# **Introduction to Databases**

A database is a structured collection of data that allows for efficient storage, retrieval, and manipulation. Databases are essential in software development, enabling applications to manage large amounts of data effectively.

## **Types of Databases**

1. **Relational Databases (RDBMS)**
   * Use structured schemas and tables to store data.
   * Follow ACID (Atomicity, Consistency, Isolation, Durability) principles.
   * Examples: PostgreSQL, MySQL, Oracle, SQL Server.
2. **NoSQL Databases**
   * Handle unstructured or semi-structured data.
   * Do not rely strictly on schemas.
   * Categories:  
     + Document-based (MongoDB, CouchDB)
     + Key-value stores (Redis, DynamoDB)
     + Column-family stores (Cassandra, HBase)
     + Graph-based (Neo4j)

## **Database Clients**

Database clients are tools or interfaces used to interact with databases. Examples:

* **pgAdmin** (Graphical interface for PostgreSQL)
* **DBeaver** (Universal database tool)
* **psql** (PostgreSQL command-line interface)

# **Relational Database Management Systems (RDBMS)**

RDBMS is a type of database management system that organizes data into tables with relationships. Features include:

* Tables, rows, and columns.
* Primary and foreign keys.
* Normalization to eliminate redundancy.
* Support for SQL queries.

## **Key Terminologies**

* **Table**: A structured set of data.
* **Row (Record)**: A single entry in a table.
* **Column (Field)**: A specific attribute of a record.
* **Primary Key (PK)**: A unique identifier for each row.
* **Foreign Key (FK)**: A reference to a primary key in another table.
* **Unique Key**: Ensures all values in a column are unique.
* **Index**: Speeds up search queries.
* **Composite Key**: A primary key consisting of multiple columns.
* **View**: A virtual table based on a query.

# **Normalization**

Normalization is the process of structuring a relational database to reduce data redundancy and improve data integrity. It involves dividing large tables into smaller ones and defining relationships among them.

### **Normalization Forms:**

1. **First Normal Form (1NF)** - Ensure all columns contain atomic values, and each column has a unique name.
2. **Second Normal Form (2NF)** - Meet 1NF requirements and ensure that all non-key attributes depend on the entire primary key.
3. **Third Normal Form (3NF)** - Meet 2NF requirements and remove transitive dependencies.

**Example of Normalization:**

**Unnormalized Table:**

| **OrderID** | **CustomerName** | **Product** | **Quantity** |
| --- | --- | --- | --- |
| 1 | John Doe | Laptop | 1 |
| 1 | Kolade Doe | Mouse | 2 |

**Normalized Tables:**

**Customers Table:**

| **CustomerID** | **CustomerName** |
| --- | --- |
| 1 | John Doe |

**Orders Table:**

| **OrderID** | **CustomerID** | **ProductID** | **Qty** |
| --- | --- | --- | --- |
| 1 | 1 | 1 | 1 |

**Product Table:**

| **ID** | **ProductName** |  |
| --- | --- | --- |
| 1 | laptop |  |
| 2 | Mouse |  |

**OrderDetails Table:**

| **OrderDetailID** | **OrderID** | **Product** | **Quantity** |
| --- | --- | --- | --- |
| 1 | 1 | Laptop | 1 |
| 2 | 1 | Mouse | 2 |

# **PostgreSQL**

PostgreSQL is an advanced, open-source RDBMS known for its extensibility, reliability, and support for SQL and NoSQL features.

## **Installation**

### **On Ubuntu:**

sudo apt update

sudo apt install postgresql postgresql-contrib

### **On Windows:**

Download and install PostgreSQL from<https://www.postgresql.org/download/>

### **Starting PostgreSQL**

sudo systemctl start postgresql

sudo -i -u postgres

psql

# **Structured Query Language (SQL)**

SQL is used to manage and query databases. It includes:

## **Data Definition Language (DDL)**

Used to define and modify database structures.

* **CREATE**
* **ALTER**
* **DROP**

## **Data Manipulation Language (DML)**

Used for data handling.

