

UNIT – II

RELATIONAL DATA MODEL IN DBMS

What is Relational Model?

Relational Model (RM) represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.

The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations. In the relational model, data are stored as tables. However, the physical storage of the data is independent of the way the data are logically organized.

RELATIONAL MODEL CONCEPTS IN DBMS

Attribute : Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME, etc.

Tables : In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.

Tuple : It is nothing but a single row of a table, which contains a single record.

Relation Schema : A relation schema represents the name of the relation with its attributes.

Degree : The total number of attributes which in the relation is called the degree of the relation.

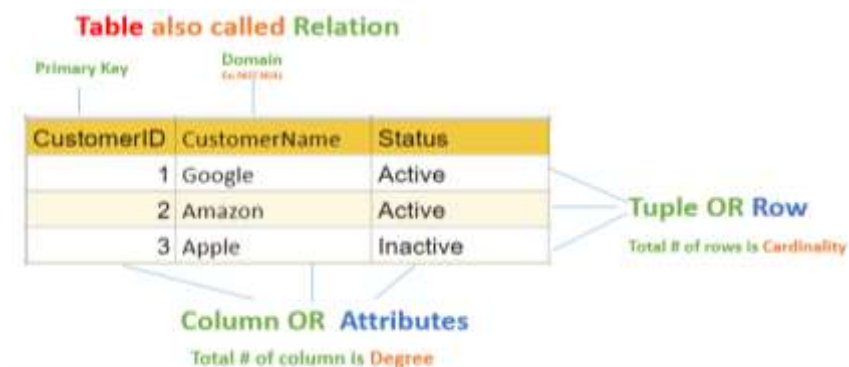
Cardinality: Total number of rows present in the Table.

Column : The column represents the set of values for a specific attribute.

Relation instance : Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.

Relation key : Every row has one, two or multiple attributes, which is called relation key.

Attribute domain : Every attribute has some pre-defined value and scope which is known as attribute domain.



CREATING AND MODIFYING RELATIONS USING SQL

The SQL language standard uses the word table to denote relation. The subset of SQL that supports the creation, deletion, and modification of tables is called the Data Definition Language (DDL).

CREATE Statement

The CREATE TABLE statement is used to create a new table in a database.

Syntax

```
CREATE TABLE table_name (column1 datatype,  
                           column2 datatype,  
                           column3 datatype , .....);
```

The column parameters specify the names of the columns of the table.

The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

Example : To create the Students relation, we can use the following statement:

```
CREATE TABLE Students ( sid      CHAR(20),  
                          name     CHAR(30) ,  
                          login    CHAR(20) ,  
                          age      INTEGER,  
                          gpa      REAL);
```

sid	name	login	age	gpa
-----	------	-------	-----	-----

Table is created.

INSERT INTO Statement

The INSERT INTO statement is used to insert new records / Tuples in a table.

Syntax : It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted

```
INSERT INTO table_name (column1, column2, column3, ...)  
VALUES (value1, value2, value3, ...);
```

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

```
INSERT INTO table_name  
VALUES (value1, value2, value3, ...);
```

Example : We can insert a single tuple into the Students table as follows:

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES (53688, 'Smith', 'smith@ee', 18, 3.2)
```

sid	name	login	age	gpa
53688	Smith	smith@ee	18	3.2

We can optionally omit the list of column names in the INTO clause and list the values in the appropriate order, but it is good style to be explicit about column names.

DELETE Statement

The DELETE statement is used to delete existing records in a table.

Syntax : DELETE FROM *table_name* WHERE *condition*;

Example : Student Table

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

We can delete tuples using the DELETE command. We can delete all Students tuples with *name* equal to Smith using the command:

```
DELETE FROM Students S
WHERE S.name = 'Smith';
```

The "Student" table will now look like this

sid	name	login	age	gpa
53831	Madayan	madayan@music	11	1.8
53832	Guldu	glldll@music	12	2.0
53666	Jones	jones@cs	18	3.4
50000	Dave	dave@cs	19	3.3

Delete All Records

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact

Syntax : DELETE FROM *table_name*;

Example : The following SQL Statement deletes all rows in the "Students" table, without deleting the table

```
DELETE FROM Students;
```

Resulting table after using this query:

sid	name	login	age	gpa
-----	------	-------	-----	-----

UPDATE Statement

The UPDATE statement is used to modify the existing records in a table.

Syntax :

```
UPDATE table_name
```

```
SET column1 = value1, column2 = value2, column3 = value3,.....
```

```
WHERE condition;
```

Example: we can increment the age and decrement the gpa of the student with *sid 53688*:

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

