DATABASE MANAGEMENT SYSTEMS

MODULE-2

QUESTIONS & ANSWERS

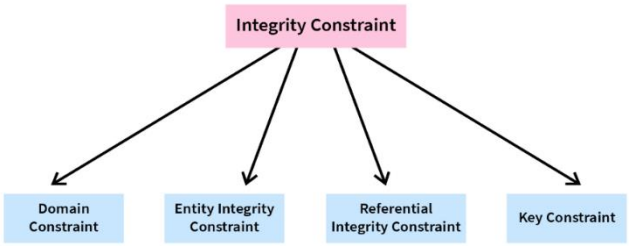
1.Explain Integrity Constraints with suitable examples.

Constraints in DBMS are **rules or conditions** that are applied to the data within a database. They define limitations and requirements that the data must meet, thereby **preventing the entry of invalid or inconsistent data.** The purpose of constraints is to enforce data quality and prevent data inconsistencies, thereby enhancing the overall data integrity and reliability of the database. Every time there is an insertion, deletion, or updating of data in the database it is the responsibility of these integrity constraints to maintain the integrity of data and thus help to prevent accidental damage to the database.

Types of Integrity Constraints

In relational databases, there are different types of constraints. They are as follows:

- **Domain Constraints**
- **Key Constraints**
- **Entity Integrity Constraints**
- **Referential Integrity Constraints**



DOMAIN INTEGRITY CONSTRAINT

A relation schema specifies the domain of each field or column in the relation instance. These **domain constraints** in the schema specify an important condition that **we want each instance of the relation to satisfy: The values that appear in a column must be drawn from the domain associated with that column.** Thus, the **domain of a field** is essentially the **type of that field**, in programming language terms, and **restricts the values** that can appear in the field.

Example:

Consider a Student's table having Student id, Student Name, Marks of students. In the below student's table, the value A in the fourth row last column violates the domain integrity constraint because the Class attribute contains only integer values while A is a character.

Student ID	Student Name	Marks (in %)
1	Guneet	90
2	Ahan	92
3	Yash	87
4	Lavish	90
5	Ashish	79

Student ID	Student Name	Marks (in %)
1	Guneet	90
2	Ahan	92
3	Yash	87
4	Lavish	A
5	Ashish	79

Types of Domain Constraints

Not Null

The values that are not assigned or are unknown can be kept null. These can be the default values for an attribute if the answer is not known.

For instance, in the above table, if we don't know the marks of the student with student id=1, we can keep null in the marks attribute. However, certain attributes cannot be null. For instance, the Student ID is a must-known value and it can never be null as we can identify each student uniquely only with the help of Student ID. So, we can apply a "not null" constraint on the Student ID attribute in the above table.

```
Create table Students
(Student_id NUMBER not null,
Student_name varchar(30),
marks NUMBER);
```

Not Null after Table Creation

```
ALTER TABLE students
ALTER COLUMN first_name SET NOT NULL,
ALTER COLUMN last_name SET NOT NULL,
ALTER COLUMN email SET NOT NULL;
```

Check

This constraint is used for specifying range of values for a particular column of a table. When this constraint is being set on a column, it ensures that the specified column must have the value falling in the specified range.

Let us say we have a class of students. Now, the school decides that only the students with marks greater than 35% will be declared qualified in the current class. Also, they decide that only the results of those students will be entered in the table who have passed the class. So, they want to check whether the marks of a student are greater than 35% or not. If not, that student's data will not be entered in the table. So, for this kind of constraint application, the Domain Constraint – Check is used.

```
Create table Students
(Student_id NUMBER not null,
Student_name varchar(30),
marks NUMBER check(marks > 35))
```

Unique

UNIQUE Constraint enforces a column or set of columns to have unique values. If a column has a unique constraint, it means that particular column cannot have duplicate values in a table.

```
CREATE TABLE STUDENT(
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL UNIQUE,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (35) UNIQUE,
PRIMARY KEY (ROLL_NO)
);
```

UNIQUE on multiple attributes

```
CREATE TABLE courses (
    course_id INT,
    course_name TEXT,
    department TEXT,
    UNIQUE (course_name, department)
);
```

Default

The DEFAULT constraint provides a default value to a column when there is no value provided while inserting a record into a table.

```
CREATE TABLE STUDENT(
ROLL_NO    INT   NOT NULL,
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
EXAM_FEE INT   DEFAULT 10000,
STU_ADDRESS VARCHAR (35) ,
PRIMARY KEY (ROLL_NO)
);
```

KEY CONSTRAINTS

A key constraint is a statement that a certain minimal subset of the fields of a relation is a unique identifier for a tuple. Types of Key Constraints

- ❑ PRIMARY KEY
- ❑ FOREIGN KEY

Primary Key

Primary key uniquely identifies each record in a table. It must have unique values and cannot contain nulls. In the below example the ROLL_NO field is marked as primary key, that means the ROLL_NO field cannot have duplicate and null values.

```
CREATE TABLE STUDENT(
ROLL_NO    INT   NOT NULL,
STU_NAME VARCHAR (35)   NOT NULL UNIQUE,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (35) UNIQUE,
PRIMARY KEY (ROLL_NO)
);
```

```
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    first_name TEXT,
    last_name TEXT,
    email TEXT,
    major TEXT,
    enrollment_year INT
);
```

Foreign Key

Foreign Key is a field in a table which uniquely identifies each row of another table. That is, this field points to primary key of another table. This usually creates a kind of link between the tables.

```
CREATE TABLE enrollments (
    student_id INT,
    course_id INT,
    ...
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
```

ENTITY INTEGRITY CONSTRAINTS

Entity Integrity Constraint is used to ensure that the primary key cannot be null. A primary key is used to identify individual records in a table and if the primary key has a null value, then we can't identify those records. There can be null values anywhere in the table except the primary key column.

