DATABASE MANAGEMENT SYSTEMS

Authors – Raghu Ramakrishna, Johannes Gehrke

Edition – 3

UNIT -2

- Introduction to Relational Model-Integrity constants over Relations, Enforcing Integrity constants-Querying Relational Data, Logical Data Base Design- Introduction To views, Destroying and Altering Tables, and Views, Relational Algebra-Selection, Projection, and Set operations- Renaming-Joins-Divisions.RelationalCalculus-TupleRelationalcalculus-DomainRelationalCalculus

Relational Model

- Relational model can represent as a table with columns and rows. Each row is known as a tuple. Each table of the column has a name or attribute.

Key Terms:

- **Tables:** In this model relations are saved in the form of tables.
- **Domain:** It contains a set of atomic values that an attribute can take.
- **Attribute:** It contains the name of a column in a particular table. Each attribute Ai must have a domain, dom (Ai)
- **Relational instance:** In the relational database system, the relational instance is represented by a finite set of tuples. Relation instances do not have duplicate tuples.

- **Relational schema:** A relational schema contains the name of the relation and name of all columns or attributes.
- **Relational key:** In the relational key, each row has one or more attributes. It can identify the row in the relation uniquely.

Example: STUDENT Relation

NAME	ROLL_NO	PHONE_NO	ADDRESS	AGE
Ram	14795	7305758992	Noida	24
Shyam	12839	9026288936	Delhi	35
Laxman	33289	8583287182	Gurugram	20
Mahesh	27857	7086819134	Ghaziabad	27
Ganesh	17282	9028 923988	Delhi	40

- In the given table, NAME, ROLL_NO, PHONE_NO, ADDRESS, and AGE are the attributes.
- The instance of schema STUDENT has 5 tuples.

Properties of Relations

- Name of the relation is distinct from all other relations
- Each relation cell contains exactly one atomic (single) value
- Each attribute contains a distinct name
- Tuple has no duplicate value
- Order of tuple can have a different sequence

Importance of NULL values

- The SQL NULL is the term used to represent a missing value
- A NULL value in a table is a value in a field that appears to be blank
- A field with a NULL value is a field with no value
- It is very important to understand that a NULL value is different than a zero value or a filled that contains space.

Syntax:

```
Create table customers (id int not null,
                        name varchar (20) not null,
                        age int not null,
                        address char (10),
                        salary decimal (18,2));
```

- Here, NOT NULL signifies that column should always accept an explicit value of the given data type.
- A field with a NULL value is the one that has been left blank during the record creation.

ID	Name	Age	Address	Salary
1	Rajesh	32	Ahmedabad	20000
2	Kalyan	25	Delhi	15000
3	Kaushik	23	Kolkata	20000
4	Kittu	25	Mumbai	35000
5	Samatha	22	Punjab	30000
6	Komali	24	MP	
7	Rani	27	Kerala	

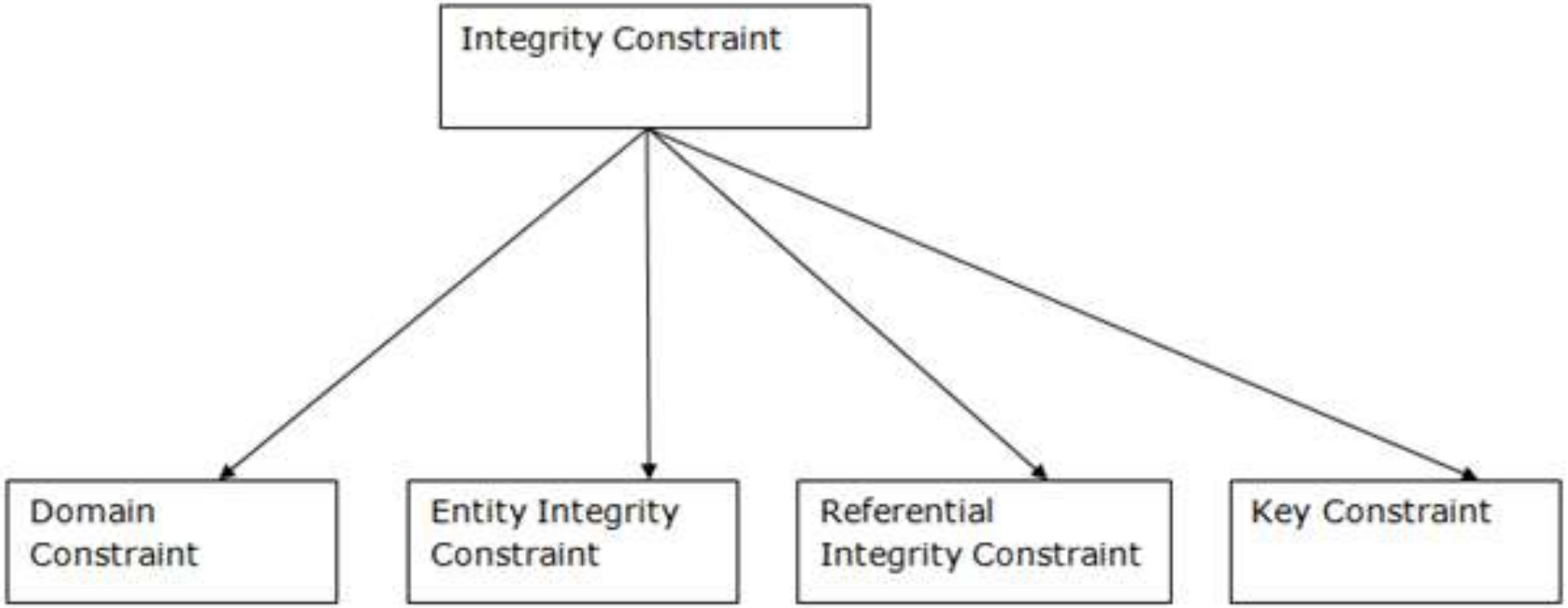
Example:

Select id , name, age, address, salary from customers where salary is not null;

ID	Name	Age	Address	Salary
1	Rajesh	32	Ahmedabad	20000
2	Kalyan	25	Delhi	15000
3	Kaushik	23	Kolkata	20000
4	Kittu	25	Mumbai	35000
5	Samatha	22	Punjab	30000

Integrity Constraints

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected. Types of integrity constraints:



Types of Integrity Constraints

1. Domain Constraints:

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

2. **Entity Integrity Constraints:**

- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field

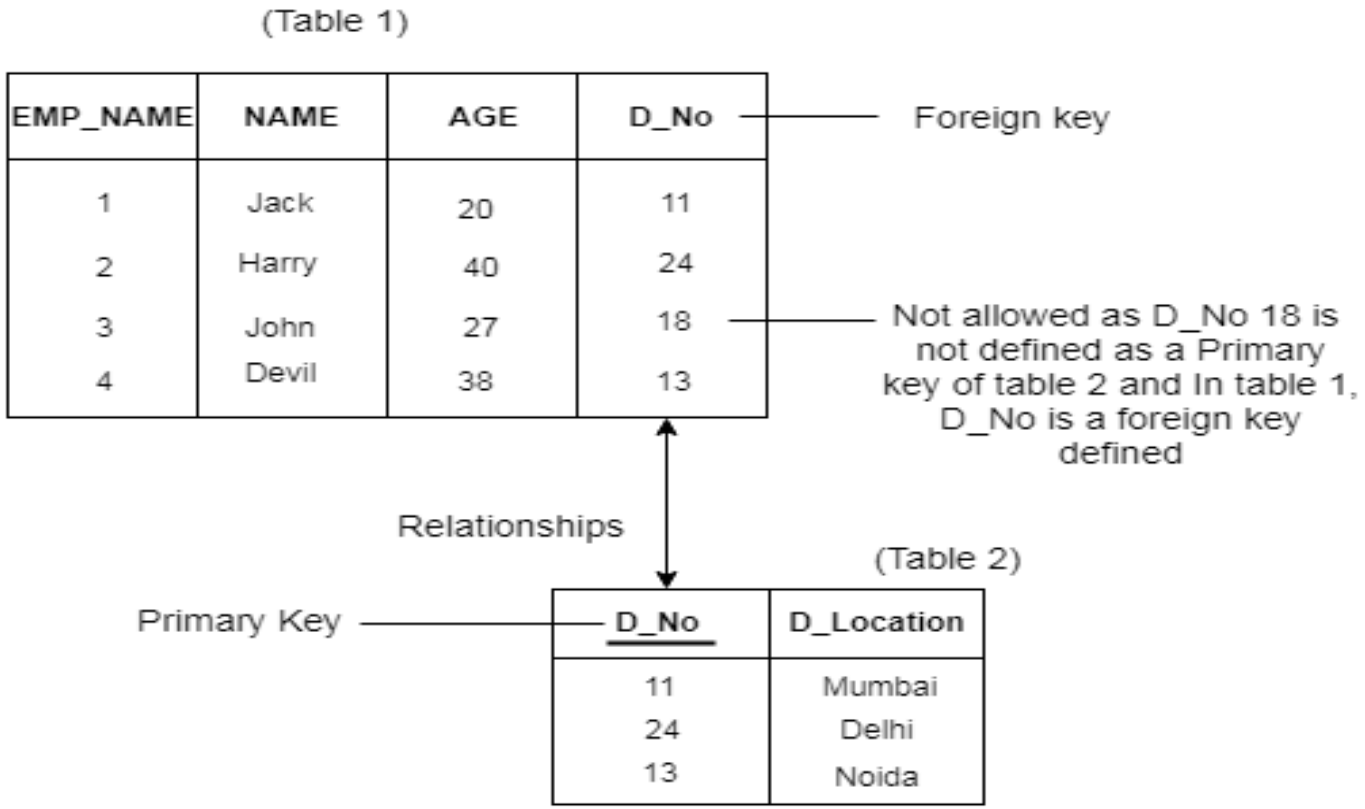
EMPLOYEE

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

3. Referential Integrity Constraints:

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.