```
UPDATE Students S
```

```
SET S.age = S.age + 1, S.gpa = S.gpa - 1
```

```
WHERE S.sid = 53688;
```

Resulting table after using this query:

sid	name	login	age	gpa
53831	Madayan	madayan@music	11	1.8
53832	Guldu	glldll@music	12	2.0
53688	Smith	smith@ee	19	2.2
53650	Smith	smith@math	19	3.8
53666	Jones	jones@cs	18	3.4
50000	Dave	dave@cs	19	3.3

These examples illustrate some important points. The WHERE clause is applied first and determines which rows are to be modified. The SET clause then determines how these rows are to be modified. If the column being modified is also used to determine the new value, the value used in the expression on the right side of equals (=) is the *old* value, that is, before the modification.

To illustrate these points further, consider the following variation of the previous query:

UPDATE Students S

SET S.gpa = S.gpa - 0.1

WHERE S.gpa >= 3.3

sid	name	login	age	gpa
53831	Madayan	madayan@music	11	1.8
53832	Guldu	glldll@music	12	2.0
53688	Smith	smith@ee	19	2.2
53650	Smith	smith@math	19	3.7
53666	Jones	jones@cs	18	3.3
50000	Dave	dave@cs	19	3.2

INTEGRITY CONSTRAINT OVER RELATIONS

Relational Integrity constraints in DBMS are referred to conditions which must be present for a valid relation. These Relational constraints in DBMS are derived from the rules in the mini-world that the database represents.

There are many types of Integrity Constraints in DBMS. Constraints on the Relational database management system is mostly divided into Four main categories are:

1. Domain Constraints
2. Entity Integrity Constraints
3. Key Constraints
4. Referential Integrity Constraints

1. Domain Constraints

Domain refers to the range of acceptable values. It refers to the range of values that we are going to accept and store in a particular column within a database.

Domain constraints specify that within each tuple, and the value of each attribute must be unique. This is specified as data types which include standard data types integers, real numbers, characters, Booleans, variable length strings, etc.

Example: If we have to store the salary of the employees in the '*employee_table*' then we can put constraints that it should only be an INTEGER. Any entry other than integer like characters would not be acceptable and when we try to give input like this, the DBMS will produce errors.

Employee_id	Name	Salary	Age
1	Andrew	486522	25
2	Angel	978978	30
3	Anamika	697abc	35

This value is out of domain(not INTEGER)so it is not acceptable.

2. Entity Integrity Constraints : The entity constraint says that the value of the primary key should not be NULL. If the value of the primary key is NULL then we can't uniquely identify the rows if all other fields are the same. Also, with the help of primary key, we can uniquely identify each record.

Example: If we have a customer database and customer_table is present there with attributes like age and name. Then each customer should be uniquely identified. There might be two customers with the same name and same age, so there might be confusion while retrieving the data. If we retrieve the data of customer named 'Angel' then two rows are having this name and there would be confusion. So, to resolve this issues primary keys are assigned in each table and it uniquely identifies each entry of the table.

Primary Key	ID	Customer_Name	Age
	1	Andrew	18
	2	Angel	20
		Angel	20

This value cannot be NULL as we will not be able to identify customers uniquely

3. Key Constraints

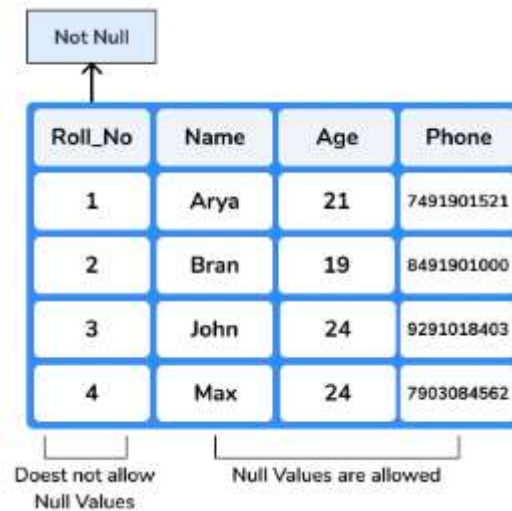
- Constraints are nothing but the rules that are to be followed while entering data into columns of the database table
- Constraints ensure that data entered by the user into columns must be within the criteria specified by the condition

We have 6 types of key constraints in DBMS

1. NOT NULL
2. UNIQUE
3. DEFAULT
4. CHECK
5. PRIMARY KEY
6. FOREIGN KEY

1. NOT NULL :

- Null represents a record where data may be missing or data for that record may be optional
- Once not null is applied to a particular column, you cannot enter null values to that column and are restricted to maintain only some proper value other than null
- A not-null constraint cannot be applied at table level



Example :

```
CREATE TABLE STUDENT
(
    ROLL_NO INT NOT NULL,
    NAME VARCHAR (20) NOT NULL,
    AGE INT NOT NULL,
    ADDRESS CHAR (25) ,
    SALARY DECIMAL (18, 2),
    PRIMARY KEY (ID)
);
```

In the above example, we have applied not null on three columns Roll_No, name and age which means whenever a record is entered using insert statement all three columns should contain a value other than null.

We have two other columns address and salary, where not null is not applied which means that you can leave the row as empty or use null value while inserting the record into the table.

2. UNIQUE :

- Sometimes we need to maintain only unique data in the column of a database table, this is possible by using a unique constraint .
- Unique constraint ensures that all values in a column are unique.

Roll_No	Name	Age	Phone
1	Arya	21	7491901521
2	Bran	19	8491901000
3	John	24	9291018403
4	Max	24	7903084562

```
CREATE TABLE Persons (  
  ID int UNIQUE,  
  LastName varchar(255) NOT NULL,  
  FirstName varchar(255),  
  Age int,  
);
```

In the above example, as we have used unique constraint on ID column we are not supposed to enter the data that is already present, simply no two ID values are same

3: DEFAULT

- Default clause in SQL is used to add default data to the columns
- When a column is specified as default with some value then all the rows will use the same value i.e each and every time while entering the data we need not enter that value
- But **default column value can be customized** i.e it can be overridden when inserting a data for that row based on the requirement.

ID_No	Name	Company	Phone
1	Arya	Preplnsta	7491901521
2	Bran	Preplnsta	8491901000
3	John	Preplnsta	9291018403
4	Max	Preplnsta	7903084562

↓
Row with Default
Value "Preplnsta"

Example for DEFAULT clause :

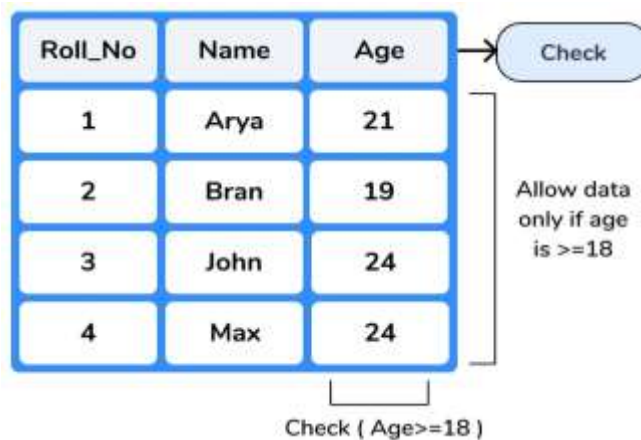
The following SQL sets a DEFAULT value for the "city" column when the "emp" table is created:

```
CREATE TABLE emp (
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int,
  City varchar(255) DEFAULT 'hyderabad'
);
```

As a result, whenever you insert a new row each time you need not enter a value for this default column that is **entering a column value for a default column is optional and if you don't enter the same value is considered that is used in the default clause**

4. Check :

- Suppose in real-time if you want to give access to an application only if the age entered by the user is greater than 18 this is done at the back-end by using a check constraint
- Check constraint ensures that the data entered by the user for that column is within the range of values or possible values specified.

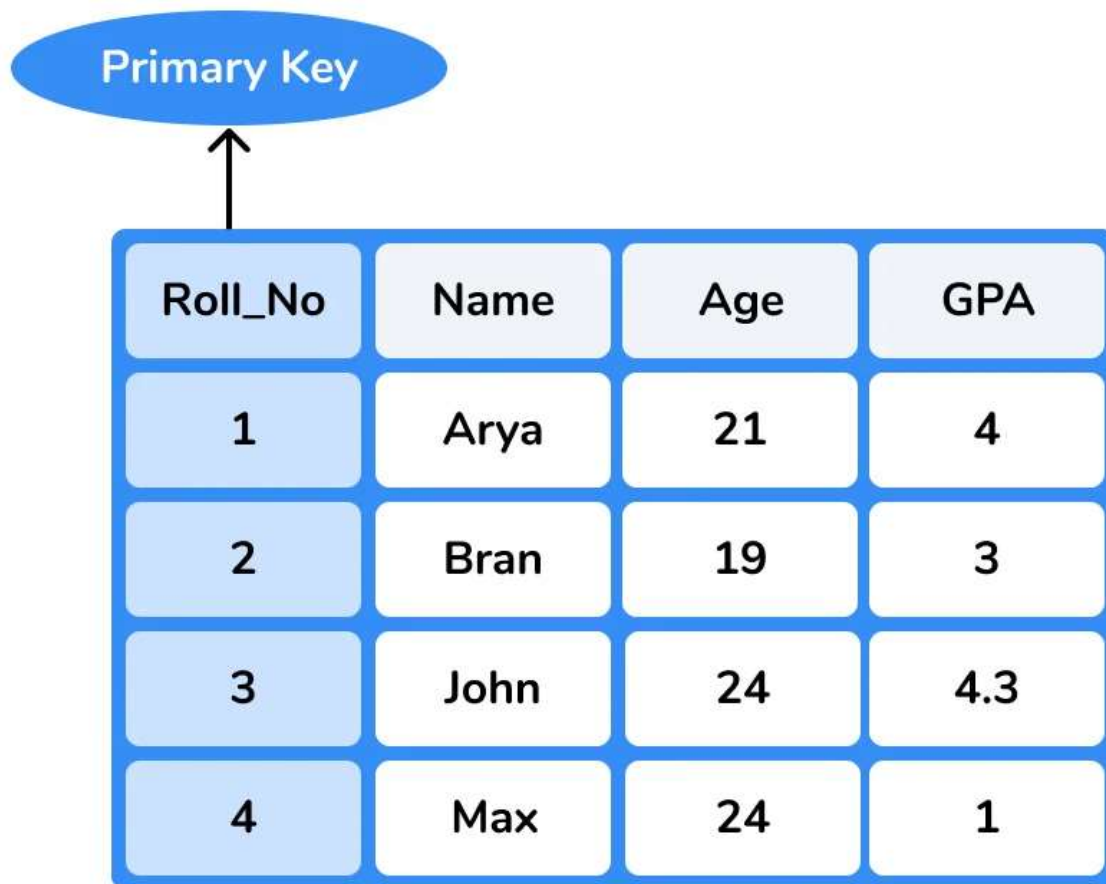


Example for check constraint

```
CREATE TABLE STUDENT (  
  ID int,  
  Name varchar(255),  
  Age int,  
  CHECK (Age>=18)  
);
```

- As we have used a **check constraint as (Age>=18)** which means **values entered by the user for this age column while inserting the data must be less than or equal to 18** otherwise an error is shown
- Simply, the only possible values that the **age column will accept is [0 -17]**

5. Primary Key : A primary key is a constraint in a table which uniquely identifies each row record in a database table by enabling one or more the column in the table as primary key.



The diagram illustrates a database table with four columns: Roll_No, Name, Age, and GPA. The Roll_No column is highlighted with a blue background, and an arrow points from a blue oval labeled "Primary Key" to this column, indicating that it is the primary key. The table contains four rows of data.

Roll_No	Name	Age	GPA
1	Arya	21	4
2	Bran	19	3
3	John	24	4.3
4	Max	24	1

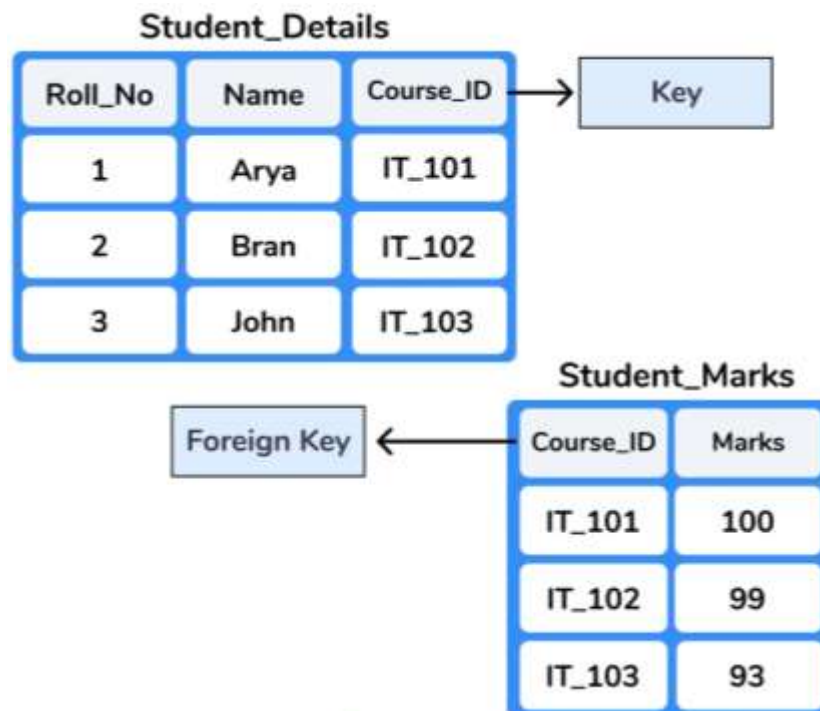
Creating a primary key : A particular column is made as a primary key column by using the primary key keyword followed with the column name.