Example: Consider Employees table having Id, Name, and salary of employees. In the below employee's table, we can see that the ID column is the primary key and contains a null value in the last row which violates the entity integrity constraint.

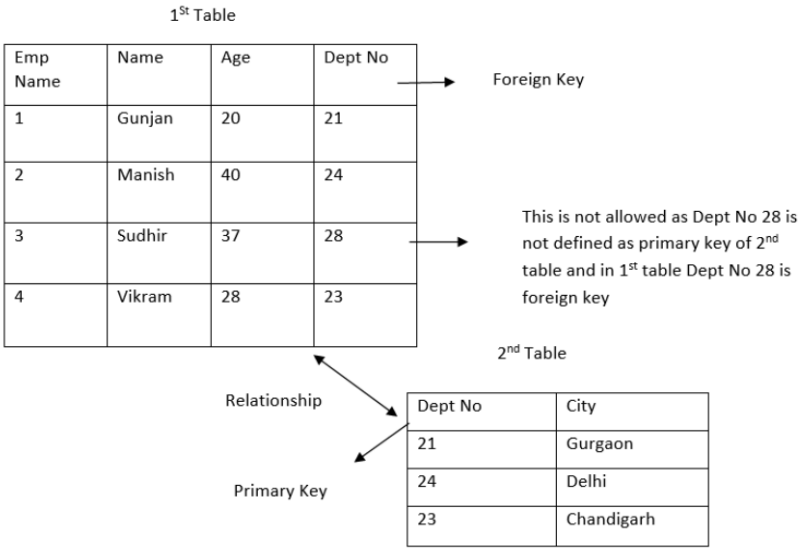
EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

REFERENTIAL INTEGRITY CONSTRAINT

Referential Integrity Constraint is specific between two tables. In case of referential integrity constraints, if a Foreign key in Table 1 refers to Primary key of Table 2 then every value of the Foreign key in Table 1 must be null or available in Table 2.



---

2. Explain Relational Algebra fundamentals with suitable example.

Relational algebra is one of the two formal query languages associated with the relational model. Queries in relational algebra are composed using a collection of operators, and each query describes a step-by-step procedure for computing the desired answer.

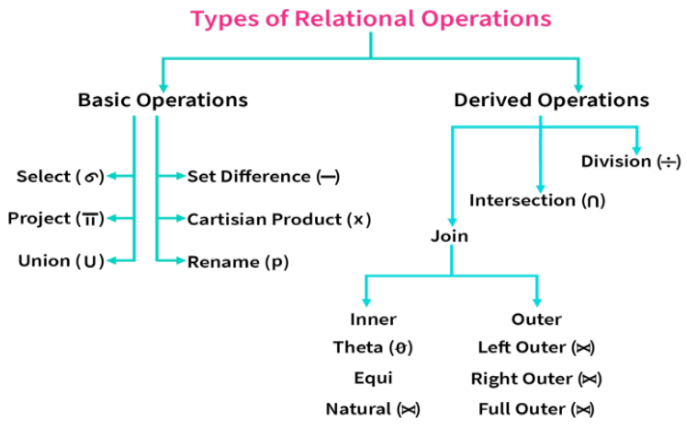
A fundamental property is that **every operator** in the algebra **accepts (one or two) relation instances** as arguments and **returns a relation instance as the result.** A **relational algebra expression** is recursively defined to be a relation, a unary algebra operator applied to a single expression, or a binary algebra operator applied to two expressions. This property makes it easy to **compose operators to form a complex query.**

Types of Relational Operations

In Relation Algebra, we have two types of Operations.

- ❑ Basic Operations
- ❑ Derived Operations

Applying these operations over relations/tables will give us new relation as output.



Select (σ)

Select operation chooses the subset of tuples from the relation that satisfies the given condition mentioned in the syntax of selection. The selection operation is also known as horizontal partitioning since it partitions the table or relation horizontally. Select operation is done by Selection Operator which is represented by "sigma"(σ). It is used to retrieve tuples (rows) from the table where the given condition is satisfied.

It is a **unary operator** means it require only one operand.

Notation: σ p(R)

Where σ is used to represent SELECTION, R is used to represent RELATION, p is the logic formula.

Example:
We can retrieve rows corresponding to expert sailors by using the operator σ. The subscript rating>8 specifies the selection criterion to be applied while retrieving tuples.

$$\sigma_{rating>8}(S2)$$

Project (*π*)

Project operation is done by Projection Operator which is represented by "pi"(***π***). It is used to retrieve certain attributes(columns) from the table. It is also known as vertical partitioning as it separates the table vertically. It is also a **unary operator**.

Notation: *π*a(r)

Where ***π*** is used to represent PROJECTION, r is used to represent RELATION, a is the attribute list
For example, we can find out all sailor names and ratings by using ***π***.
The expression $\pi_{sname,rating}(S2)$ evaluates to the relation S2.