* **SELECT**
* **INSERT**
* **UPDATE**
* **DELETE**

## **Transaction Control Statements (TCS)**

* **COMMIT**
* **ROLLBACK**
* **SAVEPOINT**

# **SQL Queries in PostgreSQL**

## **Creating a Table with Constraints**

CREATE TABLE users (

id SERIAL PRIMARY KEY,

name VARCHAR(100),

email VARCHAR(100) UNIQUE,

age INT CHECK (age >= 18)

);

## **Creating a Table with Foreign Key and Composite Key**

CREATE TABLE orders (

order\_id SERIAL,

user\_id INT,

product VARCHAR(100),

quantity INT,

PRIMARY KEY (order\_id, product),

FOREIGN KEY (user\_id) REFERENCES users(id)

);

## **SQL Queries in PostgreSQL**

### **6.1 Creating a Table**

## **sql**

## 

## **CREATE TABLE users (**

## **id SERIAL PRIMARY KEY,**

## **name VARCHAR(100),**

## **email VARCHAR(100) UNIQUE,**

## **age INT**

## **);**

ALTER TABLE USERS ADD COLUMN ADDRESS TEXT;

## 

### **6.2 Inserting Data**

## **sql**

## 

## **INSERT INTO users (name, email, age) VALUES ('John Doe', 'john@example.com', 30);**

## 

### **6.3 Selecting Data**

## **sql**

## 

## **SELECT \* FROM users;**

## **SELECT name, age FROM users WHERE age > 25;**

## 

### **6.4 Updating Data**

## **sql**

## 

## **UPDATE users SET age = 31 WHERE name = 'John Doe';**

## 

### **6.5 Deleting Data**

## **sql**

## 

## **DELETE FROM users WHERE name = 'John Doe';**

## 

## 

## **7. Advanced SQL Queries**

### **7.1 WHERE Clause**

## **Filters data based on conditions.**

## **sql**

## 

## **SELECT \* FROM users WHERE age >= 30;**

## 

### **7.2 JOINS (Combining Multiple Tables)**

#### **INNER JOIN**

## **sql**

## 

## **CREATE TABLE orders (**

## **id SERIAL PRIMARY KEY,**

## **user\_id INT REFERENCES users(id),**

## **amount DECIMAL(10,2)**

## **);**

## 

## **SELECT users.name, orders.amount**

## **FROM users**

## **JOIN orders ON users.id = orders.user\_id;**

## 

#### **LEFT JOIN**

## **sql**

## 

## **SELECT users.name, orders.amount**

## **FROM users**

## **LEFT JOIN orders ON users.id = orders.user\_id;**

## 

#### **RIGHT JOIN**

## **sql**

## 

## **SELECT users.name, orders.amount**

## **FROM users**

## **RIGHT JOIN orders ON users.id = orders.user\_id;**

## 

## 

## **8. Aggregation Functions**

## **Used to perform calculations on a set of values.**

### **8.1 COUNT**

## **sql**

## 

## **SELECT COUNT(\*) FROM users;**

## 

### **8.2 AVERAGE**

## **sql**

## 

## **SELECT AVG(age) FROM users;**

## 

### **8.3 SUM**

## **sql**

## 

## **SELECT SUM(amount) FROM orders;**

## 

## 

## **9. Transactions in PostgreSQL**

## **Transactions ensure data integrity by grouping multiple SQL statements together.**

### **9.1 Basic Transaction**

## **sql**

## 

## **BEGIN;**

## **UPDATE users SET age = 35 WHERE name = 'John Doe';**

## **COMMIT;**

## 

### **9.2 Rollback Transaction**

## **sql**

## 

## **BEGIN;**

## **UPDATE users SET age = 35 WHERE name = 'John Doe';**

## **ROLLBACK; -- Undo changes**

## 

## 

## **10. Subqueries**

## **Subqueries allow querying within a query.**

### **10.1 Basic Subquery**

## **sql**

## 

## **SELECT name FROM users WHERE id IN (SELECT user\_id FROM orders WHERE amount > 100);**

## 

### **10.2 Subquery in SELECT**

## **sql**

## 

## **SELECT name, (SELECT COUNT(\*) FROM orders WHERE orders.user\_id = users.id) AS order\_count**

## **FROM users;**

## 

## 

## 

## 

!