```
CREATE TABLE EMP (  
  ID INT  
  NAME VARCHAR (20)  
  AGE INT  
  COURSE VARCHAR(10)  
  PRIMARY KEY (ID)  
);
```

- Here we have used the primary key on ID column then ID column must contain unique values i.e ***one ID cannot be used for another student.***
- If you try to ***enter duplicate value while inserting in the row you are displayed with an error***
- Hence ***primary key will restrict you to maintain unique values and not null values in that particular column***

6. Foreign Key

- The foreign key constraint is a column or list of columns which points to the primary key column of another table
- The main purpose of the foreign key is only those values are allowed in the present table that will match to the primary key column of another table.



Example to create a foreign key

Reference Table

```
CREATE TABLE CUSTOMERS1(  
  ID INT,  
  NAME VARCHAR (20) ,  
  COURSE VARCHAR(10) ,  
  PRIMARY KEY (ID)  
);
```

Child Table

```
CREATE TABLE CUSTOMERS2(  
  ID INT ,  
  MARKS INT,  
  REFERENCES CUSTOMERS1(ID)  
);
```

ENFORCING INTEGRITY CONSTRAINTS

- ICs are specified when a relation is created and enforced when a relation is modified.
- Violating the primary key constraint, the transaction will be rejected.

Example : Consider the Instance Student Instance (SI) of students

Students(sid:String, name: String, login:String, age: Ingeger, gpa:real)

Sid	Name	Login	Age	gpa
50000	Dove	dove@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	Dare	madayan@music	22	2.0

- Inserting a tuple with an existing sid.

INSERT INTO Students (sid, name, login, age, gpa)

VALUES (53688, 'Mike', 'mike@ee', 17, 3. 4);

- The insertion violates the primary key constraint because there is already a tuple with sid 53688 and it will be rejected by DBMS.

Inserting a tuple with primary key as null.

INSERT INTO Students (sid, name, login, age, gpa)

VALUES (null, 'Mike', 'mike@ee', 17, 3. 4);

>> Insertion Violates the primary key cannot contain NULL

- Updating a tuple with an existing sid. **UPDATE** Students S **SET** S. sid = 50000 **WHERE** S. sid=53688;
- This update violates the primary key constraint because there is already a tuple with SID 50000.

Deletion of Enrolled tuples do not violate referential integrity, but insertions could.

- Inserting a tuple with an un-exist sid in Students.

INSERT INTO Enrolled (sid, cid, grade)

VALUES (51111, 'Hindi 101', 'B');

>> The insertion is illegal because there is no students tuple with sid 51111.

- Insertion of Students tuples do not violate referential integrity, but deletions could.

QUERYING RELATIONAL DATA

- A Relational database query is a question about the data, and the answer consists of a new relation containing the result.
- A query language is a specialized language for writing queries.
- SQL is the most popular commercial query language for a relational DBMS.

For example :

we might want to find all students younger than 18 or all students enrolled in Reggae203.

SELECT Statement

The SELECT statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

Syntax :

```
SELECT column1, column2,.....
```

```
FROM table_name;
```

Here, column1, column2, ... are the field names of the table you want to select data from.

If you want to select all the fields available in the table, use the following syntax:

```
SELECT * FROM table_name;
```

Example :

Consider the instance of the Students relation shown in the Figure

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

We can retrieve rows corresponding to students who are younger than 18 with the following SQL query:

```
SELECT * FROM Students S
```

```
WHERE S.age < 18
```

The symbol *, means that we retain all fields of selected tuples in the result. Think of S as a variable that takes on the value of each tuple in Students, one tuple after the other. The condition *S. age* <

18 in the WHERE clause specifies that we want to select only tuples in which the *age* field has a value less than 18. This query evaluates to the relation shown in Figure.

sid	name	login	age	gpa
53831	Madayan	madayan@music	11	1.8
53832	Guldu	glldll@music	12	2.0

In addition to selecting a subset of tuples, a query can extract a subset of the fields of each selected tuple. We can compute the names and logins of students who are younger than 18 with the following query:

```
SELECT S.name, S.login
FROM Students S
WHERE S.age < 18
```

The following Figure shows the answer to this query;

name	login
Madayan	madayan@music
Guldu	glldll@music

We can also combine information in the Students and Enrolled relations.

Enrolled Table

cid	grade	studid
Carnatic101	C	53831
Reggae203	B	53832
Topology112	A	53650
History105	B	53666

If we want to obtain the names of all students who obtained an A and the id of the course in which they got an A. **we could write the following query:**

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid = E.studid AND E.grade = 'A'
```

This query can be understood as follows: "If there is a Students tuple Sand an Enrolled tuple E such that S.sid = E.studid (so that S describes the student who is enrolled in E) and E.grade = 'A', then print the student's name and the course id." When evaluated on the instances of Students and Enrolled in this query returns a single tuple, (*Smith, Topology112*).