4. Key Constraints:

- Keys are the entity set that is used to identify an entity within its entity set uniquely.
- An entity set can have multiple keys, but out of which one key will be the primary key.
- A primary key can contain a unique value in the relational table

Example:

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

Not allowed. Because all row must be unique

C
1.

```
SQL> create table student(sid int, sname varchar2(20),branch varchar2(10), perc float,
  2  primary key(sid));

Table created.

SQL> desc student;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SID                                       NOT NULL NUMBER(38)
 SNAME                                              VARCHAR2(20)
 BRANCH                                             VARCHAR2(10)
 PERC                                               FLOAT(126)

SQL> insert into student values(101,'abc','cse',80.5);

1 row created.

SQL> select * from student;

       SID SNAME                BRANCH           PERC
---------- -------------------- ---------- ----------
       101 abc                  cse              80.5

SQL> insert into student values(101,'abc','cse',80.5);
insert into student values(101,'abc','cse',80.5)
*
ERROR at line 1:
ORA-00001: unique constraint (SYSTEM.SYS_C007037) violated
```

Constraints on table

2. NOT NULL

```
SQL> clear
SQL> create table student(sid int,sname varchar2(10),branch varchar2(10),per float not null ,primary key(sid));

Table created.

SQL> desc student;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SID                                       NOT NULL NUMBER(38)
 SNAME                                              VARCHAR2(10)
 BRANCH                                             VARCHAR2(10)
 PER                                       NOT NULL FLOAT(126)
```

```
SQL> insert into student(sid,sname,branch,per) values(101,'abc','cse',90.4);

1 row created.
```

```
SQL> insert into student(sid,sname,branch) values(101,'abc','cse');
insert into student(sid,sname,branch) values(101,'abc','cse')
*
ERROR at line 1:
ORA-01400: cannot insert NULL into ("SYSTEM"."STUDENT"."PER")


SQL> insert into student(sid,sname,branch,per) values(102,'abcd','cse',98.3);

1 row created.

SQL> insert into student(sid,branch,per) values(102,'cse',98.3);
insert into student(sid,branch,per) values(102,'cse',98.3)
*
ERROR at line 1:
ORA-00001: unique constraint (SYSTEM.SYS_C007039) violated


SQL> insert into student(sid,branch,per) values(103,'cse',98.3);

1 row created.
```

```
SQL> select * from student;

       SID SNAME      BRANCH             PER
---------- ---------- ---------- ----------
       101 abc        cse              90.4
       102 abcd       cse              98.3
       103            cse              98.3
```

Constraints on table

3. CHECK

```
SQL> create table student(
  2  sid int, sname varchar2(10),
  3  branch varchar2(20), age int,
  4  primary key(sid), check (age>=18));

Table created.

SQL> insert into student values (101,'abc','cse',19);

1 row created.

SQL> insert into student values (102,'abcd','eee',17);
insert into student values (102,'abcd','eee',17)
*
ERROR at line 1:
ORA-02290: check constraint (SYSTEM.SYS_C007042) violated
```

Constraints on table

4. DEFAULT

```
SQL> CREATE TABLE STUDENT(
  2  SID INT, SNAME VARCHAR2(10),
  3  BRANCH VARCHAR2(10), COLLEGE VARCHAR2(10) DEFAULT 'CSE',
  4  PRIMARY KEY(SID));

Table created.

SQL> DESC STUDENT;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------
 SID                                       NOT NULL NUMBER(38)
 SNAME                                              VARCHAR2(10)
 BRANCH                                             VARCHAR2(10)
 COLLEGE                                            VARCHAR2(10)

SQL> INSERT INTO STUDENT VALUES(101,'ABC','CSE','GEC');

1 row created.

SQL> INSERT INTO STUDENT(SID,SNAME,BRANCH) VALUES(102,'ABCD','EEE');

1 row created.

SQL> SELECT * FROM STUDENT;

       SID SNAME      BRANCH     COLLEGE
---------- ---------- ---------- ----------
       101 ABC        CSE        GEC
       102 ABCD       EEE        CSE
```

Constraints on table

5. UNIQUE

```
SQL> create table student(sid int, sname varchar2(10),
  2  contact int unique, primary key(sid));

Table created.

SQL> desc student;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 SID                                       NOT NULL NUMBER(38)
 SNAME                                              VARCHAR2(10)
 CONTACT                                            NUMBER(38)
```

```
SQL> insert into student values(101,'abc',123433);

1 row created.

SQL> insert into student values(102,'abc',123433);
insert into student values(102,'abc',123433)
*
ERROR at line 1:
ORA-00001: unique constraint (SYSTEM.SYS_C007046) violated
```

```
SQL> insert into student(sid,sname) values(102,'abc');

1 row created.

SQL> select * from student;

       SID SNAME          CONTACT
---------- ---------- ----------
       101 abc            123433
       102 abc

SQL>
```

