Date	04/12/2024	Unit	3

Relational Data Model

Database Management System (DBMS)

Unit 3:
Relational Data Model: Relational model concepts. Characteristics of relations. Relational model constraints: Domain constrains, key constraints, primary & foreign key constraints, integrity constraints and null values. Relational Algebra: Basic Relational Algebra operations. Set theoretical operations on relations. JOIN operations Aggregate Functions and Grouping. Nested Sub Queries-Views. Introduction to PL/SQL & programming of above operations in PL/SQL

What is Data?
Facts, Figures, statistics, observation etc.

What is Information?
Meaningful, relevant, useful

What is DB?
Collection of related data

What is DBMS?
Software , store, access, retrieve , update, create, delete

What is a File System?
A **File System** is a method and data structure that an operating system uses to control how data is stored, organized, retrieved, and managed on storage devices like hard drives, SSDs,

USB drives, etc. It provides a way to organize files into directories (or folders) and manage how data is written and accessed on the disk.

Difference Between File System and DBMS?

A **File System** is simpler and is used primarily for storing and organizing files.
A **DBMS** provides sophisticated tools for data management, including querying, indexing, and ensuring data integrity.

Feature	**File System**	**DBMS**
Data Organization	Hierarchical structure of files and folders	Relational model with tables, rows, and columns
Purpose	Storing and organizing computer files and folders	Managing and manipulating large amounts of structured data
Structure	Hierarchical structure with directories and subdirectories	Relational model with tables, rows, and columns
Data Access	Direct access to files	Access through a query language like SQL
Data Organization	Simple naming conventions	Uses metadata and indexes to organize data
Data Consistency	No built-in mechanisms to ensure data consistency	Enforces data consistency through constraints and rules
Security	Limited security features	Provides robust security features to protect data
Scalability	Limited scalability for large amounts of data	Designed to handle large amounts of data efficiently
Complexity	Simpler to manage and use	More complex to manage and use, requiring specialized skills

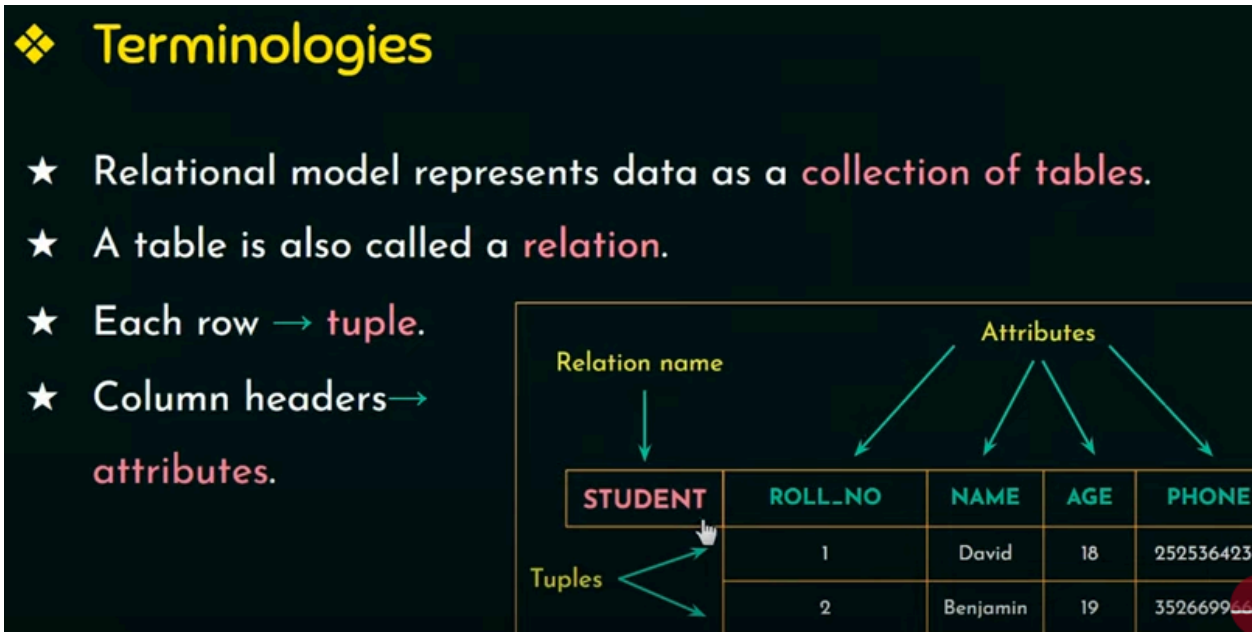
Relational Data Model

The **Relational Data Model** is a widely used data model for managing and organizing data in databases. It represents data as **relations** (tables), where each relation consists of **rows** (tuples) and **columns** (attributes).

