **Power Query** helps in **data cleaning, merging, and transformation**.
* **DAX (Data Analysis Expressions)** allows advanced calculations and custom measures.

**🔹 3. Interactive Dashboards & Reports**

* Drag-and-drop interface for creating **dynamic charts, maps, and visuals**.
* Filters, slicers, and drill-through capabilities for deep insights.

**🔹 4. Automation & Scheduling**

* Automate report refreshes and scheduling.
* No manual data updates required.

**🔹 5. Cloud & On-Premise Access**

* Publish reports on **Power BI Cloud Service** for web access.
* Use **Power BI Desktop** for offline report building.
* Integrate with **Power BI Mobile App** for on-the-go analytics.

**🔹 6. AI-Powered Insights**

* Uses **AI-driven analytics** for trend detection and forecasting.
* Includes **natural language query (Q&A)** for quick insights.

**🔹 7. Collaboration & Sharing**

* Share reports securely across teams.
* Integrates with **Microsoft Teams, Excel, and SharePoint**.

**Who Uses Power BI?**

✅ **Data Analysts** – To analyze trends and patterns.  
✅ **Business Executives** – To track KPIs and company performance.  
✅ **Finance Teams** – To monitor budgets and financial reports.  
✅ **Sales & Marketing Teams** – To track customer insights and revenue trends

**What is Python?**

Python is a **high-level, interpreted programming language** known for its **simplicity, readability, and flexibility**. It is widely used in **web development, automation, AI, machine learning, and data analysis**.

**Why Use Python for Data Analysis?**

**🔹 1. Easy to Learn & Use**

* Simple **syntax**, making it beginner-friendly.
* Readable and clean code for data manipulation.

**🔹 2. Powerful Libraries for Data Analysis**

Python has **rich libraries** for handling, processing, and visualizing data:

| **Library** | **Purpose** |
| --- | --- |
| **pandas** | Data manipulation & analysis (like Excel for Python) |
| **NumPy** | Numerical computing (arrays, matrices, math functions) |
| **Matplotlib & Seaborn** | Data visualization (charts, graphs, heatmaps) |
| **Scikit-learn** | Machine learning & predictive analytics |
| **Statsmodels** | Statistical analysis & hypothesis testing |

**🔹 3. Handles Large Datasets Efficiently**

* Can process **millions of rows** (faster than Excel).
* Works well with **structured and unstructured data** (CSV, JSON, databases, etc.).

**🔹 4. Integration with Databases & Big Data**

* Connects with **SQL, MongoDB, Hadoop, Spark** for large-scale data processing.

**🔹 5. Automation & Reproducibility**

* Automate repetitive tasks like **data cleaning, transformation, and reporting**.
* Scripts can be reused and scheduled for **automated reporting**.

**🔹 6. Machine Learning & AI Capabilities**

* Python integrates with **TensorFlow, PyTorch, and Scikit-learn** for predictive modeling.
* Used in **data science, AI, and deep learning applications**.

**Who Uses Python for Data Analysis?**

✅ **Data Analysts & Scientists** – For exploring and visualizing data.  
✅ **Business Intelligence Teams** – To extract insights from large datasets.  
✅ **Financial Analysts** – To model financial forecasts.  
✅ **Engineers & Researchers** – To analyze experimental data.

**What is SQL?**

SQL (**Structured Query Language**) is a programming language used to **manage and query relational databases**. It allows users to **store, retrieve, manipulate, and analyze data** efficiently.

**Why Should We Use SQL?**

**🔹 1. Efficient Data Retrieval**

* Quickly fetch specific data using **SELECT queries**.
* Filter and sort data using **WHERE, ORDER BY, and GROUP BY**.

**🔹 2. Handles Large Datasets**

* Can manage **millions of records** efficiently (faster than Excel or CSV files).
* Optimized for structured data storage and retrieval.
* **Who Uses SQL?**
* ✅ **Data Analysts & Scientists** – To extract insights from databases.  
  ✅ **Software Engineers** – To store and manage application data.  
  ✅ **Business Intelligence Teams** – To generate reports for decision-making.  
  ✅ **Finance & Marketing Teams** – To track revenue, customer trends, and transactions.

**When to Use Python Over SQL?**

Python and SQL both play key roles in **data analysis and engineering**, but they serve different purposes. **SQL is best for querying and managing structured data**, while **Python is more powerful for complex processing, automation, and machine learning**.

**✅ Use Python When:**

**🔹 1. Advanced Data Manipulation & Cleaning**

* SQL is great for basic filtering, but Python (with **pandas**) is better for:  
  ✅ Handling missing values (**fillna, dropna**)  
  ✅ Complex transformations (e.g., regex-based string cleaning)  
  ✅ Handling **JSON, XML, or nested data** that SQL struggles with

**Example:**  
✅ Removing outliers from a dataset → Easier in Python than SQL

**🔹 2. Statistical Analysis & Machine Learning**

* SQL can **aggregate and summarize** data, but it can't do:  
  ✅ **Regression analysis** (e.g., linear regression)  
  ✅ **Predictive modeling** (e.g., machine learning)  
  ✅ **Time series forecasting**

**Example:**  
✅ Predicting sales trends based on historical data → Best in Python (with **scikit-learn**)

**🔹 3. Data Visualization**

* SQL **retrieves data**, but **it doesn’t visualize it**.
* Python (with **Matplotlib, Seaborn, Plotly**) helps create:  
  ✅ Interactive charts  
  ✅ Heatmaps & correlations  
  ✅ Geospatial maps

**Example:**  
✅ Creating a customer retention dashboard → Easier in Python than SQL

**🔹 4. Automation & Data Pipelines**

* SQL runs **queries manually** (or via stored procedures), but Python can:  
  ✅ Automate data extraction & transformations (**ETL pipelines**)  
  ✅ Schedule tasks (**with cron jobs, Apache Airflow**)  
  ✅ Handle API calls & real-time data ingestion

**Example:**  
✅ Automating daily stock market data updates → Best in Python

**🔹 5. Working with Multiple Data Sources**

* SQL works **only within relational databases**, but Python can:  
  ✅ Merge **CSV, Excel, JSON, APIs, and NoSQL (MongoDB, Firebase, etc.)**  
  ✅ Perform complex joins across **multiple data sources**

**Example:**  
✅ Merging customer data from SQL + product data from a NoSQL database → Best in Python

**✅ Use SQL When:**

🔹 Simple data retrieval (e.g., “Give me the top 10 customers”)  
🔹 Filtering, sorting, and grouping large datasets quickly  
🔹 Joining multiple tables efficiently  
🔹 Handling structured, relational data

**🚀 When to Use Both?**

1️⃣ **SQL** → Extract & filter data from the database  
2️⃣ **Python** → Perform advanced analysis & visualization

**Week 1**

**Basic Excel Questions**

**1. What are the most commonly used Excel functions for data analysis?**  
**Answer:**

* VLOOKUP / XLOOKUP – Lookup values from another table.
* INDEX-MATCH – More flexible than VLOOKUP for searching data.
* SUMIFS / COUNTIFS – Aggregate data based on multiple conditions.
* IF, IFS, IFERROR – Conditional logic functions.
* TEXT functions – LEFT, RIGHT, MID, LEN, TRIM, CONCATENATE (or TEXTJOIN).
* DATE functions – TODAY(), EOMONTH(), YEAR(), TEXT(date, "dd-mmm-yyyy").
* Pivot Tables – Summarize and analyze large datasets quickly.