Union (∪)

Union operation is done by Union Operator which is represented by "union"(∪). It is the same as the union operator from set theory, i.e., **it selects all tuples from both relations but with the exception that for the union of two relations/tables both relations must have the same set of Attributes**.

It is a **binary operator** as it requires two operands. Notation: R ∪ S
Where R is the first relation, S is the second relation. The relations R and S must be **union-compatible**, and the schema of the result is defined to be **identical to the schema of R.**

Intersection (∩)

R ∩ S returns a relation instance containing all tuples that **occur in both R and S**. The relations R and S must be **union-compatible**, and the schema of the result is defined to be **identical to the schema of R.**

Set Difference (-)

Set Difference as its name indicates is the difference of two relations (R-S). It is denoted by a "Hyphen"(-) and it returns all the tuples (rows) which are in relation R but not in relation S. It is also a **binary operator**.

Notation: R – S

Where R is the first relation, S is the second relation

Cartesian Product (X)

Cartesian product is denoted by the "X" symbol. Let's say we have two relations R and S. Cartesian product will combine every tuple (row) from R with all the tuples from S.

Notation: R X S

Where R is the first relation, S is the second relation. As we can see from the notation it is also a **binary operator**.

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

Figure Instance $S1$ of Sailors

sid	*bid*	*day*
22	101	10/10/96
58	103	11/12/96

Figure Instance $R1$ of Reserves

(*sid*)	*sname*	*rating*	*age*	(*sid*)	*bid*	*day*
22	Dustin	7	45.0	22	101	10/10/96
22	Dustin	7	45.0	58	103	11/12/96
31	Lubber	8	55.5	22	101	10/10/96
31	Lubber	8	55.5	58	103	11/12/96
58	Rusty	10	35.0	22	101	10/10/96
58	Rusty	10	35.0	58	103	11/12/96

Figure $S1 \times R1$

Rename (ρ)

Rename operation is denoted by "Rho"(ρ). The result of a relational algebra expression inherits field names from its argument (input) relation instances. However, name conflicts can arise in some cases; for example, in **S1 X R1**. It is therefore convenient to be able **to give names explicitly** to the fields of a relation instance that is **defined by a relational algebra expression.** In fact, it is often convenient to give the instance itself a name so that we can **break a large algebra expression into smaller pieces** by giving names to the results of subexpressions.

We introduce a **renaming operator ρ** for this purpose. The expression **ρ(R(F),E)** takes an arbitrary relational algebra expression **E** and returns an instance of a (new) relation called R. R contains the **same tuples** as the result of E, and has the **same schema** as E, but some **fields are renamed.**

For example, the expression $\rho(C(1 \rightarrow sid1, 5 \rightarrow sid2),\ S1 \times R1)$

returns a relation that contains the tuples shown in Figure and has the following schema:

C(sid1:integer, sname: string, rating: integer, age: real, sid2: integer, bid: integer, day: dates)

---

3. Distinguish between Primary Key and Foreign Key with example.

Keys are the fundamental elements for constructing a relationship between two tables. They are very useful in the maintenance of a relational database structure. The main difference between them is that the primary key identifies each record in the table, whereas the foreign key is used to link two tables together.

Primary Key

The primary key is a unique or not-null key that **uniquely identifies every record** in a table or relation. Each database needs a unique identifier for every row of a table, and the primary key plays a vital role in identifying rows in the table uniquely. The primary key column can't store duplicate values. It is also called a minimal super key; therefore, we cannot specify more than one primary key in any relationship.

CREATE TABLE <Table_Name> (Column1 datatype, Column2 data type, PRIMARY KEY (Column name));

Example: CREATE TABLE Persons (
ID number NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age number,
PRIMARY KEY (ID));

Foreign Key

The foreign key is a group of one or more columns in a database to uniquely identify another database record in some other table to maintain the referential integrity. It is also known as the **referencing key that establishes a relationship** between two different tables in a database. A foreign key always matches the primary key column in another table. It means a foreign key column in one table refers to the primary key column of another table. A foreign key is beneficial in relational database normalization, especially when we need to access records from other tables.

Syntax:

CREATE TABLE tablename (
column1 datatype,
column2 datatype,
column3 datatype,
PRIMARY KEY (columnname1),
FOREIGN KEY (columnname) REFERENCES Tablename(columnname));

Key differences between Primary Key and Foreign Key

- A primary key constraint in the relational database acts as a unique identifier for every row in the table. In contrast, a foreign key constraint establishes a relationship between two different tables to uniquely identify a row of the same table or another table.

- The primary key column does not store NULL values, whereas the foreign key can accept more than one NULL value.
- Each table in a relational database can't define more than one primary key while we can specify multiple foreign keys in a table.
- We can't remove the parent table's primary key value, which is referenced with a foreign key column in the child table. In contrast, we can delete the child table's foreign key value even though they refer to the parent table's primary key.
- A primary key is a unique and non-null constraint, so no two rows can have identical values for a primary key attribute, whereas foreign key fields can store duplicate values.
- We can insert the values into the primary key column without any limitation. In contrast, we need to ensure that the value is present in a primary key column while inserting values in the foreign key table.
- We can implicitly define the primary key constraint on temporary tables, whereas we cannot enforce foreign key constraints on temporary tables.

