DATABASES AND SQL

TABLE OF CONTENTS

Databases help in systematically storing, retrieving, and managing data, while SQL (Structured Query Language) is the standard language to interact with relational databases.

Example: A company's customer database, which includes tables for storing details about customers, orders, and products. SQL is used to add, update, delete, and query data from these tables.

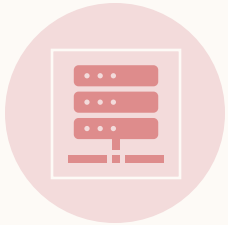
WHY DATABASES ARE NEEDED?

Databases ensure organized data storage, data consistency, controlled access, and efficient data retrieval.

Example: In an e-commerce system, databases store product information, user profiles, order histories, and payment data, allowing applications to retrieve information quickly and securely.

WHAT IS A DATABASE AND DBMS?

Database A database is a structured collection of related data, organized in a way that allows efficient storage, retrieval, and manipulation. Databases are used in almost every domain to store information—whether it's customer records, product catalogs, or transaction histories.



DBMS: A DBMS is the software that manages and provides the interface to interact with the database. It controls the data structure, storage, access methods, and security, allowing users and applications to read and write data with ease.

Example: MySQL, a popular DBMS, efficiently handles large datasets, as in e-commerce applications with millions of records.

WHERE ARE DATABASES MAINTAINED?

On-Premise Servers
- Databases hosted on physical hardware owned and maintained by the organization.
- **Examples**:
  - MySQL on an internal server
  - Microsoft SQL Server on company hardware

Database as a Service (DBaaS)
- Managed database services provided by third-party vendors in the cloud, which handle most administrative tasks.
- **Examples**:
  - MongoDB Atlas (NoSQL DBaaS)
  - Firebase Realtime Database

Cloud Databases
- Databases hosted on cloud service providers, offering scalability, remote access, and managed services.
- **Examples**:
  - Amazon RDS (Relational Database Service)
  - Google Cloud SQL
  - Azure SQL Database

Hybrid Databases
- A mix of on-premise and cloud environments, enabling flexibility and integration between systems.
- **Examples**:
  - Microsoft SQL Server with a hybrid cloud setup
  - Oracle databases using a combination of on-premise and cloud resources.

TYPES OF DATABASES

- **Key-Value**: Stores simple data pairs, efficient for fast lookups.
- **Example**: Redis storing session data with session ID as key and user data as value.

- **Column-Based**: Stores data in columns rather than rows, ideal for analytical queries.
- **Example**: Apache Cassandra storing large-scale event data where columns represent specific metrics.

- **Document-Based**: Stores data in document format, usually JSON or BSON.
- **Example**: MongoDB, where each document could represent a user profile with flexible attributes.

- **Graph-Based**: Represents relationships as nodes and edges, ideal for complex relationships.
- **Example**: Neo4j used to model social media relationships, where users are nodes and friendships are edges.

SQL

SQL is the language used to interact with relational databases.

DDL (Data Definition Language): Creates and modifies database structures.

- **Example**: CREATE TABLE Customers (ID INT, Name VARCHAR(100));

DML (Data Manipulation Language): Manipulates data within tables.

- **Example**: INSERT INTO Customers (ID, Name) VALUES (1, 'Alice');

DCL (Data Control Language): Controls access to data.

- **Example**: GRANT SELECT ON Customers TO 'user123';