Key Concepts of the Relational Model

1. **Relation**: A table with rows and columns.
2. **Tuple**: A single row in a table.
3. **Attribute**: A column in a table, representing a data field.

4. **Domain**: The set of valid values for a particular attribute.



Characteristics of Relations

1. **Unique Name:** Every relation has a specific name.
2. **Ordered Attributes:** Each relation has a set of attributes (columns).
3. **Unordered Tuples:** Each relation has a set of tuples (rows).
4. **Atomic Values:** Each value in a tuple is indivisible.
5. **Distinct Rows:** No two rows can be identical.
6. **Domain Integrity:** Values in a column must belong to a specific data type.
7. **Entity Integrity:** Each relation must have a primary key.
8. **Referential Integrity:** Foreign keys must reference valid primary keys.

Relational Model Constraints

Relational database constraints are rules that ensure data integrity and consistency.

1. **Domain Constraints**: Ensure that attribute values are from a predefined domain.
    - **Example**: An `age` column might restrict values to integers between 0 and 100.
2. **Key Constraints**: Ensure that a set of attributes uniquely identifies a tuple.
    - **Example**: In a `Student` table, the `Student_ID` serves as a primary key.
3. **Primary Key**: A unique identifier for a tuple.
    - **Example**: In the `Employee` table, `Emp_ID` can be the primary key.
4. **Foreign Key**: An attribute in one table that references the primary key in another table.
    - **Example**: `Dept_ID` in `Employee` references `Dept_ID` in `Department`.
5. **Integrity Constraints**:
    - **Entity Integrity**: Primary key cannot be null.

- **Referential Integrity**: Ensures foreign key values match primary key values in the referenced table.

6. **Null Values**: Represent missing or unknown information.

Relational Algebra:

Relational algebra is a formal query language used to manipulate data in relational databases. It provides a set of operations to retrieve and combine data from tables. Here are the basic operations:

Basic Operations:

1. **Selection (σ):**
   - Selects tuples (rows) based on a given condition.
   - Example: `σ Age>25(Student)`: Selects students older than 25.
2. **Projection (π):**
   - Selects specific attributes (columns) from a relation.
   - Example: `π Name, Age(Student)`: Selects only the Name and Age columns.

Set theoretical operations on relations

3. **Union (∪):**
   - Combines two relations with the same schema (same attributes).
   - Example: `Student ∪ Faculty`: Combines students and faculty into one relation.
4. **Intersection (∩):**
   - Finds tuples common to two relations with the same schema.
   - Example: `CS_Students ∩ Math_Students`: Finds students in both CS and Math.
5. **Set Difference (-):**
   - Removes tuples from one relation that are present in another.
   - Example: `All_Students - CS_Students`: Finds students who are not in CS.

Additional Operations:

6. **Cartesian Product (×):**
   - Combines each tuple of one relation with each tuple of another.

    - Example: `Student × Course`: Combines every student with every course.
7. **Natural Join (⋈):**
    - Joins two relations based on a common attribute.
    - Example: `Student ⋈ Enrollment`: Joins students with their enrollments.
8. **Division (÷):**
    - Divides one relation by another, finding tuples that are related to all tuples in the divisor.
    - Example: `Student ÷ Course`: Finds students enrolled in all courses.

Consider two relations:

- **Student:** (StudentID, Name, Age)
- **Enrollment:** (StudentID, CourseID, Grade)

To find the names of students enrolled in the course "Database Systems," you could use the following relational algebra expression:

π Name(σ CourseID = "Database Systems" (Student ⋈ Enrollment))

This expression first joins the `Student` and `Enrollment` relations based on the `StudentID` attribute. Then, it selects tuples where the `CourseID` is "Database Systems." Finally, it projects the `Name` attribute from the resulting relation.

SQL

JOIN operations Aggregate Functions and Grouping

JOIN Operations

JOIN operations combine rows from two or more tables based on a related column between them. This allows you to retrieve information from multiple tables in a single query.