4.Discuss about Views in Relational Model with example.

Writing Complex queries and Securing Database access is very challenging for any Database Developer and Database Administrator. So those types of queries can be simplified to proxy data or virtual data which simplifies the queries.

Views in SQL are kind of virtual tables. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain condition. In this we will learn about creating, deleting and updating Views.

Suppose, the user needs only 2 columns of data, so instead of giving him access to the whole table in the database, the Database Administrator can easily create a virtual table of 2 columns which the user needs using the views. This will not give full access to the table and the user is only seeing the projection of only 2 columns and it keeps the database secure.

Views contain only the definition of the view data in the data dictionary, not the actual data. The view has two primary purposes:

1.Simplifying complex SQL queries.
2.Restricting users from accessing sensitive data.

CREATE VIEWS in SQL

For creating a view (simple and complex views), updating and deleting the views we will take some sample data and tables that store the data.

Let's take an employee table that store data of employees in a particular company:

Employee Table: This table contains details of employees in a particular company and has data fields such as EmpID, EmpName, Address, Contact. We have added 6 records of employees for our purpose.

EmpID	EmpName	Address	Contact
1	Alex	London	05-12421969
2	Adolf	San Francisco	01-12584365
3	Aryan	Delhi	91-9672367287
4	Bhuvan	Hyderabad	91-9983288383
5	Carol	New York	01-18928992
6	Steve	California	01-13783282

The view can be created by using the CREATE VIEW statement, Views can be simple or complex depending on their usage.

Syntax:

CREATE VIEW veiwName AS SELECT column1, column2,.... FROM tableName WHERE condition;

Here, view Name is the Name for the View we set, tableName is the Name of the table and condition is the Condition by which we select rows.

Example 1: In this example, we are creating a view table from Employee Table for getting the EmpID and EmpName. So the query will be:

Syntax: CREATE VIEW EmpView1 AS SELECT EmpID, EmpName FROM Employee;

Now to see the data in the EmpView1 view created by us, We have to simply use the SELECT statement. SELECT * from EmpView1;

Output: The view table EmpView1 that we have created from Employee Table contains EmpID and EmpName as its data fields.

EmpID	EmpName
1	Alex
2	Adolf
3	Aryan
4	Bhuvan
5	Carol
6	Steve

Creating a Complex View

The complex view is the view that is made from multiple tables, It takes multiple tables in which data is accessed using joins or cross products, It contains inbuilt SQL functions like AVG(), MIN(), MAX() etc, or GROUP BY clause. So, whenever we need to access data from multiple tables we can make use of Complex Views.

Example: In this example, we are creating a view table using Employee Table and EmpRole table for getting the EmpName from Employee table and Dept from EmpRole table. So the query will be:

```
CREATE VIEW CompView AS SELECT Employee.EmpName, EmpRole.Dept
FROM Employee, EmpRole WHERE Employee.EmpID = EmpRole.EmpID;

SELECT * from CompView;
```

Output: The view table CompView that we have created from Employee Table and EmpRole Table contains EmpName and Dept as its data fields. We have used the cross join to get our necessary data.

EmpName	Dept
Alex	Engineering
Adolf	IT
Aryan	HR
Bhuvan	Engineering
Carol	Engineering
Steve	Engineering

Deleting a View

If we need to delete the view so as we DROP a table in SQL, similarly, we can delete or drop a view using the DROP statement. The DROP statement completely deletes the structure of the view.

Syntax: DROP VIEW ViewName;

Example:

Let's delete one of our created views, say EmpView1.

```
DROP VIEW EmpView1;
```

Updating a View

Suppose we want to add more columns to the created view so we will have to update the view. For updating the views we can use CREATE OR REPLACE VIEW statement, new columns will replace or get added to the view.

Syntax: CREATE OR REPLACE VIEW ViewName AS SELECT column1,coulmn2,.. FROM TableName WHERE condition;

For Updating a view CREATE OR REPLACE statement is used. Here, viewName is the Name of the view we want to update, tableName is the Name of the table and condition is the Condition by which we select rows.

Example 1:

let's take the above created simple view EmpView1 and we want to one more column of address to our EmpView1 from Employee Table.

Syntax: CREATE OR REPLACE VIEW EmpView1 AS SELECT EmpID, EmpName, Address FROM Employee;

Output:

The data fields of view table EmpView1 have been updated to EmpID, EmpName, and Address.

EmpID	EmpName	Address
1	Alex	London
2	Adolf	San Francisco
3	Aryan	Delhi
4	Bhuvan	Hyderabad
5	Carol	New York
6	Steve	California

---

5. Describe the usage of null values. Compare various SQL operations with and without null values.

Null values play a crucial role in database management by representing the **absence of data** in a particular field. A **null value** is a special marker used in databases to indicate that a **data field is empty or missing.** In other words, a null value signifies the **lack of a value** or the **unknown value.** One of the key uses of null values in database management is to allow flexibility in data entry. By allowing fields to be left empty or contain null values, databases can accommodate different levels of data completeness.

Typically, it can have one of three interpretations:

1. **Value Unknown:** The value exists but is not known.
2. **Value Not Available:** The value exists but is intentionally withheld.
3. **Attribute Not Applicable:** The value is undefined for a specific record.

For example, a customer database may have fields for a customer's phone number or email address, but not all customers may provide this information. In such cases, using null values allows for capturing only the information that is available without forcing the entry of unnecessary or irrelevant data.