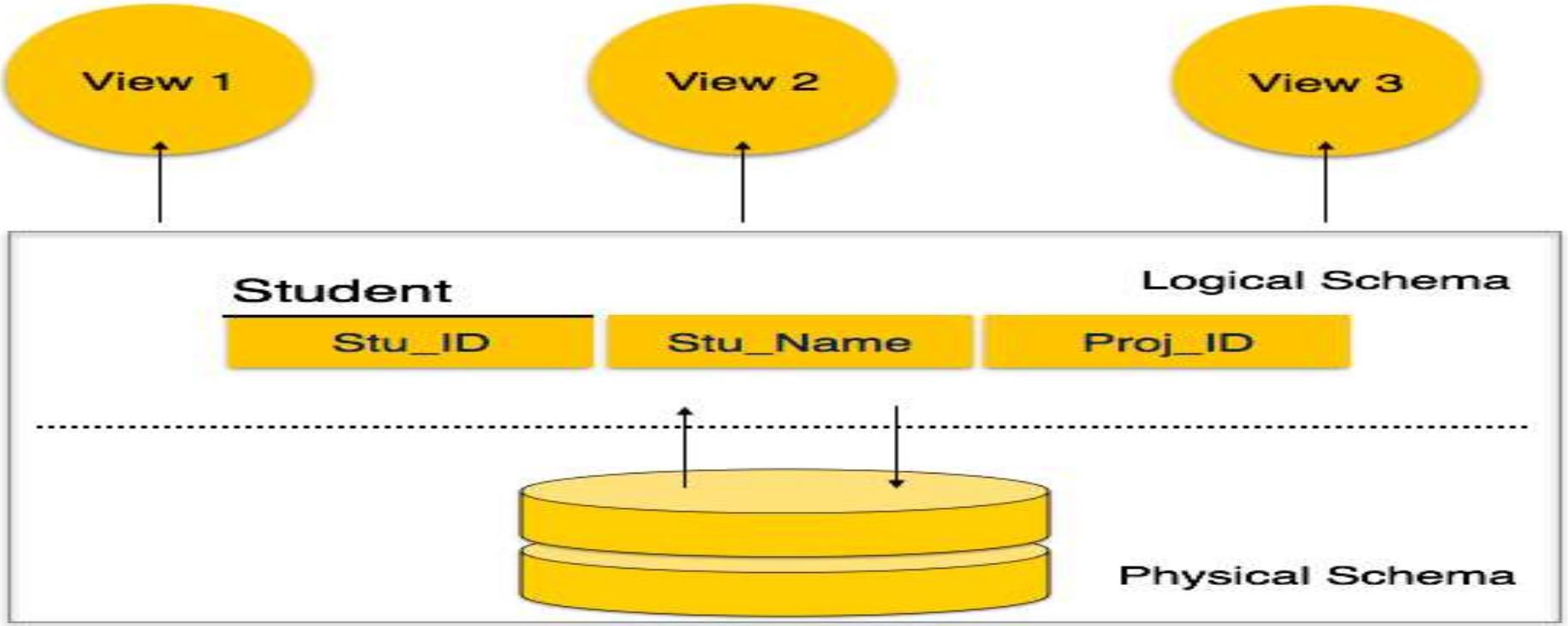
Constraints on table
6. FOREIGN KEY

```
SQL> CREATE TABLE COURSE(CID INT,CNAME VARCHAR2(10),
  2  PRIMARY KEY(CID));

Table created.

SQL> CREATE TABLE STUDENT(SID INT, SNAME VARCHAR2(10),
  2  BRANCH VARCHAR2(10), CID INT,
  3  PRIMARY KEY(SID),
  4  FOREIGN KEY(CID) REFERENCES COURSE(CID));

Table created.
```

```
SQL> DESC COURSE;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 CID                                       NOT NULL NUMBER(38)
 CNAME                                              VARCHAR2(10)

SQL> DESC STUDENT;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 SID                                       NOT NULL NUMBER(38)
 SNAME                                              VARCHAR2(10)
 BRANCH                                             VARCHAR2(10)
 CID                                                NUMBER(38)
```

Database Schema (Database Design)

- It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data. A database schema defines its entities and the relationship among them.



Querying Relational data

1. SQL Data Types:

- Data types are used to represent the nature of the data that can be stored in the database table.
- For example, in a particular column of a table, if we want to store a string type of data then we will have to declare a string data type of this column.
- Data types mainly classified into three categories for every database.