**Intermediate Excel Questions**

**2. What is the difference between a Pivot Table and a normal table?**  
**Answer:**  
A **Pivot Table** dynamically summarizes, filters, and aggregates data without modifying the raw dataset. A **normal table** stores data in a structured way but lacks built-in analysis tools.

**3. How would you remove duplicate values from a dataset?**  
**Answer:**  
Use **"Remove Duplicates"** under the **Data** tab or use the UNIQUE() function (available in newer Excel versions).

**4. Explain how Power Query is useful for data analysis.**  
**Answer:**  
Power Query helps in:

* **Extracting** data from various sources (CSV, databases, APIs).
* **Transforming** data (cleaning, merging, pivoting/unpivoting).
* **Loading** data into Excel or Power BI.
* Automating **repeatable** data-cleaning tasks.

**Advanced Excel Questions**

**5. What is the difference between absolute, relative, and mixed cell references?**  
**Answer:**

* **Relative (A1)** – Changes when copied to another cell.
* **Absolute ($A$1)** – Stays fixed when copied.
* **Mixed ($A1 or A$1)** – Partially fixed (column or row stays constant).

**6. How would you handle large datasets efficiently in Excel?**  
**Answer:**

* Use **Tables** (Ctrl + T) for structured data.
* Use **Pivot Tables** instead of complex formulas.
* Minimize volatile functions like INDIRECT() and OFFSET().
* Avoid using full-column references (A:A), use specific ranges (A1:A1000).
* Turn off automatic calculation (File → Options → Formulas → Manual Calculation).

**7. How do you perform a dynamic lookup in Excel?**  
**Answer:**  
Use XLOOKUP (modern Excel) or INDEX-MATCH. Example:

excel

CopyEdit

=INDEX(B2:B10, MATCH(1001, A2:A10, 0))

(Looks up **1001** in Column A and returns corresponding value from Column B.)

**Scenario-Based Questions**

**8. Suppose you have sales data in Excel. How would you find total revenue by region?**  
**Answer:**

* Use **Pivot Table**: Drag "Region" to **Rows** and "Revenue" to **Values (Sum)**.
* Use SUMIFS():

excel

CopyEdit

=SUMIFS(Revenue\_Column, Region\_Column, "NSW")

(Gets revenue only for **NSW** region.)

**9. How would you visualize trends in sales over time?**  
**Answer:**

* **Line Chart** for time-series trends.
* **Bar/Column Chart** for comparisons.
* **Conditional Formatting** to highlight trends (e.g., green for increasing, red for decreasing).

**10. How do you handle missing data in an Excel dataset?**  
**Answer:**

* Use IFERROR() or IFNA() to replace errors.
* Use =IF(A2="", "Missing Data", A2).
* Use Power Query to filter or replace missing values.

# DAY 3

**Data Cleaning & Preparation Questions**

1. **What data quality issues do you see in this portfolio dataset? How would you clean them?**
   * Example answer: "I noticed that the Sale Price column has a property sold for **$1**, which is likely an error or missing data. I would check if it’s a data entry mistake or an outlier."
2. **How would you handle missing or inconsistent values in this portfolio dataset?**
   * Discuss techniques like **imputation, filtering, or flagging missing data**.
3. **How would you standardize suburb names in case they have inconsistent formats?**
   * Example: Convert "AUBURN NSW 2144" to just "AUBURN" using text functions or Power Query.
4. **How would you detect duplicate property records?**
   * Example approach: Identify duplicates based on Address, Sale Date, and Sale Price.

### **Descriptive Statistics**

1. **What are the main types of descriptive statistics?**
   * **Answer:** Measures of central tendency (**mean, median, mode**) and measures of dispersion (**range, variance, standard deviation**).
2. **What is the difference between mean, median, and mode? When would you use each?**
   * **Example answer:** The **mean** is the average, the **median** is the middle value, and the **mode** is the most frequent value.
   * **Use case:** Median is better when the data has outliers (e.g., property prices).
3. **What does standard deviation tell us about a dataset?**
   * **Answer:** It measures how much data varies from the mean. A high standard deviation means more variability.
4. **How would you identify outliers in a dataset?**
   * **Answer:**
     + **Using the IQR method**: An outlier is a value **outside Q1 - 1.5 × IQR or Q3 + 1.5 × IQR**.
     + **Using Z-score**: If Z-score > **3** or < **-3**, it’s an outlier.
5. **What is the difference between population and sample in statistics?**
   * **Answer:** A **population** includes all possible data points, while a **sample** is a subset of the population used for analysis.

**📌 Correlation & Regression**

1. **What is correlation, and how is it different from causation?**

* **Answer:** Correlation measures the strength and direction of a relationship between two variables, while causation means one variable directly affects another.

1. **What is the difference between Pearson and Spearman correlation?**

* **Pearson Correlation:** Measures **linear relationships** between variables.
* **Spearman Correlation:** Measures **monotonic relationships** (not necessarily linear).

1. **What is linear regression, and when would you use it?**

* **Answer:** Linear regression predicts a dependent variable using an independent variable (e.g., predicting house prices based on number of bedrooms).

1. **What is R-squared in regression analysis?**

* **Answer:** It represents the proportion of variance in the dependent variable explained by the independent variable(s). A higher **R²** means a better fit.

1. **What assumptions must hold for linear regression to work properly?**

* Linearity
* Independence of errors
* Normality of residuals
* Homoscedasticity (constant variance of errors)

**Week 3**

1. **What is dual, import and direct query mode in powerbi?**

In Power BI, data can be stored and retrieved using different storage modes:

1. Import Mode
2. Direct Query Mode
3. Dual Mode

These modes affect how data is loaded, refreshed, and queried in Power BI.

1. Import Mode

* Data is imported into Power BI and stored in memory (cache).
* Fast performance since queries are executed on the in-memory dataset.
* Requires manual or scheduled refreshes to update data.
* Ideal for small to medium datasets where performance is a priority.

✅ Best for: Fast reporting, small datasets, and when real-time data isn't required.

🔴 Limitation: Refresh needed to get updated data; might not work well for very large datasets.

2. Direct Query Mode

* Data remains in the source database; Power BI queries it live.
* No need for data refresh since queries run in real time.
* Performance depends on the database; slow for complex queries.
* Used when working with large datasets or real-time analytics.

✅ Best for: Large datasets, real-time data reporting, and when using enterprise databases like SQL Server, Snowflake, or Oracle.

🔴 Limitation: Slower performance compared to Import Mode; limited DAX and Power Query transformations.

3. Dual Mode

* A hybrid of Import and Direct Query mode—table can switch between both.
* Allows frequently used data to be cached (Import Mode) while keeping other parts dynamic (Direct Query).
* Improves performance while still enabling real-time updates for specific queries.