Another important role of null values in database management is to distinguish between a true value and the absence of a value. Without null values, it would be challenging to differentiate between a valid data entry of, for example, "0" and an empty data field. In conclusion, null values are an essential component of effective database management. They provide flexibility in data entry, help differentiate between the absence of a value and a valid value, and contribute to maintaining data integrity and consistency.

Example: Employee Table

```
CREATE TABLE Employee (
  Fname VARCHAR(50),
  Lname VARCHAR(50),
  SSN VARCHAR(11),
  Phoneno VARCHAR(15),
  Salary FLOAT
);

INSERT INTO Employee (Fname, Lname, SSN, Phoneno, Salary)
VALUES
  ('Shubham', 'Thakur', '123-45-6789', '9876543210', 50000.00),
  ('Aman', 'Chopra', '234-56-7890', NULL, 45000.00),
  ('Aditya', 'Arpan', NULL, '8765432109', 55000.00),
  ('Naveen', 'Patnaik', '345-67-8901', NULL, NULL),
  ('Nishant', 'Jain', '456-78-9012', '7654321098', 60000.00);

Select * FROM Employee;
```

Output

Fname	Lname	SSN	Phoneno	Salary
Shubham	Thakur	123-45-6789	9876543210	50000
Aman	Chopra	234-56-7890		45000
Aditya	Arpan		8765432109	55000
Naveen	Patnaik	345-67-8901		
Nishant	Jain	456-78-9012	7654321098	60000

Employee Table

The IS NULL Operator

In this query, it retrieves the Fname and Lname of employees whose SSN is NULL. Since SSN represents a unique identifier, rows with NULL in this column indicate missing data. This query helps identify records that lack this essential information.

Table: employee

id	name	department	Salary
1	John	HR	5000
2	Sarah	Finance	NULL
3	Mike	NULL	7000
4	NULL	IT	NULL

Examples:

- Query: SELECT * FROM employee WHERE salary = NULL;
  - **Result**: No rows returned (because NULL = NULL is UNKNOWN).
- Query: SELECT * FROM employee WHERE salary IS NULL;
  - **Result**: Returns rows with IDs 2 and 4.

Without Null Values:

- Query: SELECT AVG(salary) FROM employee;
  - If all salaries were defined: Computes the average of all rows.
- Query: SELECT COUNT(*) FROM employee;
  - Counts all rows regardless of null presence.

The IS NOT NULL Operator

In this query, it counts the number of employees who have a valid SSN by excluding rows where SSN is NULL. The result provides the total number of employees with an SSN present in the table. The COUNT(*) function ensures that all non-NULL rows are included in the count.

Query: SELECT COUNT(*) AS Count FROM Employee WHERE SSN IS NOT NULL;

---

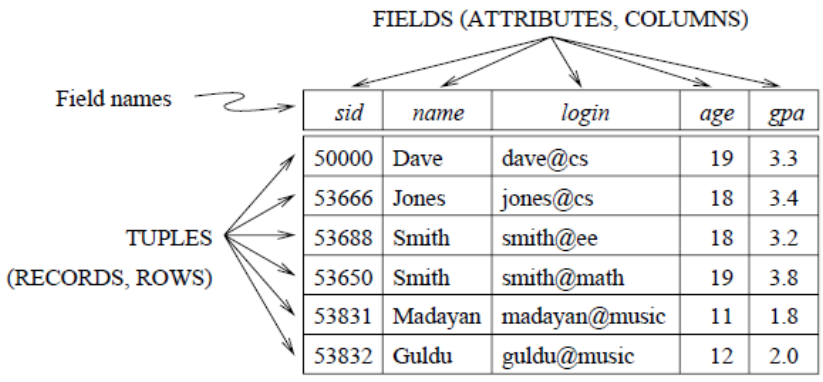
6. Write notes on Relational Model with an Example

The relational model is very simple and elegant. A database is a collection of one or more relations, where each relation is a table with rows and columns. The relational model represents DB in the form of a collection of various relations. This relation refers to a table of various values. And every row present in the table happens to denote some real-world entities or relationships and every column represent properties of an entity. This data gets represented in the form of a set of various relations. Thus, in the relational model, basically, this data is stored in the form of tables.

This simple tabular representation enables even novice users to understand the contents of a database, and it permits the use of simple, high-level languages to query the data. The **major advantages** of the relational model over the older data models are its **simple data representation** and the **ease with which even complex queries can be expressed.**

The main construct for representing data in the relational model is a relation. A relation consists of a **relation schema** and a **relation instance**. The **relation instance** is a table, and the **relation schema** describes the column heads for the table.