1. String Data types
2. Numeric Data types
3. Date and time Data types

MySQL String Data Types

CHAR(Size)	It is used to specify a fixed length string that can contain numbers, letters, and special characters. Its size can be 0 to 255 characters. Default is 1.
VARCHAR(Size)	It is used to specify a variable length string that can contain numbers, letters, and special characters. Its size can be from 0 to 65535 characters.
BINARY(Size)	It is equal to CHAR() but stores binary byte strings. Its size parameter specifies the column length in the bytes. Default is 1.
VARBINARY(Size)	It is equal to VARCHAR() but stores binary byte strings. Its size parameter specifies the maximum column length in bytes.
TEXT(Size)	It holds a string that can contain a maximum length of 255 characters.
TINYTEXT	It holds a string with a maximum length of 255 characters.
MEDIUMTEXT	It holds a string with a maximum length of 16,777,215.
LONGTEXT	It holds a string with a maximum length of 4,294,967,295 characters.
ENUM(val1,val2...)	It is used when a string object having only one value, chosen from a list of possible values. It contains 65535 values in an ENUM list. If you insert a value that is not in the list, a blank value will be inserted.
SET(val1,val2,....)	It is used to specify a string that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values at one time in a SET list.
BLOB(size)	It is used for BLOBs (Binary Large Objects). It can hold up to 65,535 bytes.

MySQL Numeric Data Types

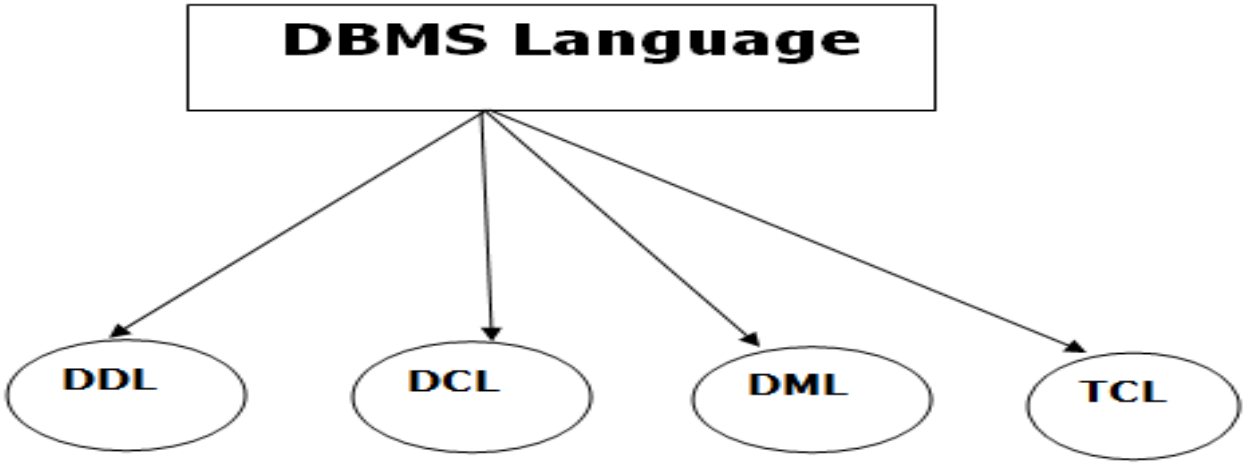
BIT(Size)	It is used for a bit-value type. The number of bits per value is specified in size. Its size can be 1 to 64. The default value is 1.
INT(size)	It is used for the integer value. Its signed range varies from -2147483648 to 2147483647 and unsigned range varies from 0 to 4294967295. The size parameter specifies the max display width that is 255.
INTEGER(size)	It is equal to INT(size).
FLOAT(size, d)	It is used to specify a floating point number. Its size parameter specifies the total number of digits. The number of digits after the decimal point is specified by d parameter.
FLOAT(p)	It is used to specify a floating point number. MySQL used p parameter to determine whether to use FLOAT or DOUBLE. If p is between 0 to24, the data type becomes FLOAT (). If p is from 25 to 53, the data type becomes DOUBLE().
DOUBLE(size, d)	It is a normal size floating point number. Its size parameter specifies the total number of digits. The number of digits after the decimal is specified by d parameter.
DECIMAL(size, d)	It is used to specify a fixed point number. Its size parameter specifies the total number of digits. The number of digits after the decimal parameter is specified by d parameter. The maximum value for the size is 65, and the default value is 10. The maximum value for d is 30, and the default value is 0.
DEC(size, d)	It is equal to DECIMAL(size, d).
BOOL	It is used to specify Boolean values true and false. Zero is considered as false, and nonzero values are considered as true.