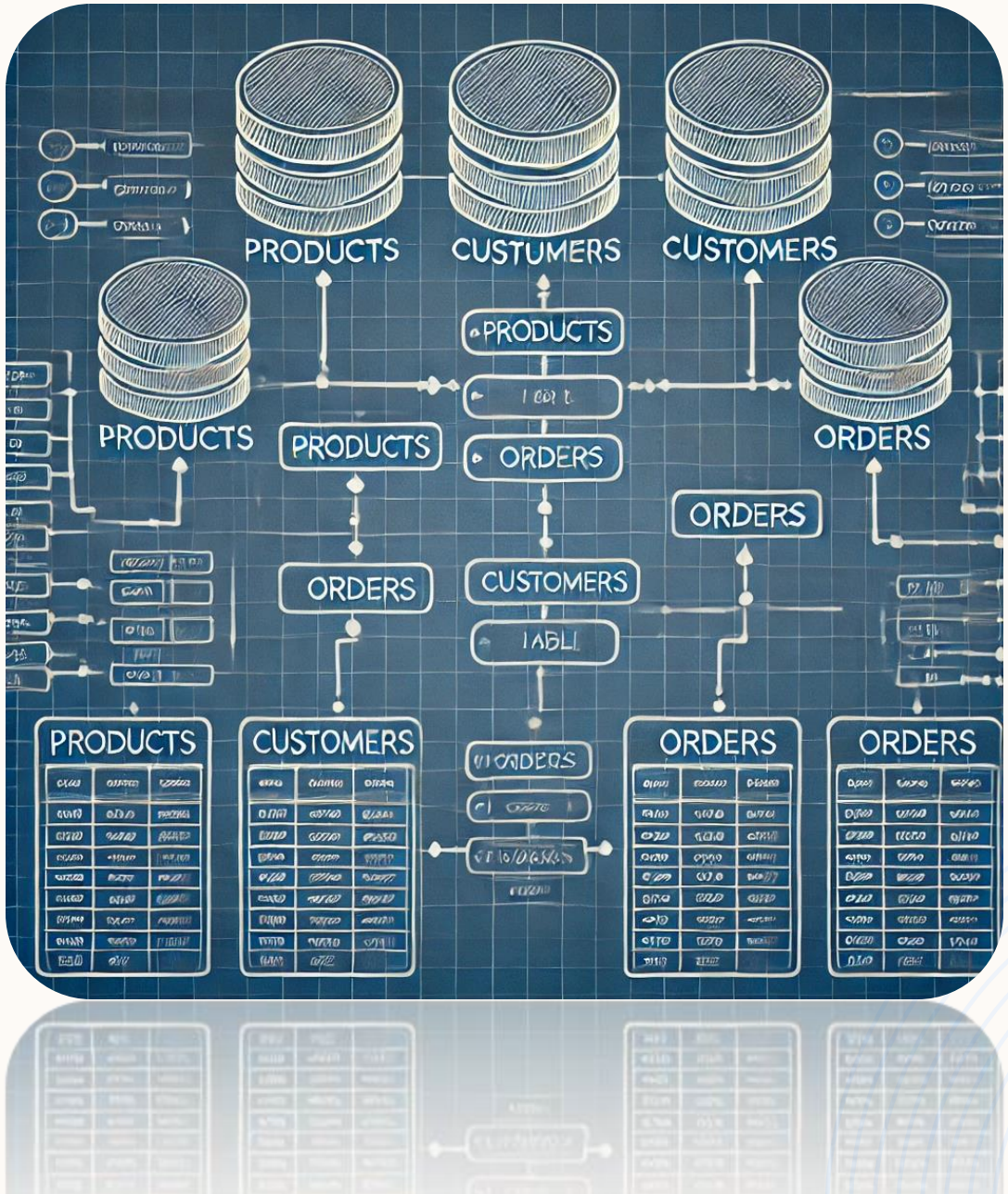
TCL (Transaction Control Language): Manages transactions for data integrity.

- **Example**: COMMIT; to save all changes in a transaction.

SCHEMA

A blueprint of the database structure, including tables and relationships.

Example: In an online store database, the schema might define tables like Products, Customers, and Orders, with relationships between customers and orders.



VIEWS

Views: Virtual tables derived from queries, simplifying data presentation and enhancing security.

Example:

```
CREATE VIEW CustomerOrders AS SELECT Name, OrderDate FROM Customers JOIN Orders
ON Customers.ID = Orders.CustomerID;
```

This view shows customer names and their order dates, hiding other columns.

TRIGGERS

Triggers: Automated actions executed in response to events, often used for auditing or enforcing rules.

Example:
A trigger to log each update on a product's price.

```
CREATE TRIGGER LogPriceChange
AFTER UPDATE ON Products
FOR EACH ROW
BEGIN
  INSERT INTO PriceLog(ProductID, OldPrice, NewPrice) VALUES (OLD.ID, OLD.Price, NEW.Price);
END;
```