First describe the relation schema and then the relation instance. The schema specifies the relation's name, the name of each field (or column, or attribute), and the domain of each field. A domain is referred to in a relation schema by the domain name and has a set of associated values. A relational model represents how we can store data in Relational Databases. Here, a relational database stores information in the form of relations or tables.



sid	*name*	*login*	*age*	*gpa*
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Figure An Instance $S1$ of the Students Relation

Important Terminologies

Here are some Relational Model concepts in DBMS:

- **Attribute:** It refers to every column present in a table. The attributes refer to the properties that help us define a relation. E.g., Employee_ID, Student_Rollno, SECTION, NAME, etc.
- **Tuple:** It is a single row of a table that consists of a single record. The relation above consists of four tuples, one of which is like:

  ex: C1 RIYA DELHI 15 20
- **Degree:** It refers to the total number of attributes that are there in the relation. The EMPLOYEE relation defined here has degree 5.
- **Relation Schema:** It represents the relation's name along with its attributes. E.g., EMPLOYEE (ID_NO, NAME, ADDRESS, ROLL_NO, AGE) is the relation schema for EMPLOYEE. If a schema has more than 1 relation, then it is known as Relational Schema.
- **Cardinality:** It refers to the total number of rows present in the given table. The EMPLOYEE relation defined here has cardinality 4.
- **Attribute Domain:** Every attribute has some defined value and scope, which is known as the attribute domain.
- **NULL Values:** The value that is NOT known or the value that is unavailable is known as a NULL value. This null value is represented by the blank spaces. E.g., the MOBILE of the EMPLOYEE having ID_NO 4 is NULL.
- **Relational Instance:** It is the collection of records present in the relation at a given time. . It refers to a finite set of tuples present in the RDBMS system. A relation instance never has duplicate tuples.
- **Relation key:** Each and every row consists of a single or multiple attributes. It can identify the row in the relation uniquely. It is known as a relation key.

Properties of a Relational Model

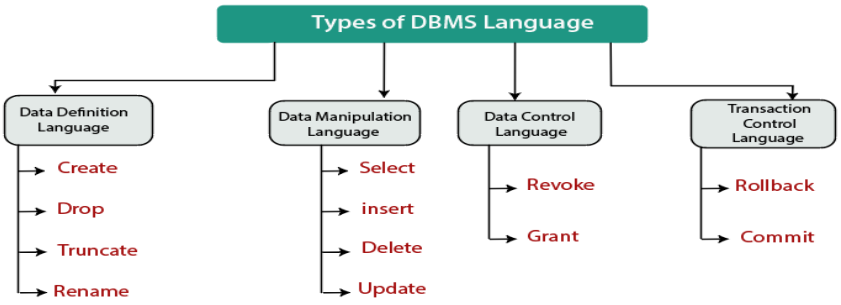
The relational databases consist of the following properties:

- Every row is unique
- All of the values present in a column hold the same data type
- Values are atomic
- The columns sequence is not significant
- The rows sequence is not significant
- The name of every column is unique

7. Write a short note on DDL and DML Commands with example.

A DBMS has appropriate languages and interfaces to express database queries and updates. And they can be used to read, store and update the data in the database.The different types of DBMS languages are as follows:

- ❑ Data Definition Language
- ❑ Data Manipulation Language
- ❑ Data Control Language
- ❑ Transaction Control Language



DATA DEFINITION LANGUAGE

The **set of relations in a database** must be specified to the system by means of a Data Definition Language (DDL).

The SQL DDL allows specification of not only a set of relations, but **also information about each relation**, including:

- ◍ The **schema** for each relation.
- ◍ The **types of values** associated with each attribute.
- ◍ The **integrity constraints.**
- ◍ The **set of indices** to be maintained for each relation.
- ◍ The **security and authorization** information for each relation.
- ◍ The **physical storage structure** of each relation on disk.

The basic DDL commands are

CREATE

Create is used to create schema, tables, index, and constraints in the database. The basic syntax to create table is as follows.

Syntax: CREATE TABLE tablename (Column1 DATATYPE, Column2 DATATYPE, … ColumnN DATATYPE);

The general form of the **create table** command is:

create table r
$(A_1 \ D_1,$
$A_2 \ D_2,$
$\ldots,$
$A_n \ D_n,$
$\langle \text{integrity-constraint}_1 \rangle,$
$\ldots,$
$\langle \text{integrity-constraint}_k \rangle);$

For example: Create employee table
CREATE TABLE Employee (eid number(5),ename varchar2(20),salary number(6));

ALTER

Suppose we have created a STUDENT table with his ID and Name. Later we realize that this table should have his address and Age too. We will add the column to the existing table by the use of ALTER command.

This command is used to modify the structure of the Schema or table. It can even used to add, modify or delete the columns in the table. The syntax for alter statement is as follows.

Syntax:

To add a new column: ALTER TABLE table_name ADD column_name *datatype*;

Ex: ALTER TABLE employee ADD department varchar(50);

To delete a column: ALTER TABLE table_name DROP COLUMN column_name;

Ex: ALTER TABLE Employee DROP COLUMN salary;

To modify a column: ALTER TABLE table_name MODIFY column_name *datatype*;

To rename table: ALTER TABLE table_name RENAME TO new_table_name;
To rename the column: ALTER TABLE table_name RENAME COLUMN old_Column_name to new_Column_name

TRUNCATE

This command will remove the data permanently. But structure will not be removed.

Syntax: TRUNCATE TABLE <Table name>

Example TRUNCATE TABLE EMP1;

DROP

This is used to delete the structure of a relation. It permanently deletes the records in the table.

Syntax:	**Example:**
DROP TABLE relation_name;	SQL>DROP TABLE Student; **Result:** Table dropped.