✅ Best for: Optimizing performance when some data is frequently used and other data needs to be live.

🔴 Limitation: Requires good data model design to balance between speed and real-time data access.

**Comparison Table**

| **Mode** | **Data Storage** | **Performance** | **Data Freshness** | **Best For** |
| --- | --- | --- | --- | --- |
| Import | In-memory (cache) | Fast | Requires refresh | Small/medium datasets |
| Direct Query | Source database | Slower | Real-time | Large datasets, real-time reporting |
| Dual | Both (Import + Direct Query) | Balanced | Some cached, some real-time | Mixed-use cases |

Which One Should You Use?

* Use Import Mode if your dataset is small and performance is critical.
* Use Direct Query Mode if you need real-time data updates and work with large datasets.
* Use Dual Mode when you need the best of both worlds—frequently used data cached, while keeping some tables dynamic.

1. **What is data modelling?**

Data modelling in Power BI is the process of structuring, relating, and optimizing data to create efficient, scalable, and easy-to-analyze reports. It defines how tables connect to each other using relationships, ensuring smooth filtering and aggregation.

1. **What is fact and dimension table?**

**Definition:**

A Fact Table contains quantitative data or measurable events that can be analysed and aggregated. It often stores the primary metrics or facts about a particular business process.

Characteristics:

* Contains numeric data (e.g., sales, profit, quantity, score).
* Stores transactions or events.
* Has foreign keys that link to dimension tables.
* Often large in size because it contains transaction data.

**Definition:**

A **Dimension Table** contains **descriptive or categorical data** that provides context to the facts. It helps in filtering, grouping, and categorizing data in reports.

**Characteristics:**

* **Contains descriptive attributes** (e.g., names, locations, dates, categories).
* **Helps filter or group the data** in analysis.
* Relatively smaller compared to fact tables.
* Contains **primary keys** that are linked to fact tables through **foreign keys**.

1. What is DAX in powerbi?

DAX (Data Analysis Expressions) is a formula language used in Microsoft Power BI, Excel, and other tools to define custom calculations and aggregations on data models. It allows you to create calculated columns, measures, and tables that help analyze your data more deeply.

1. **Difference between measure and calculated field in Powerbi?**

In Power BI DAX, both **measures** and **calculated columns** (sometimes referred to as calculated fields) are used to create new data points based on existing data, but they serve different purposes and behave differently.

**1. Measures:**

* **Definition**: A **measure** is a DAX expression used to calculate an aggregated value on the fly. It is calculated at query time based on the filters applied in the report or visualization.
* **Use Case**: Measures are ideal for calculations like sums, averages, counts, percentages, etc., that are displayed dynamically in visuals like charts or tables.
* **Calculation Scope**: Measures are not stored in the data model. Instead, they are calculated based on the current context of the report (e.g., filter, slicer, row context).
* **Example**:
  + Total Marks: Total Marks = SUM(StudentData[Marks])
  + This measure will sum the Marks for each context (for example, when filtering by a specific student or subject).
* **Performance**: Measures are typically more efficient as they are calculated at query time and don’t add new columns to the underlying data model.

**2. Calculated Columns (Calculated Field):**

* **Definition**: A **calculated column** is a DAX expression that creates a new column in the data model. It is calculated during data refresh and stored as part of the table.
* **Use Case**: Calculated columns are useful when you need to create a new data point for each row in a table, like categorizing data, creating flag indicators, or performing row-level operations.
* **Calculation Scope**: Calculated columns are evaluated when the data is loaded into the model, and they are stored in the table as new columns.
* **Example**:
  + Pass/Fail:

DAX

Pass/Fail = IF(StudentData[Marks] >= 50, "Pass", "Fail")

* + This calculated column will evaluate each student's marks row by row and categorize them as "Pass" or "Fail."
* **Performance**: Calculated columns are stored in the data model, which means they take up memory, and large datasets with many calculated columns can impact performance.

**Key Differences:**

| **Aspect** | **Measure** | **Calculated Column** |
| --- | --- | --- |
| **When is it calculated?** | On-the-fly (during query time) | During data refresh (stored in the table) |
| **Where is it stored?** | Not stored in the table, just calculated dynamically | Stored as part of the table in the model |
| **Calculation Context** | Depends on report filters, slicers, and row context | Row context (calculated for each row) |
| **Use Case** | Aggregations (sum, average, etc.), dynamic calculations | Row-level calculations or categorization |
| **Example** | Total Sales = SUM(Sales[Amount]) | Discounted Price = Sales[Price] \* 0.9 |

**Summary:**

* **Measures** are used for aggregated, dynamic calculations that change based on the context of the report.
* **Calculated Columns** are row-level calculations that create new columns in your table, and their values are stored in the data model.

1. **Types of relationships in powerbi modelling**?

In Power BI data modeling, relationships define how tables are connected to each other. The type of relationship you choose determines how data from multiple tables can be combined and queried together. There are several types of relationships in Power BI, each serving a different purpose for your data model.

**Types of Relationships in Power BI:**

1. **One-to-One (1:1) Relationship**:
   * **Definition**: A one-to-one relationship means that for each row in the first table, there is exactly one matching row in the second table.
   * **Use Case**: Rarely used in most data models. This relationship type is useful when you have two tables that store different attributes about the same entity, and each row in both tables corresponds directly to the same unique value.
   * **Example**: A Student table and a StudentDetails table where each student has one unique record in each table.
2. *One-to-Many (1: or 1:N) Relationship*\*:
   * **Definition**: In a one-to-many relationship, one row in the "one" table (the primary table) is related to many rows in the "many" table (the related table).
   * **Use Case**: This is the most common type of relationship. It is typically used in fact-dimension scenarios, where the "one" side is usually the dimension table, and the "many" side is the fact or transactional table.
   * **Example**: A Customer table (one side) and an Orders table (many side). Each customer can have many orders, but each order belongs to only one customer.
   * **Relationship**: The CustomerID in the Customer table is linked to the CustomerID in the Orders table.
3. **Many-to-One (\*:1 or N:1) Relationship**:
   * **Definition**: A many-to-one relationship is essentially the reverse of a one-to-many relationship, but it’s typically seen in the same context because it’s just the opposite direction of querying. In this case, multiple rows in the "many" table are related to one row in the "one" table.
   * **Use Case**: The "many" side typically holds transactional data, and the "one" side holds a master or reference data table.
   * **Example**: In a Sales table (many side) and a Region table (one side), many sales records can belong to a single region.
4. **Many-to-Many (*:*) Relationship**:
   * **Definition**: In a many-to-many relationship, rows from both tables can have multiple matching rows in the other table. This is a more complex relationship that requires careful modeling to avoid ambiguity in how data is aggregated.
   * **Use Case**: Used when both tables have multiple matching rows that you need to connect and analyze together. Often used with a bridge table or intermediate table to resolve the relationship.
   * **Example**: A Products table and a Customers table. A product can be purchased by many customers, and a customer can purchase many products. A bridge table (e.g., CustomerProduct table) is often used to resolve this many-to-many relationship.