MySQL Date and Time Data Types

DATE	It is used to specify date format YYYY-MM-DD. Its supported range is from '1000-01-01' to '9999-12-31'.
DATETIME (fsp)	It is used to specify date and time combination. Its format is YYYY-MM-DD hh:mm:ss. Its supported range is from '1000-01-01 00:00:00' to 9999-12-31 23:59:59'.
TIMESTAMP(fsp)	It is used to specify the timestamp. Its value is stored as the number of seconds since the Unix epoch('1970-01-01 00:00:00' UTC). Its format is YYYY-MM-DD hh:mm:ss. Its supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC.
TIME(fsp)	It is used to specify the time format. Its format is hh:mm:ss. Its supported range is from '-838:59:59' to '838:59:59'
YEAR	It is used to specify a year in four-digit format. Values allowed in four digit format from 1901 to 2155, and 0000.

2. SQL Languages

- Database languages can be used to read, store and update the data in the database.



- **Data Definition Language (DDL)**
- **Data Manipulation Language (DML)**
- **Data Control Language (DCL)**
- **Transaction Control Language (TCL)**

3. SQL Queries

- **Create**: create table table_name;
- **Insert**: Insert into table_name(col1,col2….)into (value1, value2..)
- **Drop**: drop table table_name
- **Alter**: alter table table_name add column_name datatype;
- **Rename**: rename old_name to new_name;
- **Truncate**: truncate table table_name;
- **Update**: update table_name set col=val1,col2=val2..where condition
- **Delete**: delete from table_name where condition;
- **Select**: select column from table_name where condition;

4. SQL operators

i) **SQL Arithmetic Operators:** two variables "a" and "b". Here "a" is valued 50 and "b" valued 100.

Example:

Operators	Descriptions	Examples
+	It is used to add containing values of both operands	a+b will give 150
-	It subtracts left hand operand from Right hand operand	a-b will give -50
*	It multiply both operand's values	a*b will give 5000
/	It divides Right hand operand by left hand operand	b/a will give 2
%	It divides Right hand operand by left hand operand and returns reminder	b%a will give 0

ii) **SQL Comparison Operators:** two variables "a" and "b" that are valued 50 and 100.

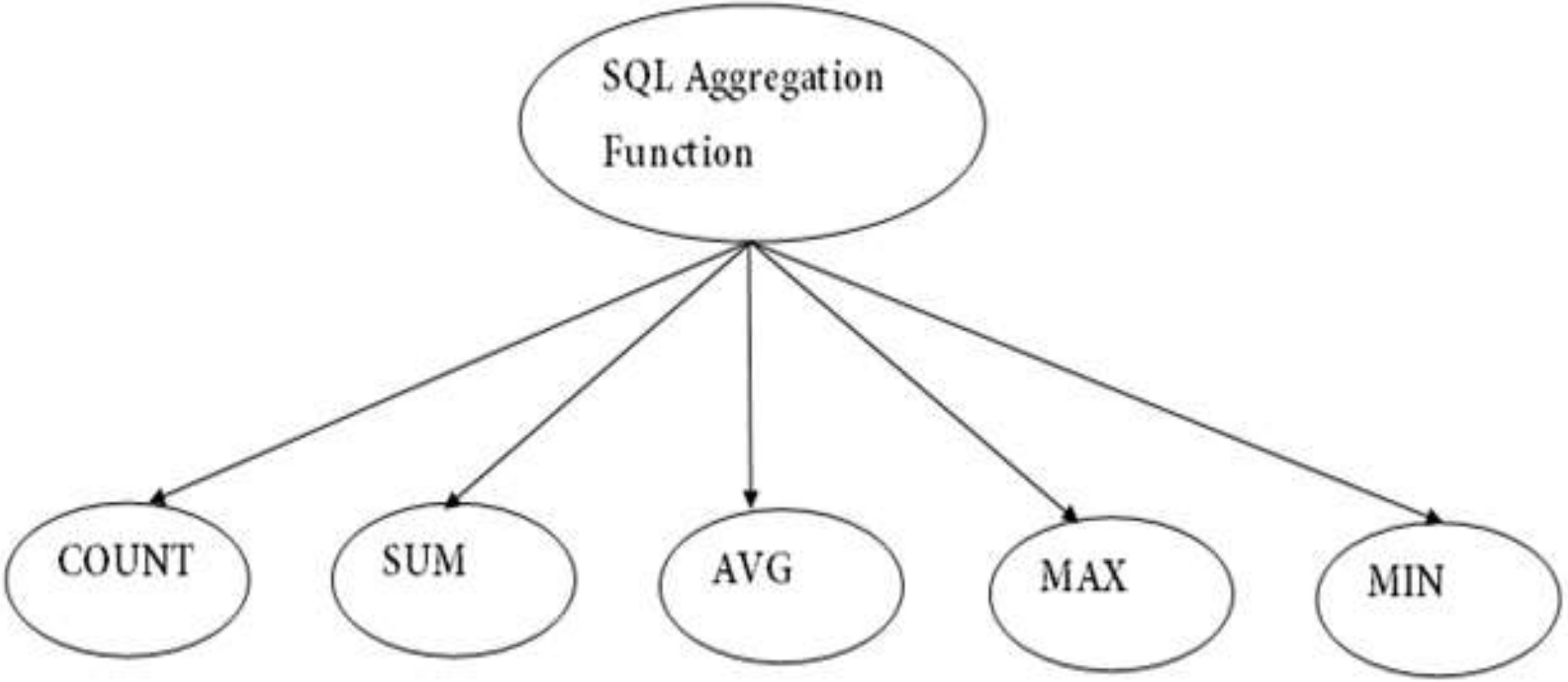
Operator	Description	Example
=	Examine both operands value that are equal or not, if yes condition become true.	(a=b) is not true
!=	This is used to check the value of both operands equal or not, if not condition become true.	(a!=b)is true
<>	Examines the operand's value equal or not, if values are not equal condition is true	(a<>b) is true
>	Examine the left operand value is greater than right Operand, if yes condition becomes true	(a>b) is not true
<	Examines the left operand value is less than right Operand, if yes condition becomes true	(a<="" td="">
>=	Examines that the value of left operand is greater than or equal to the value of right operand or not, if yes condition become true	(a>=b) is not true
<=	Examines that the value of left operand is less than or equal to the value of right operand or not, if yes condition becomes true	(a<=b) is true
!<	Examines that the left operand value is not less than the right operand value	(a!<="" td="">
!>	Examines that the value of left operand is not greater than the value of right operand	(a!>b) is true