DATA MANIPULATION LANGUAGE

When we have to insert records into table or get specific record from the table, or need to change some record, or delete some record or perform any other actions on records in the database, we need to have some media to perform it. DML helps to handle user requests. It helps to insert, delete, update, and retrieve the data from the database.
Following are the four main commands in DML:

- **Select**
- **Insert**
- **Update**
- **Delete**

Select

Select command helps to pull the records from the tables or views in the database. It either pulls the entire data from the table/view or pulls specific records based on the condition. It can even retrieve the data from one or more tables/view in the database.

Syntax: SELECT * FROM table_name;

Insert

Insert statement is used to insert new records into the table. The general syntax for insert is as follows:

Syntax: INSERT INTO TABLE_NAME (col1, col2, col3,...colN) VALUES (value1, value2, value3,...valueN);

Update

Update statement is used to modify the data value in the table.

Syntax: UPDATE relation_name SET Field_name1=data,field_name2=data,
WHERE field_name=data;

Example: SQL>UPDATE emp1
SET ename = 'kumar'
WHERE empno=1;

Delete

Using Delete statement, we can delete the records in the entire table or specific record by specifying the condition. Using DELETE statement, we can delete few records by specifying the condition. If we do not specify the condition, it deletes entire records in table. Whereas TRUNCATE deletes all the records in the table.

Syntax: DELETE FROM relation_name WHERE condition;

Example: SQL>DELETE FROM Stu1 WHERE empno = 2;

---

8. Explain Basic Form of SQL Query.

Let us see the syntax of a simple SQL query through a conceptual evaluation strategy. A conceptual evaluation strategy is a way to evaluate the query that is intended to be easy to understand, rather than efficient. A DBMS would typically execute a query in a different and more efficient way.

The basic form of an SQL query is as follows:

```
SELECT   [ DISTINCT ] select-list
FROM     from-list
WHERE    qualification
```

Every query must have

- a SELECT clause, which specifies columns to be retained in the result, and
- In cases where we want to force the elimination of duplicates, we insert the keyword distinct after select.

- a FROM clause, which specifies a cross-product of tables.
- the optional WHERE clause specifies selection conditions on the tables mentioned in the FROM clause.

Conceptual Evaluation Strategy

1. Compute the cross-product of the tables in the **from-list.**
2. Delete those rows in the cross-product that **fail the qualification conditions.**
3. Delete all columns that do not appear in the **select-list.**
4. If **DISTINCT** is specified, eliminate duplicate rows.

Example:

(Q1) Find the names of sailors who have reserved boat number 103.

It can be expressed in SQL as follows.

```
SELECT S.sname
FROM   Sailors S, Reserves R
WHERE  S.sid = R.sid AND R.bid=103
```

The first step is to construct the cross-product $S4 \times R3$, which is shown in Figure

sid	*bid*	*day*
22	101	10/10/96
58	103	11/12/96

Figure Instance $R3$ of Reserves

sid	*sname*	*rating*	*age*
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Figure Instance $S4$ of Sailors

sid	*sname*	*rating*	*age*	*sid*	*bid*	*day*
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Figure $S4 \times R3$

sname
rusty

Figure Answer to Query Q1 on R3 and S4

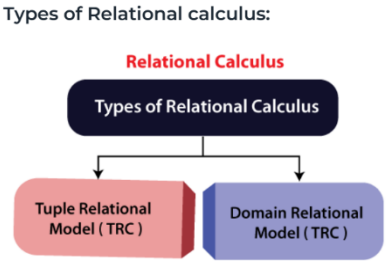
---

9. Explain Relational Calculus with examples.

Relational calculus is a non-procedural query language. In the non-procedural query language, the user is concerned with the details of how to obtain the end results. The relational calculus tells what to do but never explains how to do.

Relational calculus has had a big influence on the design of commercial query languages such as **SQL** and, especially, **Query-by-Example (QBE).** The variant of the calculus that we present in detail is called the Tuple Relational Calculus (TRC) and Domain Relational Calculus (DRC).

- Variables in Tuple Relational Calculus (TRC) take on tuples as values. TRC has had more of an influence on SQL.
- In the Domain Relational Calculus (DRC), the variables range over field values. DRC has strongly influenced QBE.



Tuple Relational Calculus (TRC)

It is a non-procedural query language which is based on finding a number of tuple variables also known as range variable for which predicate holds true. It describes the desired information without giving a specific procedure for obtaining that information. The tuple relational calculus is specified to select the tuples in a relation. In TRC, filtering variable uses the tuples of a relation. The result of the relation can have one or more tuples.

Notation:
A Query in the tuple relational calculus is expressed as following notation
{T | P (T)} or {T | Condition (T)}

Where **T** is the resulting tuples, **P(T)** is the condition used to fetch T.

For Example:
{T.name | Author(T) AND T.article = 'database' }

Output: This query selects the tuples from the AUTHOR relation. It returns a tuple with 'name' from Author who has written an article on 'database'.

Table Loan

Loan number	Branch name	Amount
L33	ABC	10000
L35	DEF	15000
L49	GHI	9000
L98	DEF	65000

Example 1: Find the loan number, branch, and amount of loans greater than or equal to 10000 amount.
{t| t ∈ loan ∧ t[amount]>=10000}

Resulting relation:

Loan number	Branch name	Amount
L33	ABC	10000
L35	DEF	15000
L98	DEF	65000

In the above query, t[amount] is known as a tuple variable.