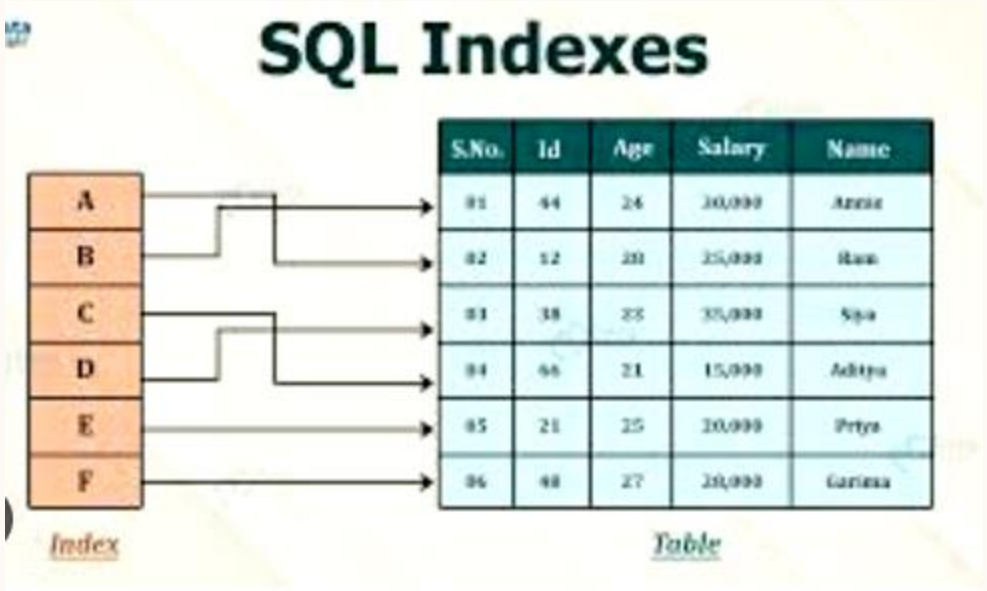
INDEXES

Indexes: Data structures that make data retrieval faster by providing quick access paths.

Example:

Adding an index on a Customers table's LastName column improves the speed of queries filtering by last name.

```
CREATE INDEX idx_lastname ON
Customers(LastName);
```

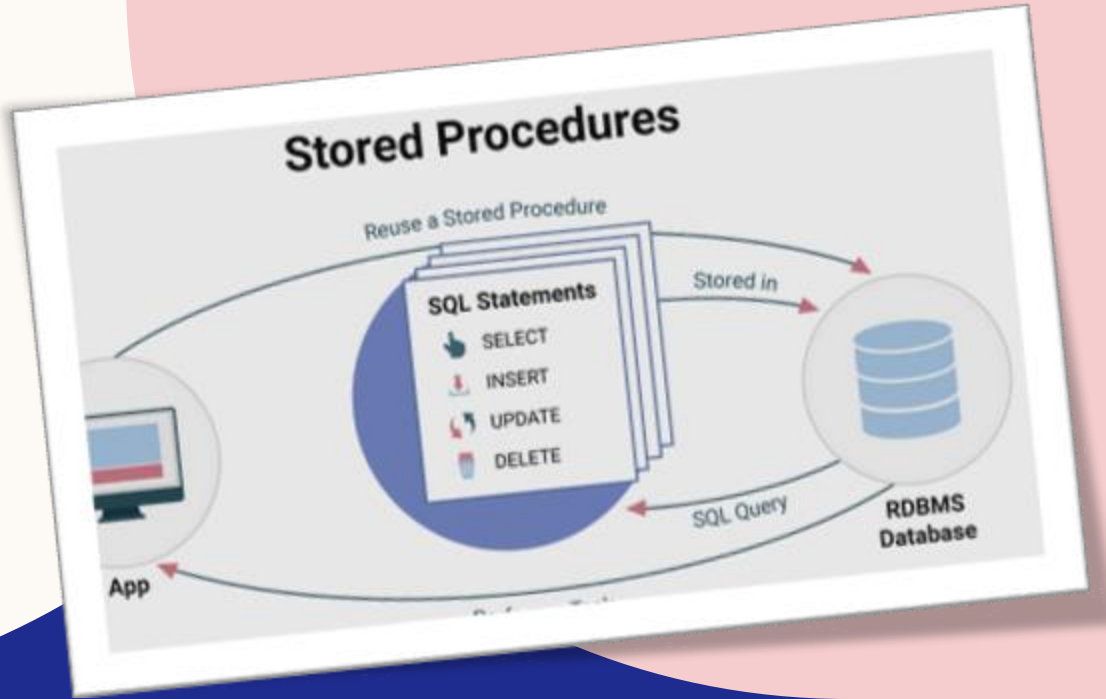


STORED PROCEDURES

Reusable SQL code blocks for performing repetitive tasks, encapsulating logic for easier maintenance and security.

•**Example**: A stored procedure to retrieve all orders for a specific customer.

```
CREATE PROCEDURE GetCustomerOrders (IN customerID INT)
BEGIN
    SELECT * FROM Orders WHERE CustomerID = customerID;
END;
```



Advanced SQL Concepts

NULL

In SQL, **NULL** represents a missing or undefined value. It is not the same as an empty string or a zero—it specifically indicates the absence of data.