**Relationship Cardinality in Power BI:**

* **Single Relationship**: Each pair of related tables can have only one active relationship (but you can create multiple relationships that are inactive, which can be used with DAX functions like USERELATIONSHIP).
* **Active vs. Inactive Relationships**: You can define multiple relationships between tables, but only one can be active at a time. Inactive relationships can be used in DAX formulas to temporarily activate them when needed.

**Relationship Direction:**

* **Single Directional Relationship**: The filter direction flows in one direction, from one table to another. This is the default relationship in Power BI.
  + **Use Case**: This is useful when you want to filter from the "one" table to the "many" table.
* **Bidirectional Relationship**: The filter direction flows in both directions, meaning changes in one table can affect the other.
  + **Use Case**: This is used when you need to filter data from both tables in a model and requires caution because it can lead to circular dependencies or ambiguous results in complex models.

1. **How to create a date table in DAX?**

To create a **Date Table** in DAX for Power BI, you can use the CALENDAR or CALENDARAUTO function, depending on your needs.

**1. Using CALENDAR Function:**

The CALENDAR function creates a continuous range of dates between a specified start date and end date. This is useful if you have specific date ranges to define for your analysis.

**Example: Creating a Date Table from January 1, 2010, to December 31, 2030.**

DAX

CopyEdit

DateTable =

CALENDAR(DATE(2010, 1, 1), DATE(2030, 12, 31))

This will create a date table with a single column of dates from January 1, 2010, to December 31, 2030.

**2. Using CALENDARAUTO Function:**

The CALENDARAUTO function automatically generates a date range based on the data in your model. It looks at the earliest and latest dates in your data and creates a date table that covers that range.

**Example: Using CALENDARAUTO to automatically generate a date range:**

DAX

CopyEdit

DateTable =

CALENDARAUTO()

CALENDARAUTO() will detect the minimum and maximum dates from your data model and generate a date range that covers the full years in your dataset. If you need more control over the fiscal year, you can pass an optional parameter to CALENDARAUTO() to specify the first month of the fiscal year.

**3. Adding Additional Columns:**

Once you have created your date table, you can add useful columns like Year, Quarter, Month, Weekday, etc., to enhance your date table.

**Example: Adding additional columns to the Date Table**

DAX

CopyEdit

DateTable =

ADDCOLUMNS(

CALENDAR(DATE(2010, 1, 1), DATE(2030, 12, 31)),

"Year", YEAR([Date]),

"Month", MONTH([Date]),

"Month Name", FORMAT([Date], "MMMM"),

"Quarter", QUARTER([Date]),

"Day", DAY([Date]),

"Weekday", WEEKDAY([Date]),

"Weekday Name", FORMAT([Date], "dddd"),

"Year-Month", FORMAT([Date], "YYYY-MM")

)

**Breakdown of Additional Columns:**

* **Year**: Extracts the year from the date.
* **Month**: Extracts the month as a number (1–12).
* **Month Name**: Formats the month as a full month name (e.g., "January").
* **Quarter**: Extracts the quarter (1–4).
* **Day**: Extracts the day of the month.
* **Weekday**: Returns the weekday as a number (1 for Sunday, 7 for Saturday).
* **Weekday Name**: Extracts the full weekday name (e.g., "Monday").
* **Year-Month**: A formatted string like "2023-03" for use in visuals or comparisons.

**4. Marking the Date Table:**

After creating the date table, it's essential to mark it as a "Date Table" in Power BI so it can be used for time intelligence functions (e.g., YTD, QTD, etc.).

To mark the date table:

1. Go to the **Model** view in Power BI.
2. Right-click the DateTable in the fields pane.
3. Select **Mark as Date Table**.
4. Choose the **Date** column as the date field.

**Summary:**

* **CALENDAR**: Use this when you need to specify a custom start and end date for your date table.
* **CALENDARAUTO**: Use this when you want Power BI to automatically create a date table based on the range of dates in your data.
* **ADDCOLUMNS**: Enhances your date table by adding additional useful columns like year, month, weekday, etc.

**Week 4**

1. Difference between star and snowflake schema?

Refer <https://www.geeksforgeeks.org/difference-between-star-schema-and-snowflake-schema/> link

1. Practice from <https://www.examtopics.com/exams/microsoft/pl-300/view/> site for PL-300 certification.

**Week 5**

**Key Time Intelligence Functions in Power BI**

1. Year-to-Date (YTD) Calculations

* Total Sales YTD:

DAX

CopyEdit

TotalSalesYTD = TOTALYTD(SUM(Sales[Amount]), Sales[Date])

* Alternative using DATESYTD:

DAX

CopyEdit

TotalSalesYTD = CALCULATE(SUM(Sales[Amount]), DATESYTD(Sales[Date]))

2. Month-to-Date (MTD) and Quarter-to-Date (QTD)

* Total Sales MTD:

DAX

CopyEdit

TotalSalesMTD = TOTALMTD(SUM(Sales[Amount]), Sales[Date])

* Total Sales QTD:

DAX

CopyEdit

TotalSalesQTD = TOTALQTD(SUM(Sales[Amount]), Sales[Date])

3. Previous Period Comparisons

* Previous Month Sales:

DAX

CopyEdit

SalesPreviousMonth = CALCULATE(SUM(Sales[Amount]), PREVIOUSMONTH(Sales[Date]))

* Previous Year Sales:

DAX

CopyEdit

SalesPreviousYear = CALCULATE(SUM(Sales[Amount]), PREVIOUSYEAR(Sales[Date]))

4. Rolling Averages & Moving Averages

* Rolling 3-Month Average Sales:

DAX

CopyEdit

Rolling3MonthAvg = AVERAGEX(DATESINPERIOD(Sales[Date], MAX(Sales[Date]), -3, MONTH), SUM(Sales[Amount]))

5. Comparing Periods

* Year-over-Year Growth (YoY % Change):

DAX

CopyEdit

YoY Growth = DIVIDE([TotalSales] - [SalesPreviousYear], [SalesPreviousYear])

* Month-over-Month Growth:

DAX

CopyEdit

MoM Growth = DIVIDE([TotalSales] - [SalesPreviousMonth], [SalesPreviousMonth])

6. Custom Date Ranges

* Sales in Last 30 Days:

DAX

CopyEdit

SalesLast30Days = CALCULATE(SUM(Sales[Amount]), DATESINPERIOD(Sales[Date], MAX(Sales[Date]), -30, DAY))

**. Alerts**

**Q1: What are data alerts, and how do they work in BI tools like Power BI?**

**A:** Data alerts notify users when certain conditions in a report or dashboard are met. In Power BI, alerts can be set on dashboard tiles containing card, KPI, or gauge visuals. When the threshold condition is met, users receive notifications via email or in the Power BI service.

**Q2: How would you set up an alert in Power BI?**

**A:**

1. Open a **Power BI dashboard** and locate a tile with a numeric value.
2. Click the **ellipsis (...)** on the tile and select **Manage alerts**.
3. Set a condition, such as "Notify me when the value is greater than 1000."
4. Choose notification frequency and save the alert.