Domain Relational Calculus (DRC)

The second form of relation is known as Domain relational calculus. In domain relational calculus, filtering variable uses the domain of attributes. Domain relational calculus uses the same operators as tuple calculus. It uses logical connectives ∧ (and), ∨ (or) and ¬ (not). It uses Existential (∃) and Universal Quantifiers (∀) to bind the variable. The Query by example is a query language related to domain relational calculus.

Notation:
{ a1, a2, a3, ..., an | P (a1, a2, a3, ... ,an)}

Where **a1, a2** are attributes, **P** stands for formula built by inner attributes

For example:
{< article, page, subject > | ∈ course ∧ subject = 'database'}

Output: This query will yield the article, page, and subject from the relational course, where the subject is a database.

Table-2: Loan

Loan number	Branch name	Amount
L01	Main	200
L03	Main	150
L10	Sub	90
L08	Main	60

Query-1: Find the loan number, branch, amount of loans of greater than or equal to 100 amount.

$\{\prec l, b, a\succ \mid \prec l, b, a\succ \in \text{loan} \wedge (a \geq 100)\}$

Resulting relation:

Loan number	Branch name	Amount
L01	Main	200
L03	Main	150

---

10. Discuss about Derived operations in Relational Algebra with example.

Relational algebra is one of the two formal query languages associated with the relational model. Queries in relational algebra are composed using a collection of operators, and each query describes a step-by-step procedure for computing the desired answer. Derived Operations are also known as extended operations, these operations can be derived from basic operations hence named Derived Operations. These include three operations: Join Operations, Intersection operation, and Division operation.

INTERSECTION

- R ∩ S returns a relation instance containing all tuples that **occur in both R and S**.
- The relations R and S must be **union-compatible**, and the schema of the result is defined to be **identical to the schema of R.**

JOIN OPERATIONS

The **join operation** is one of the most useful operations in relational algebra and is the most commonly used way to **combine information from two or more relations.** Although a join can be defined as a **cross-product followed by selections and projections,** joins arise much more frequently in practice than plain cross-products.

Conditional Joins

The most general version of the join operation accepts **a join condition c** and a **pair of relation instances as arguments**, and returns a **relation instance.** The join condition is identical to a selection condition in form. The operation is defined as follows:

$$R \bowtie_c S = \sigma_c(R \times S)$$

Thus ⋈ is defined to be a **cross-product followed by a selection.** Note that the condition c can refer to attributes of both R and S.

As an example, the result of $S1 \bowtie_{S1.sid<R1.sid} R1$

(sid)	sname	rating	age	(sid)	bid	day
22	Dustin	7	45.0	22	101	10/10/96
22	Dustin	7	45.0	58	103	11/12/96
31	Lubber	8	55.5	22	101	10/10/96
31	Lubber	8	55.5	58	103	11/12/96
58	Rusty	10	35.0	22	101	10/10/96
58	Rusty	10	35.0	58	103	11/12/96

Figure $S1 \times R1$

(sid)	sname	rating	age	(sid)	bid	day
22	Dustin	7	45.0	58	103	11/12/96
31	Lubber	8	55.5	58	103	11/12/96

Figure $S1 \bowtie_{S1.sid<R1.sid} R1$

Equi Join

A common special case of the join operation R ⋈ S is when the join condition consists solely of equalities of the form

R:name1 = S:name2, that is, equalities between two fields in R and S.

In this case, obviously, there is some redundancy in retaining both attributes in the result. For join conditions that contain only such equalities, the join operation is refined by doing an additional projection in which S:name2 is dropped. The join operation with this refinement is called equijoin.

(sid)	sname	rating	age	(sid)	bid	day
22	Dustin	7	45.0	22	101	10/10/96
22	Dustin	7	45.0	58	103	11/12/96
31	Lubber	8	55.5	22	101	10/10/96
31	Lubber	8	55.5	58	103	11/12/96
58	Rusty	10	35.0	22	101	10/10/96
58	Rusty	10	35.0	58	103	11/12/96

Figure $S1 \times R1$

sid	sname	rating	age	bid	day
22	Dustin	7	45.0	101	10/10/96
58	Rusty	10	35.0	103	11/12/96

Figure $S1 \bowtie_{R.sid=S.sid} R1$

Natural Join

A comparison operator is not used in a natural join. It does not concatenate like a Cartesian product. A Natural Join can be performed only if two relations share at least **one common attribute**. Furthermore, the attributes **must share the same name and domain**. Natural join operates on matching attributes where the values of the attributes in both relations are the same and **remove the duplicate** ones. **Preferably Natural Join is performed on the foreign key.**

Notation : R ⋈ S Where R is the first relation, S is the second relation

DIVISION (÷)

Division Operation is represented by "division"(÷ or /) operator and is used in queries which involve keyword **"every"**, **"all"**, etc.

Notation: R(X,Y)/S(Y)

Here,R is the first relation from which data is to retrieved. S is second relation which will help to retrieve the data. X and Y are the attributes/columns present in relation. We can have multiple attributes in relation, but keep in mind that attributes of S must be proper subset of attributes of R. For each corresponding value of Y, above notation will return us the value of X from tuple<X,Y> which exist **everywhere**.

A

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

B1

pno
p2

B2

pno
p2
p4

B3

pno
p1
p2
p4

A/B1

sno
s1
s2
s3
s4

A/B2

sno
s1
s4

A/B3

sno
s1

Figure Examples Illustrating Division