Example:

To find all customers who do not have a phone number, the query would be:

```
SELECT * FROM Customers WHERE PhoneNumber IS NULL;
```

This query returns all rows where the **PhoneNumber** field is **NULL** (i.e., missing).

Practical Use:

- Detect incomplete records.
- Handle missing data in reports or data validation tasks.

JOINS

Joins are used to combine rows from two or more tables based on a related column.

Types of Joins:

Inner Join: Returns only matching rows.

```
SELECT Orders.OrderID, Customers.Name
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.ID;
```

Left Join: Returns all rows from the left table and matched rows from the right table.

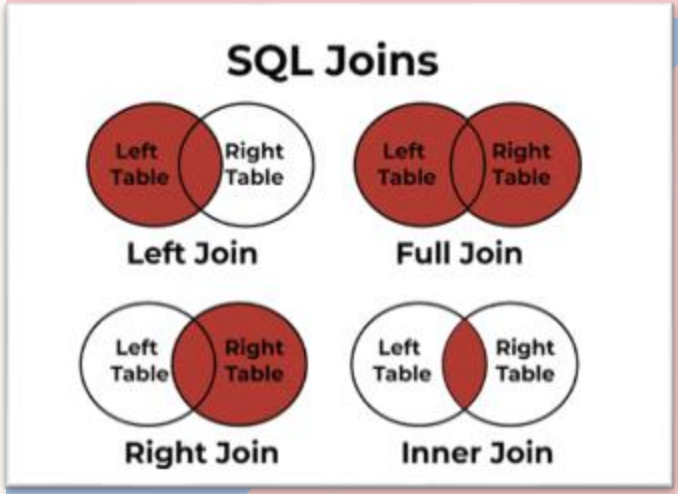
```
SELECT Orders.OrderID, Customers.Name
FROM Orders
LEFT JOIN Customers ON Orders.CustomerID = Customers.ID;
```

Right Join: Returns all rows from the right table and matched rows from the left table.

```
SELECT Orders.OrderID, Customers.Name
FROM Orders
RIGHT JOIN Customers ON Orders.CustomerID = Customers.ID;
```

Full Join: Returns all rows when there is a match in either table.

```
SELECT Orders.OrderID, Customers.Name
FROM Orders
FULL JOIN Customers ON Orders.CustomerID = Customers.ID;
```



Aggregations

- Aggregate functions perform calculations on sets of rows, returning a single result.

Common Aggregates:

- **SUM**: Calculates the total.

```
SELECT SUM(Price) FROM Orders;
```

- **AVG**: Finds the average.

```
SELECT AVG(Price) FROM Products;
```

- **COUNT**: Counts rows.

```
SELECT COUNT(*) FROM Customers;
```

- **MAX / MIN**: Finds the maximum or minimum.

```
SELECT MAX(Price), MIN(Price) FROM Products;
```

Aliases

- Aliases provide temporary names for tables or columns to improve readability.
- **Example**:

```
SELECT Customers.Name AS CustomerName, Orders.OrderDate AS OrderDate
FROM Customers
INNER JOIN Orders ON Customers.ID = Orders.CustomerID;
```

Use Case: Useful for simplifying complex queries or avoiding column name conflicts.

GroupBy

- The GROUP BY statement groups rows with the same values in specified columns, often used with aggregate functions.

- **Example:**

```
SELECT Country, COUNT(*) AS TotalCustomers
FROM Customers
GROUP BY Country;
```

Use Case: Useful for summarizing data by category, like total sales by region.

Having

- The HAVING clause filters groups after the aggregation.
- **Example:**

```
SELECT Country, COUNT(*) AS TotalCustomers
FROM Customers
GROUP BY Country
HAVING COUNT(*) > 5;
```

Use Case: Often used to filter groups based on aggregate conditions (e.g., only countries with more than five customers).

Complex Queries

- Complex queries involve multiple subqueries, joins, and conditions to retrieve precise information.
- **Example:** Finding customers with the highest total order amount.

```
SELECT CustomerID, SUM(TotalAmount) AS TotalSpent
FROM Orders
GROUP BY CustomerID
ORDER BY TotalSpent DESC
LIMIT 1;
```

•**Use Case**: Useful for business insights that require detailed data analysis, like identifying top customers or products.

QUERY TUNING

- Query tuning refers to the process of optimizing database queries to improve their performance, reduce execution time, and minimize resource consumption. The goal is to ensure that queries run efficiently, especially in environments with large datasets or complex operations.

Example: Analyzing a query execution plan to identify bottlenecks and adjusting indexes accordingly.

```
EXPLAIN SELECT * FROM Orders WHERE OrderDate > '2024-01-01';
```

THANK YOU