**Q3: What challenges might you face with alerts, and how would you solve them?**

**A:**

* **Delayed alerts** → Check dataset refresh schedules and ensure the gateway is running.
* **Too many notifications** → Fine-tune conditions and adjust thresholds.
* **Missing alerts** → Verify access permissions and check if the data type supports alerts.

**2. Subscription**

**Q4: What is a subscription in Power BI / BI tools?**

**A:** A subscription is an automated way to receive reports or dashboards via email at scheduled intervals. Users can subscribe to reports to get the latest insights without manually accessing the platform.

**Q5: How do you configure a subscription in Power BI?**

**A:**

1. Open a **Power BI report or dashboard**.
2. Click **Subscribe** and enter email recipients.
3. Choose the **delivery frequency** (daily, weekly, etc.).
4. Save the subscription.

**Q6: What are some common issues with subscriptions?**

**A:**

* **Report not sending** → Check dataset refresh status and permissions.
* **Incorrect data** → Ensure scheduled refresh is working before the email is sent.
* **Limited subscribers** → Some BI tools restrict the number of recipients per subscription.

**3. Gateway Setup**

**Q7: What is an On-Premises Data Gateway, and why is it used?**

**A:** An **On-Premises Data Gateway** allows cloud BI services like Power BI to connect securely to on-premises databases such as SQL Server, Oracle, or Excel files. It enables real-time data access and scheduled refreshes.

**Q8: How do you set up a data gateway in Power BI?**

**A:**

1. Download and install the **On-Premises Data Gateway**.
2. Sign in with **Power BI credentials**.
3. Register the gateway and connect it to a **data source** (e.g., SQL Server).
4. Configure **scheduled refresh** settings in Power BI.

**Q9: What are the different types of Power BI gateways?**

**A:**

* **Personal Mode Gateway** → Used for individual reports, supports direct queries.
* **Standard Mode Gateway** → Supports multiple users and enterprise-wide data refreshes.

**Q10: How do you troubleshoot gateway issues?**

**A:**

* **Gateway offline** → Restart the service and check firewall settings.
* **Dataset refresh failed** → Verify authentication and data source connectivity.
* **Performance issues** → Upgrade hardware or distribute load across multiple gateways.

**4. AI Visuals**

**Q11: What AI-powered visuals are available in Power BI?**

**A:**

* **Key Influencers** → Identifies factors affecting a given metric.
* **Decomposition Tree** → Breaks down data hierarchically.
* **Smart Narratives** → Generates text summaries of visualizations.
* **Anomaly Detection** → Highlights unexpected trends in time-series data.

**Q12: How does the Key Influencers visual work?**

**A:** It uses **machine learning** to analyze data and identify key factors affecting an outcome. For example, it can determine what influences customer churn the most.

**Q13: How can AI visuals improve decision-making?**

**A:**

* Detects trends automatically.
* Provides **explainability** to complex datasets.
* Automates insights, reducing manual analysis efforts.

**5. Row-Level Security (RLS)**

**Q14: What is Row-Level Security (RLS) in Power BI?**

**A:** RLS restricts access to **specific rows of data** based on user roles. This ensures that users only see data relevant to them, enhancing security and compliance.

**Q15: How do you implement RLS in Power BI?**

**A:**

1. Define **roles** in Power BI Desktop using DAX filters (e.g., [Region] = "West").
2. Publish the report to the **Power BI Service**.
3. Assign **users or groups** to roles in the Power BI workspace.

**Q16: How does dynamic RLS work?**

**A:** Dynamic RLS assigns access **based on the logged-in user**. Instead of hardcoding filters, it uses a **USERPRINCIPALNAME()** function in DAX to filter data dynamically.

**Q17: What are some challenges with RLS?**

**A:**

* **Performance impact** → Filtering large datasets can slow queries.
* **Complex role management** → Requires careful design to avoid conflicts.
* **Troubleshooting visibility issues** → Testing with "View as Role" helps debug.

**Week 6**

1. **Which chart to use when in powerbi?**

1. Line Chart

📌 When to Use:

* To show trends over time (e.g., sales growth over months).
* To compare multiple trends.

✅ Example:

* Monthly sales revenue from January to December.
* Website traffic trend over a year.

❌ Limitations:

* Can become cluttered with too many lines.
* Not ideal for non-time-series data.

2. Bar/Column Chart

📌 When to Use:

* To compare categories side by side.
* Column charts for time-based comparisons.

✅ Example:

* Bar Chart: Sales by region (NSW, VIC, QLD).
* Column Chart: Quarterly revenue comparison.

❌ Limitations:

* Too many categories make it hard to read.
* Doesn’t show part-to-whole relationships well.

3. Stacked Bar/Column Chart

📌 When to Use:

* To show part-to-whole within categories.
* Useful when comparing sub-categories.

✅ Example:

* Revenue by product (split by online & offline sales).
* Employees by department (split by gender).

❌ Limitations:

* Hard to compare individual segment values across bars.

4. Pie/Donut Chart

📌 When to Use:

* To show part-to-whole proportions (limited to a few categories).

✅ Example:

* Market share of different companies.
* Expense breakdown of a project.

❌ Limitations:

* Hard to read when there are many slices.
* A bar chart is often more effective.

5. Treemap

📌 When to Use:

* To show hierarchical data with size comparison.

✅ Example:

* Company revenue by department and sub-department.
* Website traffic by country and city.

❌ Limitations:

* Difficult to compare small values.

6. Scatter Chart

📌 When to Use:

* To visualize relationships between two numerical values.

✅ Example:

* Ad spend vs. Sales revenue correlation.
* Employee experience vs. Salary.

❌ Limitations:

* Becomes cluttered with too many data points.

7. Histogram

📌 When to Use:

* To show the distribution of a variable.

✅ Example:

* Customer age distribution in a store.
* Delivery time variation across orders.

❌ Limitations:

* Only works with numerical data.

8. Box Plot

📌 When to Use:

* To display data distribution, median, and outliers.

✅ Example:

* Salary range across different job levels.
* Delivery time spread for different regions.

❌ Limitations:

* Not easily understood by non-technical users.

9. Waterfall Chart

📌 When to Use:

* To show incremental changes leading to a total.

✅ Example:

* Profit breakdown: Revenue → Costs → Taxes → Net Profit.
* Budget vs. actual expenses analysis.

❌ Limitations:

* Hard to read if too many small steps.

10. Map & Filled Map

📌 When to Use:

* To display geographic data.

✅ Example:

* Sales by city on a map.
* COVID-19 cases per state in Australia.

❌ Limitations:

* Requires accurate location data (latitude/longitude or city names).

11. KPI Card & Gauge Chart

📌 When to Use:

* To track performance against a target.

✅ Example:

* Monthly sales vs. target (KPI Card).
* Customer satisfaction score progress (Gauge Chart).

❌ Limitations:

* Doesn’t provide trend details like line charts.