LOGICAL DATA BASE DESIGN

Logical database design is the process of deciding how to arrange the attributes of the entities in a given business environment into database structures, such as the tables of a relational database.

The goal of logical database design is to create well structured tables that properly reflect the company's business environment. The tables will be able to store data about the company's entities in a non-redundant manner and foreign keys will be placed in the tables so that all the relationships among the entities will be supported.

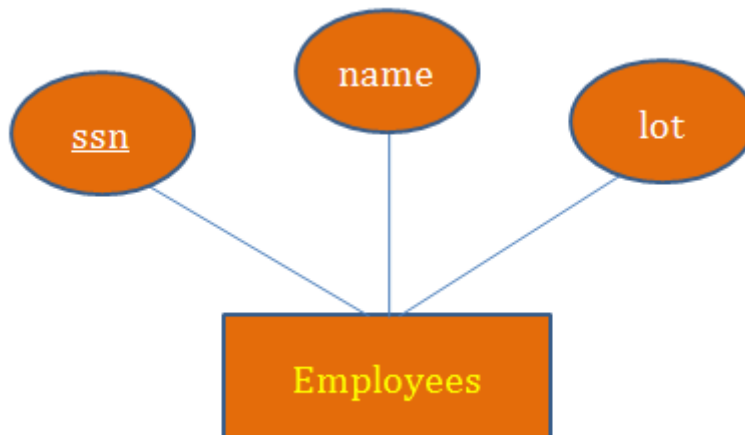
We now describe how to translate an ER diagram into a collection of tables with associated constraints, that is, a relational database schema.

Entity Sets to Tables

An entity set is mapped to a relation in a straightforward way: Each attribute of the entity set becomes an attribute of the table.

Note that we know both the domain of each attribute and the (primary) key of an entity set.

Consider the Employees entity set with attributes *ssn*, *name*, and *lot* shown in Figure



A possible instance of the Employees entity set, containing three Employees entities, is shown in Figure in a tabular format.

ssn	name	lot
123-22-3666	Attishoo	48
231-31-5368	Smiley	22
131-24-3650	Smethurst	35

The following SQL statement captures the preceding information, including the domain constraints and key information:

```
CREATE TABLE Employees ( ssn  CHAR(11),  
                          name CHAR(30) ,  
                          lot   INTEGER, PRIMARY KEY (ssn) )
```

Relationship Set into a Table

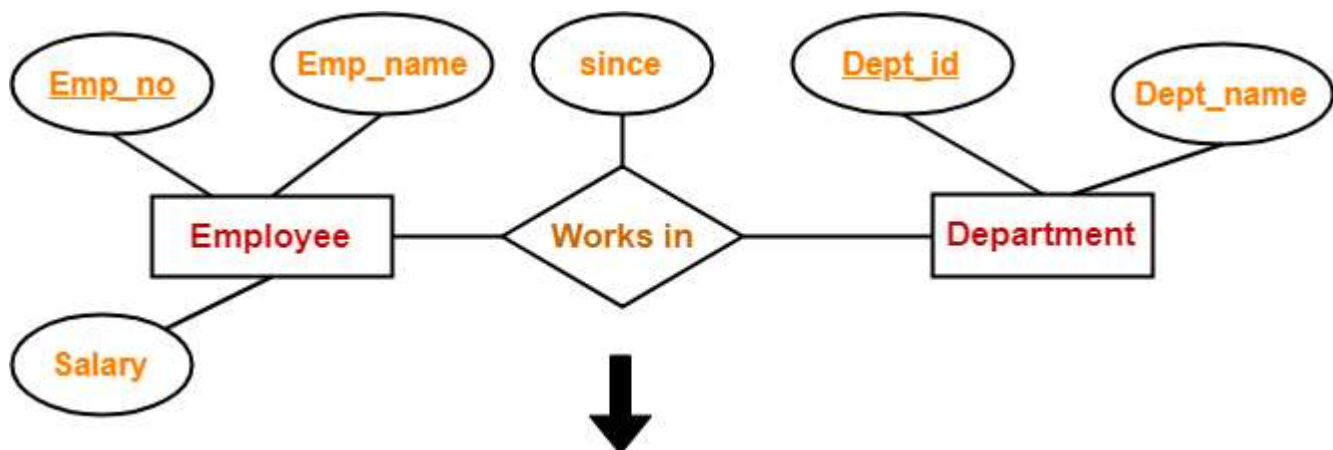
A relationship set will require one table in the relational model.

Attributes of the table are-

- Primary key attributes of the participating entity sets
- Its own descriptive attributes if any.

Set of non-descriptive attributes will be the primary key.

Example-



<u>Emp_no</u>	<u>Dept_id</u>	since

Schema : Works in (Emp_no , Dept_id , since)

NOTE-

If we consider the overall ER diagram, three tables will be required in relational model-

- One table for the entity set “Employee”
- One table for the entity set “Department”
- One table for the relationship set “Works in”

INTRODUCTION TO VIEWS

A view is a table whose rows are not explicitly stored in the database but are computed as needed from a view Definition.

[OR]

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

- Views in SQL are considered as a virtual table. A view also contains rows and columns.
- To create the view, we can select the fields from one or more tables present in the database.
- A view can either have specific rows based on certain condition or all the rows of a table.

CREATE VIEW

A view is created with the CREATE VIEW statement.