iii) SQL Logical Operators:

Operator	Description
ALL	to compare a value to all values in another value set.
AND	operator allows the existence of multiple conditions in an SQL statement.
ANY	To compare a value to any applicable value in the list as per the condition.
BETWEEN	to search for values, that are within a set of values
IN	to compare a value to that specified list value
NOT	reverse the meaning of any logical operator
OR	to combine multiple conditions in SQL statements
EXISTS	to search for the presence of a row in a specified table
LIKE	to compare a value to similar values using wildcard operator

5. SQL aggregation functions

- SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.



- **COUNT():** COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types

Syntax: COUNT(*) or COUNT ([All /DISTINCT] expression)

Example:

PRODUCT_MAST:

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Com1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

EXAMPLE:

- SELECT COUNT (*) FROM PRODUCT_MAST;   //10
- SELECT COUNT(*) FROM PRODUCT_MAST WHERE RATE>=20;   //7
- SELECT COUNT(DISTINCT COMPANY) FROM PRODUCT_MAST;   //3
- SELECT COMPANY, COUNT(*) FROM PRODUCT_MAST GROUP BY COMPANY;

  // COM1 5, COM2 3, COM3 2
- SELECT COMPANY, COUNT(*) FROM PRODUCT_MAST GROU BY COMPANY HAVING COUNT(*) > 2;

  // COM1 5, COM2 3

```
SQL> SELECT COUNT(*) FROM PRODUCT_MAST;

  COUNT(*)
----------
        10

SQL> SELECT COUNT(DISTINCT QTY) FROM PRODUCT_MAST;

COUNT(DISTINCTQTY)
------------------
                 4

SQL> SELECT COMPANY, COUNT(*) FROM PRODUCT_MAST GROUP BY COMPANY;

COMPANY      COUNT(*)
---------- ----------
COM3                2
COM2                3
COM1                5

SQL> SELECT COMPANY, COUNT(*) FROM PRODUCT_MAST GROUP BY COMPANY;

COMPANY      COUNT(*)
---------- ----------
COM3                2
COM2                3
COM1                5

SQL> SELECT COMPANY, COUNT(*) FROM PRODUCT_MAST GROUP BY COMPANY HAVING COUNT(*)>2;

COMPANY      COUNT(*)
---------- ----------
COM2                3
COM1                5
```

- **Sum()** : used to calculate the sum of all selected columns.

Syntax: SUM([ALL/DISTINCT] EXPRESSION)

Example: SELECT SUM(COST) FROM PRODUCT_MAST; //670

Example: SELECT SUM(COST) FROM PRODUCT_MAST WHERE QTY>3;

//320

- **AVG():** Used to calculate the average value of a numeric type.