**📌 Quick Selection Guide**

| **Chart Type** | **Use Case** | **Example** | **Limitation** |
| --- | --- | --- | --- |
| Line Chart | Trends over time | Monthly sales growth | Cluttered with many lines |
| Bar Chart | Compare categories | Sales by region | Hard to show part-to-whole |
| Stacked Bar | Compare sub-categories | Revenue split by channel | Difficult to compare segments |
| Pie/Donut | Proportion of total | Market share of brands | Hard to read with many slices |
| Treemap | Hierarchical data | Revenue by product category | Hard to compare small values |
| Scatter | Relationship between two variables | Advertising spend vs. sales | Cluttered with many points |
| Histogram | Distribution of data | Age distribution of customers | Only works with numerical data |
| Box Plot | Median, spread & outliers | Salary range by job level | Complex for non-experts |
| Waterfall | Step-by-step change | Profit breakdown | Hard to read with many steps |
| Map | Geospatial data | Sales by city | Needs accurate location data |
| KPI Card | Key metric tracking | Sales vs. Target | No trend visualization |
| Gauge | Performance vs. goal | Customer satisfaction score | Limited detail |

1. **How to import data in sql?**

The Bronze, Silver, and Gold architecture is a common data processing framework used in medallion architecture, especially within Delta Lake and data lakehouses. When applied to data cleaning, each layer progressively refines the data, making it more structured and usable.

1. Bronze Layer (Raw Data)

* Purpose: Stores raw, unprocessed data as ingested from various sources.
* Data Cleaning Focus: Minimal processing, ensuring data is captured without loss.
* Common Steps:
  + Remove corrupt records (e.g., empty rows, unreadable formats).
  + Add ingestion metadata (e.g., timestamps, source IDs).
  + Convert to a standardized format (e.g., Parquet, Delta).

2. Silver Layer (Cleaned & Processed Data)

* Purpose: Standardizes, cleans, and enriches the data for analytical use.
* Data Cleaning Focus: Resolving inconsistencies and ensuring referential integrity.
* Common Steps:
  + Handle missing values (impute or drop).
  + Standardize column names and formats.
  + Remove duplicates and inconsistencies.
  + Enforce schema validation (e.g., correct data types).
  + Perform deduplication and integrity checks.

3. Gold Layer (Curated & Analytics-Ready Data)

* Purpose: Optimized for business intelligence (BI), reporting, and ML.
* Data Cleaning Focus: High-level aggregations and transformations.
* Common Steps:
  + Create business-specific tables (e.g., summary reports, KPIs).
  + Perform aggregations and calculations.
  + Join with other enriched datasets.
  + Optimize indexing for faster queries.

**Example: Customer Transactions Data Cleaning**

| Step | Bronze (Raw Data) | Silver (Cleaned & Processed) | Gold (Analytics-Ready) |
| --- | --- | --- | --- |
| Duplicate Removal | Raw entries with duplicates | Deduplicated | Aggregated customer-level data |
| Missing Values | Null values in important columns | Imputed or removed | No missing values |
| Schema Standardization | Inconsistent column names | Standardized column names | Business-friendly names |
| Aggregations | Detailed transaction records | Cleaned transaction table | Customer lifetime value, total sales per region |

* + 1. SQL Basics

**1. What are the different types of SQL commands?**

* **DDL (Data Definition Language)**: CREATE, ALTER, DROP
* **DML (Data Manipulation Language)**: SELECT, INSERT, UPDATE, DELETE
* **DCL (Data Control Language)**: GRANT, REVOKE
* **TCL (Transaction Control Language)**: COMMIT, ROLLBACK

**2. How to retrieve all records from a table?**

sql

CopyEdit

SELECT \* FROM employees;

**3. How to retrieve specific columns from a table?**

sql

CopyEdit

SELECT first\_name, last\_name FROM employees;

**4. What is the difference between COUNT(\*) and COUNT(column\_name)?**

sql

CopyEdit

SELECT COUNT(\*) FROM employees; -- Counts all rows

SELECT COUNT(department) FROM employees; -- Counts only non-null departments

**5. How does DISTINCT work in SQL?**

sql

CopyEdit

SELECT DISTINCT department FROM employees;

* Removes duplicate values.

**6. What is the difference between WHERE and HAVING?**

* WHERE filters **before** aggregation.
* HAVING filters **after** aggregation.

sql

CopyEdit

-- WHERE filters before aggregation

SELECT \* FROM sales WHERE region = 'East';

-- HAVING filters after aggregation

SELECT region, SUM(sales)

FROM sales

GROUP BY region

HAVING SUM(sales) > 50000;

**7. What is the difference between UNION and UNION ALL?**

sql

CopyEdit

SELECT first\_name FROM employees UNION SELECT first\_name FROM customers;

* UNION: Removes duplicates.
* UNION ALL: Includes duplicates.

**8. How to filter records using WHERE?**

sql

CopyEdit

SELECT \* FROM employees WHERE age > 30;

**9. How to filter records using BETWEEN?**

sql

CopyEdit

SELECT \* FROM employees WHERE age BETWEEN 25 AND 40;

**10. How to filter records using IN?**

sql

CopyEdit

SELECT \* FROM employees WHERE department IN ('HR', 'Finance', 'IT');

**2. SQL Joins**

**11. What are different types of joins?**

* **INNER JOIN**: Returns matching rows.
* **LEFT JOIN**: Returns all rows from the left table and matching rows from the right.
* **RIGHT JOIN**: Returns all rows from the right table and matching rows from the left.
* **FULL JOIN**: Returns all rows from both tables.

**12. INNER JOIN example**

sql

CopyEdit

SELECT e.name, d.department\_name

FROM employees e

INNER JOIN departments d ON e.department\_id = d.department\_id;

**13. LEFT JOIN example**

sql

CopyEdit

SELECT e.name, d.department\_name

FROM employees e

LEFT JOIN departments d ON e.department\_id = d.department\_id;

**14. RIGHT JOIN example**

sql

CopyEdit

SELECT e.name, d.department\_name

FROM employees e

RIGHT JOIN departments d ON e.department\_id = d.department\_id;

**15. FULL JOIN example**

sql

CopyEdit

SELECT e.name, d.department\_name

FROM employees e

FULL JOIN departments d ON e.department\_id = d.department\_id;

**3. Aggregation Functions**

**16. How to find the total salary of employees?**

sql

CopyEdit

SELECT SUM(salary) FROM employees;

**17. How to find the average salary?**

sql

CopyEdit

SELECT AVG(salary) FROM employees;

**18. How to find the maximum and minimum salary?**

sql

CopyEdit

SELECT MAX(salary) FROM employees;

SELECT MIN(salary) FROM employees;

**19. How to group data using GROUP BY?**

sql

CopyEdit

SELECT department, COUNT(\*) FROM employees GROUP BY department;

**20. How to filter aggregated data using HAVING?**

sql

CopyEdit

SELECT department, AVG(salary) FROM employees

GROUP BY department

HAVING AVG(salary) > 50000;

**4. Advanced Filtering and Sorting**

**21. How to sort records in ascending order?**

sql

CopyEdit

SELECT \* FROM employees ORDER BY salary ASC;

**22. How to sort records in descending order?**

sql

CopyEdit

SELECT \* FROM employees ORDER BY salary DESC;

**23. How to get the top 5 highest salaries?**

sql

CopyEdit

SELECT \* FROM employees ORDER BY salary DESC LIMIT 5;

**24. How to find employees earning more than the department average?**

sql

CopyEdit

SELECT name, salary FROM employees e

WHERE salary > (SELECT AVG(salary) FROM employees WHERE department\_id = e.department\_id);

**25. How to find duplicate records?**

sql

CopyEdit

SELECT name, COUNT(\*) FROM employees GROUP BY name HAVING COUNT(\*) > 1;

**5. Subqueries**

**26. What is a subquery?**

A query inside another query.