Syntax : `CREATE VIEW view_name AS`

`SELECT column1, column2, ...`

`FROM table_name`

`WHERE condition;`

Example : Student Table

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

Enrolled Table

cid	grade	studid
Carnatic101	C	53831
Reggae203	B	53832
Topology112	A	53650
History105	B	53666

```
CREATE VIEW B-Students ( name,sid, course)
```

```
AS SELECT S.sname, S.sid, E.cid
```

```
FROM Students S, Enrolled E
```

```
WHERE S.sid = 'E.studid AND E.grade='B';
```

The view B-Students has three fields name,sid, and course with the same domain as the fields sname and sid in Students and cid in Enrolled. (If the optional arguments name,sid, and course are omitted from the CREATE VIEW Statements , the coloumn name sname , sid, and cid are inheritted.).

This view can be used just like a **Base table** , or explicitly stored table, in defining new queries or views.

B-Students cntains the tuples shown in figure

An instance of the B-Student View

Name	Sid	course
Jones	53666	History105
Guldu	53832	Reggae203

Updating a View

A view can be updated with the CREATE OR REPLACE VIEW statement.

Syntax :

```
CREATE OR REPLACE VIEW view_name AS
```

```
SELECT column1, column2, ...
```

```
FROM table_name
```

```
WHERE condition;
```

Example :

```
CREATE OR REPLACE VIEW GoodStudents ( sid, gpa )
```

```
AS SELECT S.sid, S.gpa
```

```
FROM Students S
```

```
WHERE S.gpa>3.0;
```

sid	gpa
53688	3.2
53650	3.8
53666	3.4
50000	3.3

DESTROYING/ALTERING TABLES AND VIEWS

Deleting View / Destroying View

A view can be deleted using the Drop View statement.

Syntax : DROP VIEW view_name;

Example:

If we want to delete the View **B-Students**, we can do this as:

```
DROP VIEW B-Students;
```

ALTER TABLE Statement

- The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
- The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

ALTER TABLE - ADD Column

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
```

```
ADD column_name datatype;
```

Example :

```
ALTER TABLE Students
```

```
ADD DateOfBirth date;
```

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

ALTER TABLE - ALTER/MODIFY COLUMN

To change the data type of a column in a table, use the following syntax:

Syntax :

```
ALTER TABLE table_name  
    ALTER COLUMN column_name datatype;
```

Example :

```
ALTER TABLE Students  
    ALTER COLUMN DateOfBirth year;
```

ALTER TABLE - DROP COLUMN

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

Syntax :