Types of JOINs:

1. **INNER JOIN:** Returns rows that have matching values in both tables.
2. **LEFT JOIN:** Returns all rows from the left table, and the matched rows from the right table.
3. **RIGHT JOIN:** Returns all rows from the right table, and the matched rows from the left table.

4. **FULL OUTER JOIN:** Returns all rows when there is a match in either the left or right table.

Customer Table:

CustomerID	CustomerName	City
1	John Doe	New York
2	Jane Smith	Los Angeles
3	Michael Johnson	Chicago
5	John Doe	New York
6	Jane Smith	Los Angeles

Orders Table:

OrderID	CustomerID	OrderDate	Amount
101	1	2023-11-20	100.00
102	2	2023-11-22	150.50
103	1	2023-11-25	75.25
NULL	NULL	NULL	NULL

```
show databases;
create database XYZ;
show databases;
use XYZ;


-- Create the Customers table
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(50),
    City VARCHAR(30)
);



INSERT INTO Customers (CustomerID, CustomerName, City)
VALUES
    (1, 'John Doe', 'New York'),
    (2, 'Jane Smith', 'Los Angeles'),
    (3, 'Michael Johnson', 'Chicago');



-- Create the Orders table
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE,
```

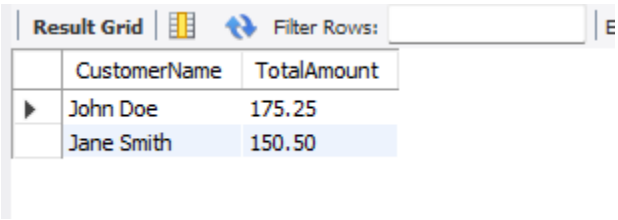
```
    Amount DECIMAL(10,2),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);


select * from Orders;

-- Insert data into the Orders table
INSERT INTO Orders (OrderID, CustomerID, OrderDate, Amount)
VALUES
    (101, 1, '2023-11-20', 100.00),
    (102, 2, '2023-11-22', 150.50),
    (103, 1, '2023-11-25', 75.25);
```

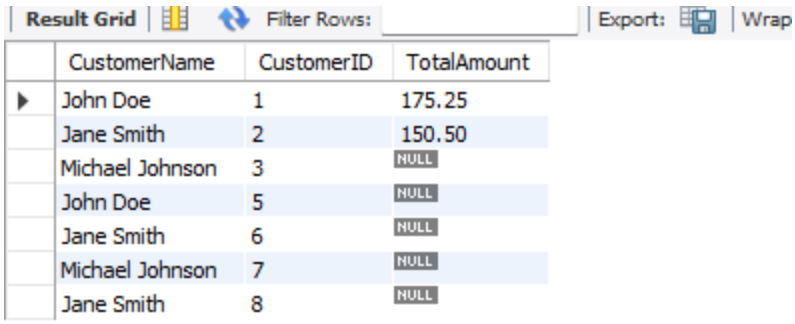
Inner Join:

```
SELECT Customers.CustomerName, SUM(Orders.Amount) AS TotalAmount
FROM Customers
INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID
GROUP BY Customers.CustomerName;
```



Left Join:

```
SELECT Customers.CustomerName, Customers.CustomerID, SUM(Orders.Amount) AS
TotalAmount
FROM Customers
left JOIN Orders ON Customers.CustomerID = Orders.CustomerID
GROUP BY Customers.CustomerID;
```



Result Grid | Filter Rows: | Export: | Wrap

	CustomerName	CustomerID	TotalAmount
▶	John Doe	1	175.25
	Jane Smith	2	150.50
	Michael Johnson	3	NULL
	John Doe	5	NULL
	Jane Smith	6	NULL
	Michael Johnson	7	NULL
	Jane Smith	8	NULL

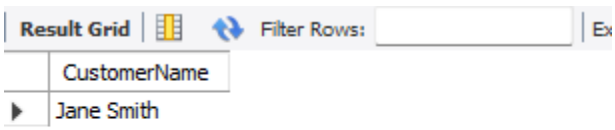
Nested Subqueries

A nested subquery is a query embedded within another query. It's often used to filter data, calculate values, or create temporary result sets.