**27. Find employees in departments with more than 10 employees.**

sql

CopyEdit

SELECT name FROM employees WHERE department\_id IN (

SELECT department\_id FROM employees GROUP BY department\_id HAVING COUNT(\*) > 10

);

**28. Find the second highest salary.**

sql

CopyEdit

SELECT DISTINCT salary FROM employees ORDER BY salary DESC LIMIT 1 OFFSET 1;

**29. Find employees who do not have a department assigned.**

sql

CopyEdit

SELECT \* FROM employees WHERE department\_id IS NULL;

**30. Find the employees who were hired before a specific date.**

sql

CopyEdit

SELECT \* FROM employees WHERE hire\_date < '2023-01-01';

**6. Window Functions**

**31. How to rank employees by salary?**

sql

CopyEdit

SELECT name, salary, RANK() OVER (ORDER BY salary DESC) AS rank FROM employees;

**32. How to calculate a running total of sales?**

sql

CopyEdit

SELECT order\_date, customer\_id,

SUM(amount) OVER (PARTITION BY customer\_id ORDER BY order\_date) AS running\_total

FROM orders;

**33. How to get the last order date for each customer?**

sql

CopyEdit

SELECT customer\_id, MAX(order\_date) FROM orders GROUP BY customer\_id;

**34. How to use CASE WHEN for conditional logic?**

sql

CopyEdit

SELECT name, salary,

CASE

WHEN salary > 80000 THEN 'High'

WHEN salary BETWEEN 50000 AND 80000 THEN 'Medium'

ELSE 'Low'

END AS salary\_category

FROM employees;

**7. Data Modifications**

**35. How to insert a new record?**

sql

CopyEdit

INSERT INTO employees (name, department, salary) VALUES ('John Doe', 'HR', 50000);

**36. How to update a record?**

sql

CopyEdit

UPDATE employees SET salary = 60000 WHERE name = 'John Doe';

**37. How to delete a record?**

sql

CopyEdit

DELETE FROM employees WHERE name = 'John Doe';

**38. How to create a new table?**

sql

CopyEdit

CREATE TABLE employees (

id INT PRIMARY KEY,

name VARCHAR(100),

department VARCHAR(50),

salary DECIMAL(10,2)

);

**39. How to drop a table?**

sql

CopyEdit

DROP TABLE employees;

**All Types of SQL Joins with Examples (Commonly Asked in Interviews)**

SQL joins are used to combine records from two or more tables based on a related column. Here are the different types of joins, their explanations, and examples.

Visualize:- https://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins

**1. INNER JOIN**

🔹 **Returns only the matching rows from both tables.**  
✅ **Example Use Case**: Find employees who have a department assigned.

sql

CopyEdit

SELECT e.employee\_id, e.name, d.department\_name

FROM employees e

INNER JOIN departments d ON e.department\_id = d.department\_id;

✅ **Example Dataset:**

| **employee\_id** | **name** | | **department\_id** | |
| --- | --- | --- | --- | --- |
| 1 | Alice | | 101 | |
| 2 | Bob | | 102 | |
| 3 | Charlie | | NULL | |
| **department\_id** | | **department\_name** | |
| 101 | | HR | |
| 102 | | IT | |

✅ **Result:**

| **employee\_id** | **name** | **department\_name** |
| --- | --- | --- |
| 1 | Alice | HR |
| 2 | Bob | IT |

**2. LEFT JOIN (or LEFT OUTER JOIN)**

🔹 **Returns all rows from the left table and matching rows from the right table. If no match, NULL is returned for right table columns.**  
✅ **Example Use Case**: Find all employees, even those without a department.

sql

CopyEdit

SELECT e.employee\_id, e.name, d.department\_name

FROM employees e

LEFT JOIN departments d ON e.department\_id = d.department\_id;

✅ **Result:**

| **employee\_id** | **name** | **department\_name** |
| --- | --- | --- |
| 1 | Alice | HR |
| 2 | Bob | IT |
| 3 | Charlie | NULL |

**3. RIGHT JOIN (or RIGHT OUTER JOIN)**

🔹 **Returns all rows from the right table and matching rows from the left table. If no match, NULL is returned for left table columns.**  
✅ **Example Use Case**: Find all departments, even those with no employees.

sql

CopyEdit

SELECT e.employee\_id, e.name, d.department\_name

FROM employees e

RIGHT JOIN departments d ON e.department\_id = d.department\_id;

✅ **Example Additional Data in Departments Table:**

| **department\_id** | **department\_name** |
| --- | --- |
| 101 | HR |
| 102 | IT |
| 103 | Finance |

✅ **Result:**

| **employee\_id** | **name** | **department\_name** |
| --- | --- | --- |
| 1 | Alice | HR |
| 2 | Bob | IT |
| NULL | NULL | Finance |

**4. FULL JOIN (or FULL OUTER JOIN)**

🔹 **Returns all rows from both tables. If there's no match, NULL is filled in from the missing side.**  
✅ **Example Use Case**: Find all employees and all departments, even if some employees don't have a department or some departments have no employees.

sql

CopyEdit

SELECT e.employee\_id, e.name, d.department\_name

FROM employees e

FULL JOIN departments d ON e.department\_id = d.department\_id;

✅ **Result:**

| **employee\_id** | **name** | **department\_name** |
| --- | --- | --- |
| 1 | Alice | HR |
| 2 | Bob | IT |
| 3 | Charlie | NULL |
| NULL | NULL | Finance |

**5. CROSS JOIN**

🔹 **Returns the Cartesian product (all possible combinations) of the two tables.**  
✅ **Example Use Case**: Pair every employee with every department.

sql

CopyEdit

SELECT e.employee\_id, e.name, d.department\_name

FROM employees e

CROSS JOIN departments d;

✅ **Result (All Combinations):**

| **employee\_id** | **name** | **department\_name** |
| --- | --- | --- |
| 1 | Alice | HR |
| 1 | Alice | IT |
| 1 | Alice | Finance |
| 2 | Bob | HR |
| 2 | Bob | IT |
| 2 | Bob | Finance |
| 3 | Charlie | HR |
| 3 | Charlie | IT |
| 3 | Charlie | Finance |

**6. SELF JOIN**

🔹 **A table joins with itself.**  
✅ **Example Use Case**: Find employees and their managers.

sql

CopyEdit

SELECT e1.name AS employee, e2.name AS manager

FROM employees e1

LEFT JOIN employees e2 ON e1.manager\_id = e2.employee\_id;