```
ALTER TABLE table_name  
    DROP COLUMN column_name;
```

Example :

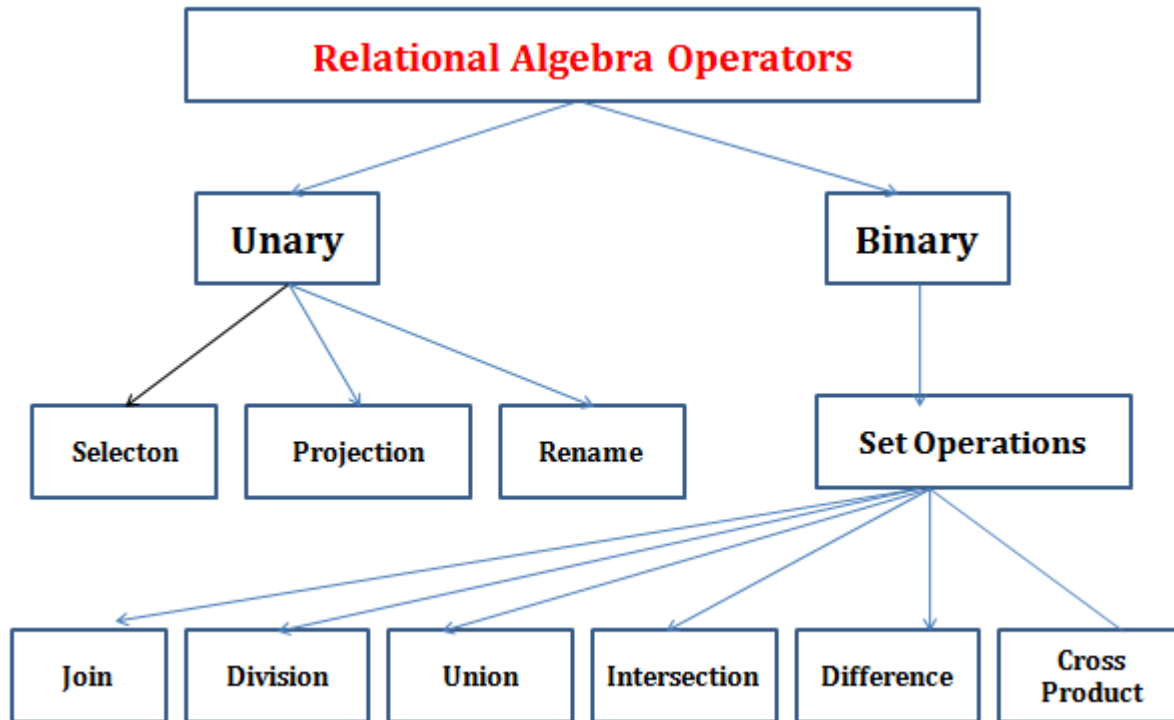
```
ALTER TABLE Students  
    DROP COLUMN DateOfBirth;
```

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

RELATIONAL ALGEBRA

Relational Algebra is a procedural query language which takes a relation as an input and generates a relation as an output.

The operators in relational algebra is classified as-



Selection Operator :

- Selection Operator (σ) is a unary operator in relational algebra that performs a selection operation.
- It selects those rows or tuples from the relation that satisfies the selection condition.
- It is denoted by sigma (σ).

Notation: $\sigma p(r)$

Where:

σ is used for selection prediction

r is used for relation

p is used as a propositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like $=, \neq, \geq, <, >, \leq$.

Relation Schema :

Sailors(sid: integer, sname:string, rating:integer,age:real)

Boats(bid: integer, bname:string, color:string)

Reserves(sid: integer, bid: integer, day:date)

In several examples the relational algebra operators, we use the instances S1 and S2 (of Sailors) and R1 (of Reserves) Shown in Figures 1,2 and 3, respectively.

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

Figure 1 Instance S1 of Sailors

sid	sname	rating	age
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

Figure 2 Instance S2 of Sailors

sid	bid	day
28	yuppy	9
31	Lubber	8

Figure 3 Instance R1 of Reserves

Example : Consider the instance of the Sailors relation shown in Figure 2

- We can retrieve rows corresponding to expert sailors by using Sigma(σ) Operator.
- The expression is

$\sigma_{\text{rating} > 8}(\text{S2})$ [Input]

Output is :

sid	sname	rating	age
28	yuppy	9	35.0
58	Rusty	10	35.0

Projection Operator :

- Projection Operator (π) is a unary operator in relational algebra that performs a projection operation.
- It displays the columns of a relation or table based on the specified attributes.

- It is denoted by Π .

Notation : $\Pi A_1, A_2, A_n (r)$

Where

A1, A2, A3 is used as an attribute name of relation **r**.

Example 1:

- We can find out all Sailor names and Ratings by using Π
- The expression is

$\Pi_{sname, rating}(S_2)$ [Input]

Output is :

sname	rating
yuppy	9
Lubber	8
guppy	5
Rusty	10

Example 2:

- Suppose that we wanted to find out only the ages of Sailors.
- The expression is $\Pi_{age}(S_2)$

Output is :

age
35.0
35.0

Example 3:

- The important point to note is that ,although three sailors are aged 35, a single tuple with age=35.0 appears in the result of the projection.
- We can compute the names and ratings of highly rated sailors by combining two of the preceding queries.
- The expression is

$\Pi_{sname, rating}(\sigma_{rating > 8}(S_2))$

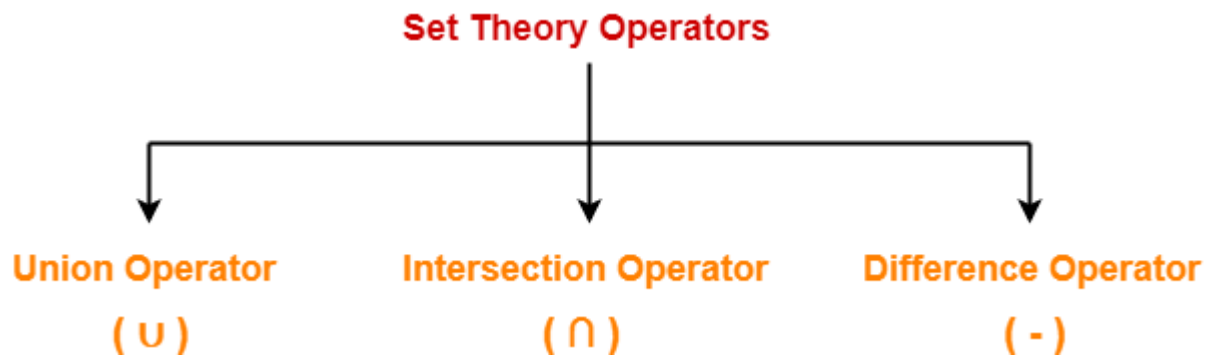
Output is :

sname	rating
-------	--------

yuppy	9
Rusty	10

Set Operations

Following operators are called as set theory operators-



Condition For Using Set Theory Operators

- To use set theory operators on two relations,
- The two relations must be union compatible.

Union compatible property means-

- Both the relations must have same number of attributes.
- The attribute domains (types of values accepted by attributes) of both the relations must be compatible.

Union Operator (U)

Let R and S be two relations. Then

- $R \cup S$ is the set of all tuples belonging to either R or S or both.
- In $R \cup S$, duplicates are automatically removed.
- Union operation is both commutative and associative.

Example :

The Union of S1 and S2 is shown in figure

sid	sname	rating	age
-----	-------	--------	-----

22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0
28	yuppy	9	35.0
44	guppy	5	35.0

Intersection Operator (\cap)

Let R and S be two relations. Then

- $R \cap S$ is the set of all tuples belonging to both R and S.
- In $R \cap S$, duplicates are automatically removed.
- Intersection operation is both commutative and associative.

Example :

The Intersection of S1 and S2 is shown in figure

sid	sname	rating	age
31	Lubber	8	55.5
58	Rusty	10	35.0

Difference Operator (-)

Let R and S be two relations. Then

- $R - S$ is the set of all tuples belonging to R and not to S.
- In $R - S$, duplicates are automatically removed.
- Difference operation is associative but not commutative.

Example:

The Difference of S1 and S2 is shown in figure

sid	sname	rating	age
22	Dustin	7	45.0

Cross-Product (X)

- $R \times S$ returns a relation instance whose schema contains All the fields of R followed by all the fields of S
- The result of $R \times S$ contains one tuple $\langle r, s \rangle$ for each pair of tuples $r \in R, s \in S$.
- The cross product operation is sometimes called Cartesian Product.

Example : The result of the Cross-Product S1 X R1 is Shown in Figure

(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

RENAMING Table :

The RENAMING TABLE statement is used to change the table name.

Syntax : RENAME tableName TO newTableName;

Example : RENAME EMPLOYEE1 TO EMPLOYEE2;

JOIN

In DBMS, a join statement is mainly used to combine two tables based on a specified common field between them. If we talk in terms of Relational algebra, it is the cartesian product of two tables followed by the selection operation. Thus, we can execute the product and selection process on two tables using a single join statement. We can use either 'on' or 'using' clause in MySQL to apply predicates to the join queries.

A Join can be broadly divided into two types:

1. Inner Join
2. Outer Join

For all the examples, we will consider the below-mentioned employee and department table.

employee Table :

empId	empName	deptId
1	Harry	2

2	Tom	3
3	Joy	5
4	Roy	8

department table

deptId	deptName
1	CSE
2	MECH
3	IT

Inner Join

Inner Join is a join that can be used to return all the values that have matching values in both the tables.

The inner join can be further divided into the following types:

1. **Equi Join**
2. **Natural Join**

1. Equi Join

Equi Join is an inner join that uses the equivalence condition for fetching the values of two tables.

The MySQL query for equi join can be as follows:

```
select employee.empId, employee.empName, department.deptName from employee Inner Join
department on employee.deptId = department.deptId;
```

empId	empName	deptName
1	Harry	MECH
2	Tom	IT

2. Natural Join

Natural Join is an inner join that returns the values of the two tables on the basis of a common attribute that has the same name and domain. It does not use any comparison operator. It also removes the duplicate attribute from the results.

The MySQL query for natural join can be as follows:

select * from employee Natural Join department;

deptId	empId	empName	deptName
2	1	Harry	MECH
3	2	Tom	IT

The above query will return the values of tables removing the duplicates. If we want to specify the attribute names, the query will be as follows:

The returned values for the above two queries are as follows:

select employee.empId, employee.empName, department.deptId, department.deptName from employee Natural Join department;

empId	empName	deptId	deptName
1	Harry	2	MECH
2	Tom	3	IT

Outer Join

Outer Join is a join that can be used to return the records in both the tables whether it has matching records in both the tables or not.

The outer join can be further divided into three types:

1. **Left-Outer Join**
2. **Right-Outer Join**
3. **Full-Outer Join**

1.Left-Outer Join

The Left-Outer Join is an outer join that returns all the values of the left table, and the values of the right table that has matching values in the left table. If there is no matching result in the right table, it will return null values in that field.

The MySQL query for left-outer join can be as follows:

select employee.empId, employee.empName, department.deptName from employee Left Outer Join department on employee.deptId = department.deptId;

The returned values for the above query is as follows:

empId	empName	deptName
1	Harry	MECH
2	Tom	IT
3	Joy	NULL
4	Roy	NULL

2. Right-Outer Join:

The Right-Outer Join is an outer join that returns all the values of the right table, and the values of the left table that has matching values in the right table.

The MySQL query for right-outer join can be as follows:

Select employee.empId, employee.empName, department.deptName from employee Right Outer Join department on employee.deptId = department.deptId;

The returned values for the above query is as follows:

empId	empName	deptName
NULL	NULL	CSE
1	Harry	MECH
2	Tom	IT

3. Full-Outer Join:

The Full-Outer join contains all the values of both the tables whether they have matching values in them or not.

The MySQL query for full-outer join can be as follows:

Select * from employee Full Join department;

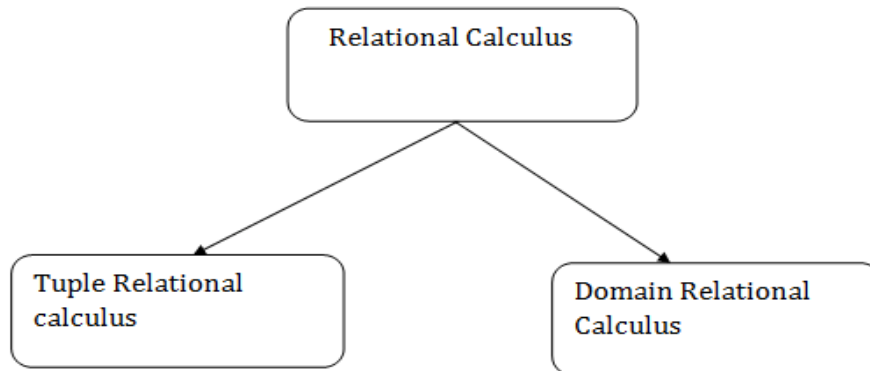
empId	empName	deptId	deptId	deptName
1	Harry	2	1	CSE
1	Harry	2	2	MECH
1	Harry	2	3	IT
2	Tom	3	1	CSE
2	Tom	3	2	MECH
2	Tom	3	3	IT
3	Joy	5	1	CSE
3	Joy	5	2	MECH
3	Joy	5	3	IT
4	Roy	8	1	CSE
4	Roy	8	2	MECH
4	Roy	8	3	IT

RELATIONAL CALCULUS

Relational Calculus

- 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.

Types of Relational calculus:



TUPLE RELATIONAL CALCULUS (TRC)

- 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:

$\{T \mid P(T)\}$ or $\{T \mid \text{Condition}(T)\}$

Where

T is the resulting tuples

P(T) is the condition used to fetch T.

Example:

$\{ T.name \mid \text{Author}(T) \text{ AND } T.article = 'database' \}$

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'.

TRC (tuple relation calculus) can be quantified. In TRC, we can use Existential (\exists) and Universal Quantifiers (\forall).

For example:

$\{ R \mid \exists T \in \text{Authors}(T.\text{article} = \text{'database'} \text{ AND } R.\text{name} = T.\text{name}) \}$

Output: This query will yield the same result as the previous one.

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 \wedge (and), \vee (or) and \neg (not).
- It uses Existential (\exists) and Universal Quantifiers (\forall) to bind the variable.

Notation:

$\{ a_1, a_2, a_3, \dots, a_n \mid P(a_1, a_2, a_3, \dots, a_n) \}$

Where

a1, a2 are attributes

P stands for formula built by inner attributes

Example:

$\{ \langle \text{article}, \text{page}, \text{subject} \rangle \mid \in \text{javatpoint} \wedge \text{subject} = \text{'database'} \}$

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