Types of Nested Subqueries:

1. **Subquery in the WHERE Clause:**
    - Used to filter rows based on conditions involving a subquery's result.

```
SELECT CustomerName
FROM Customers
WHERE CustomerID IN (
   SELECT CustomerID
   FROM Orders
   WHERE Amount > 100
);
```



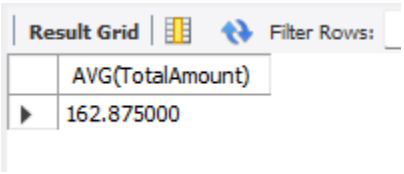
Result Grid | Filter Rows: | Ex

	CustomerName
▶	Jane Smith

Subquery in the FROM Clause:

- Used to treat the result of a subquery as a table.

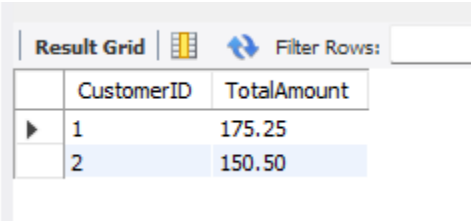
```
SELECT AVG(TotalAmount)
FROM (
   SELECT CustomerID, SUM(Amount) AS TotalAmount
   FROM Orders
   GROUP BY CustomerID
) AS CustomerTotals;
```



Subquery in the HAVING Clause:

- Used to filter groups based on conditions involving subquery results.

```
SELECT CustomerID, SUM(Amount) AS TotalAmount FROM Orders GROUP BY CustomerID
HAVING SUM(Amount) > ( SELECT AVG(Amount) FROM Orders );
```



Types of SQL Commands

SQL commands are categorized into five main types based on their functionality. Each category serves a distinct purpose in managing and manipulating data within a database.

---

1. Data Definition Language (DDL)

DDL commands are used to define or modify the structure of the database objects like tables, schemas, indexes, etc.

Common DDL Commands:

CREATE: Creates a new database object (e.g., table, view).

```
CREATE TABLE Students (
    ID INT PRIMARY KEY,
    Name VARCHAR(50),
    Age INT
);
```

- **ALTER**: Modifies an existing database object.
  `ALTER TABLE Students ADD COLUMN Gender VARCHAR(10);`
- **DROP**: Deletes a database object permanently.
  `DROP TABLE Students;`
- **TRUNCATE**: Removes all records from a table, but keeps the structure.
  `TRUNCATE TABLE Students;`

---

2. Data Manipulation Language (DML)

DML commands are used to manipulate data stored in the database.

Common DML Commands:

- **INSERT**: Adds new data to a table.
  ```
  INSERT INTO Students (ID, Name, Age) VALUES (1, 'Alice', 22);
  ```
- **UPDATE**: Modifies existing data in a table.
  ```
  UPDATE Students SET Age = 23 WHERE ID = 1;
  ```
- **DELETE**: Removes specific data from a table.
  ```
  DELETE FROM Students WHERE ID = 1;
  ```

---

3. Data Query Language (DQL)

DQL commands are used to retrieve data from the database.

Common DQL Command:

SELECT: Fetches data from one or more tables.
```
SELECT * FROM Students;
```

---

4. Transaction Control Language (TCL)

TCL commands manage transactions in a database to ensure data integrity and consistency.

Common TCL Commands:

- **COMMIT**: Saves all changes made in the current transaction.
  ```
  COMMIT;
  ```
- **ROLLBACK**: Undoes changes made in the current transaction.
  ```
  ROLLBACK;
  ```
- **SAVEPOINT**: Creates a point within a transaction to which you can later roll back.
  ```
  SAVEPOINT sp1;
  ```
- **SET TRANSACTION**: Sets the properties of a transaction.
  ```
  SET TRANSACTION READ ONLY;
  ```

---

5. Data Control Language (DCL)

DCL commands control access to the database, ensuring data security.

Common DCL Commands:

- **REVOKE**: Removes specific privileges from users or roles.
  ```
  REVOKE INSERT ON Students FROM user1;
  ```
- **GRANT**: Gives specific privileges to users or roles.
  ```
  GRANT SELECT, INSERT ON Students TO user1;
  ```

Summary of SQL Command Types:

Type	Purpose	Commands
DDL	Define database objects	`CREATE`, `ALTER`, `DROP`, `TRUNCATE`
DML	Manipulate data	`INSERT`, `UPDATE`, `DELETE`
DQL	Query data	`SELECT`
TCL	Manage transactions	`COMMIT`, `ROLLBACK`, `SAVEPOINT`
DCL	Control access	`GRANT`, `REVOKE`

END