✅ **Example Dataset:**

| **employee\_id** | **name** | **manager\_id** |
| --- | --- | --- |
| 1 | Alice | 3 |
| 2 | Bob | 3 |
| 3 | Charlie | NULL |

✅ **Result:**

| **employee** | **manager** |
| --- | --- |
| Alice | Charlie |
| Bob | Charlie |

**7. NATURAL JOIN**

🔹 **Automatically joins tables based on common column names (dangerous in large databases).**  
✅ **Example Use Case**: Find employees with their department names.

sql

CopyEdit

SELECT \* FROM employees NATURAL JOIN departments;

✅ **Works like INNER JOIN but automatically detects the column to join on.**

**Interview Insights:**

* **INNER JOIN**: Most commonly used.
* **LEFT JOIN**: Important when working with missing data.
* **RIGHT JOIN**: Less common but useful for ensuring completeness.
* **FULL JOIN**: Used when you need to compare two datasets fully.
* **CROSS JOIN**: Used for creating all possible combinations.
* **SELF JOIN**: Important in hierarchical relationships.
* **NATURAL JOIN**: **Avoid in real-world scenarios** because it relies on column names matching.

Week 8:

Slowly Changing Dimensions (SCD) is an important concept in data warehousing, especially for tracking and managing historical changes in data over time. In simple terms, **SCDs** help us understand how data changes and how we can handle those changes in a way that is both efficient and accurate.

**Types of Slowly Changing Dimensions (SCD)**

There are 4 primary types of SCDs: **Type 0, Type 1, Type 2, and Type 3**.

**SCD Type 0: No Change**

* **Explanation**: The data never changes. Once a record is inserted, it remains constant and unaltered.
* **Use Case**: Rare in practice; this might be used for data that is **immutable**, such as a product SKU.

**Example**:

* Product: 12345 — *initial data entry: Product Name = "Laptop"*
* No updates or changes to this product are allowed after it is initially entered.

**SCD Type 1: Overwrite**

* **Explanation**: When data changes, the old value is **overwritten** by the new value. No historical data is retained.
* **Use Case**: Used when historical accuracy is not important, and you only need the **current state** of the data.

**Example**:

* Suppose a customer changes their email address.
  + **Initial data**:
    - CustomerID=101, Name=Alice, Email=alice@example.com
  + After a change:
    - CustomerID=101, Name=Alice, Email=alice.new@example.com

Here, the old email is simply replaced, and no historical email addresses are kept.

**SCD Type 2: Add New Row for Changes**

* **Explanation**: When a change happens, **a new row is added** to store the new value. The old row is retained but marked as inactive or expired. This is used when it's important to **track historical data** (like customer addresses, product prices, etc.).
* **Key Concepts**:
  + **Effective Date**: The date the record became valid.
  + **Expiration Date**: The date the record was superseded by a new version.
  + **Current Flag**: A flag to indicate whether the record is the most recent version.

**Example**:

* **Customer**: 101 changes address.
  + **Initial record**:
    - CustomerID=101, Name=Alice, Address=123 Main St, EffectiveDate=2020-01-01, ExpiryDate=None, CurrentFlag=Y
  + **Updated record** (new address):
    - CustomerID=101, Name=Alice, Address=456 Oak St, EffectiveDate=2021-01-01, ExpiryDate=None, CurrentFlag=Y
  + The old record would be updated:
    - CustomerID=101, Name=Alice, Address=123 Main St, EffectiveDate=2020-01-01, ExpiryDate=2020-12-31, CurrentFlag=N

**SCD Type 3: Add New Attribute for Changes**

* **Explanation**: When data changes, a **new column** is added to the existing row to store the old value, and the new value is recorded in the original column. This method keeps **only a limited history** (usually 1 or 2 previous states).
* **Use Case**: Used when it's necessary to store only a limited amount of historical data (e.g., **previous** and **current** values).

**Example**:

* **Product**: ProductID=101 changes its description.
  + **Initial record**:
    - ProductID=101, ProductName=Widget, Description=Basic Widget
  + **Updated record** (with previous description stored):
    - ProductID=101, ProductName=Widget, Description=Advanced Widget, PreviousDescription=Basic Widget

**SCD Type 4: Separate Historical Table**

* **Explanation**: This type involves creating a separate historical table to store all past versions of records, while the main dimension table contains only the current record.
* **Use Case**: Used when we want to keep a clean dimension table but also retain a full history in a separate table.

**Example**:

* The **Current Table** stores only the latest record:
  + CustomerID=101, Name=Alice, Address=456 Oak St
* The **History Table** stores all previous versions:
  + CustomerID=101, Name=Alice, Address=123 Main St, EffectiveDate=2020-01-01
  + CustomerID=101, Name=Alice, Address=456 Oak St, EffectiveDate=2021-01-01

**Example of Type 2 Implementation in SQL Server**

Let's assume we are tracking changes to **Customer Information** in a Type 2 SCD system.

**Step 1: Table Setup**

sql

CopyEdit

CREATE TABLE Customers (

CustomerID INT,

Name VARCHAR(100),

Address VARCHAR(200),

EffectiveDate DATE,

ExpiryDate DATE,

CurrentFlag CHAR(1) -- 'Y' for current, 'N' for historical

);

**Step 2: Inserting Initial Data**

sql

CopyEdit

INSERT INTO Customers (CustomerID, Name, Address, EffectiveDate, ExpiryDate, CurrentFlag)

VALUES (101, 'Alice', '123 Main St', '2020-01-01', NULL, 'Y');

**Step 3: Updating Customer Data (with SCD Type 2 logic)**

When Alice moves, the previous record is marked as inactive, and a new record is inserted.

sql

CopyEdit

-- Mark the old record as expired

UPDATE Customers

SET ExpiryDate = '2021-01-01', CurrentFlag = 'N'

WHERE CustomerID = 101 AND CurrentFlag = 'Y';

-- Insert the new record with updated address

INSERT INTO Customers (CustomerID, Name, Address, EffectiveDate, ExpiryDate, CurrentFlag)

VALUES (101, 'Alice', '456 Oak St', '2021-01-01', NULL, 'Y');

**Step 4: Querying the Latest Customer Data**

sql

CopyEdit

SELECT \* FROM Customers WHERE CustomerID = 101 AND CurrentFlag = 'Y';

**SCD Best Practices**

* **For Type 2**, use **surrogate keys** (like a Customer\_SK in the table) rather than the business key (CustomerID) to ensure each historical record has a unique ID.
* **Efficient storage**: For large datasets, partitioning the historical table by time periods (e.g., monthly) can improve performance.

**Summary**

* **SCD Type 0**: No changes are tracked (static data).
* **SCD Type 1**: Overwrite the old value with the new one.
* **SCD Type 2**: Track full history by creating new records for changes.
* **SCD Type 3**: Store limited history by adding new columns for the previous value.
* **SCD Type 4**: Use a separate history table for tracking changes.

If you're working on a data warehousing project, you'll need to decide which type of SCD best fits the needs of your business use case. Would you like more detailed examples or code for implementing these types?