Syntax: AVG([ALL/DISTINCT] EXPRESSION)

Example: SELECT AVG(COST) FROM PRODUCT_MAST; //67.00

```
SQL> SELECT * FROM PRODUCT_MAST;

PRODUCT    COMPANY              QTY         RATE         COST
---------- ---------- ---------- ---------- ----------
ITEM1      COM1                  2         10           20
ITEM2      COM2                  3         25           75
ITEM3      COM1                  2         30           60
ITEM4      COM3                  5         10           50
ITEM5      COM2                  2         20           40
ITEM6      COM1                  3         25           75
ITEM7      COM1                  5         30          150
ITEM8      COM1                  3         10           30
ITEM9      COM2                  2         25           50
ITEM10     COM3                  4         30          120

10 rows selected.

SQL> SELECT SUM(COST) FROM PRODUCT_MAST;

 SUM(COST)
----------
       670

SQL> SELECT AVG(COST) FROM PRODUCT_MAST;

 AVG(COST)
----------
        67
```

• **Min():** used to find the minimum value of certain column

Syntax: MIN([ALL/DISTINCT] expression)

Example: SELECT MAX(RATE) FROM PRODUCT_MAST;

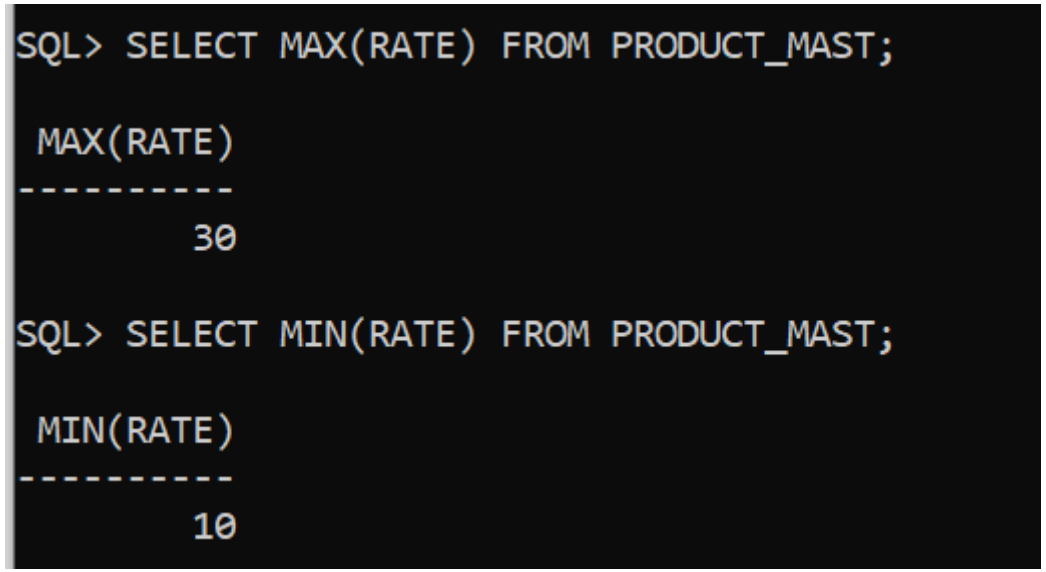
//10

• **Max():** used to find the maximum value of certain column.

Syntax: MAX([ALL/DISTINCT] expression)

Example: SELECT MAX(RATE) FROM PRODUCT_MAST;

//30



6. Scalar Functions

- These functions are based on user input, these too return single values.
  - UPPER()/UCASE():
  - LOWER()
  - SUBSTR()
  - LENGTH()
  - ROUND()
  - NOW()
  - FORMAT()

```
SQL> SELECT * FROM STUDENTDATA;

        ID STUD_NAME                 MARKS        AGE
---------- -------------------- ---------- ----------
         1 HARISH                       90         19
         2 MEGHANA                      50         20
         3 PRAKRUTHI                    80         19
         4 DHANA                        95         20
         5 DEVI                         85         18
```

```
SQL> SELECT UPPER(STUD_NAME) FROM STUDENTDATA;

UPPER(STUD_NAME)
--------------------
HARISH
MEGHANA
PRAKRUTHI
DHANA
DEVI
```

- UPPER(), LOWER(),SUBSTR(),LENGTH(),ROUND()

```
SQL> SELECT LOWER(STUD_NAME) FROM STUDENTDATA;

LOWER(STUD_NAME)
--------------------
harish
meghana
prakruthi
dhana
devi
```

```
SQL> SELECT SUBSTR(STUD_NAME,2,5) FROM STUDENTDATA;

SUBSTR(STUD_NAME,2,5
--------------------
ARISH
EGHAN
RAKRU
HANA
EVI
```

```
SQL> SELECT LENGTH(STUD_NAME) FROM STUDENTDATA;

LENGTH(STUD_NAME)
-----------------
                6
                7
                9
                5
                4
```

```
SQL> SELECT ROUND(MARKS,0) FROM STUDENTDATA;

ROUND(MARKS,0)
--------------
            90
            50
            80
            95
            85
```

NOW() AND FORMAT()

- NOW(): Returns the current system date and time.

Syntax: SELECT NOW() FROM TABLE_NAME;

Example: SELECT NAME, NOW() AS DATETIME FROM STUDENTDATA;

NAME	DateTime
HARSH	1/13/2017 1:30:11 PM

FORMAT(): Used to format how a field is to be displayed.

Syntax: SELECT FORMAT(COLUMN_NAME ,FORMAT) FROM TABLE_NAME;

Example: SELECT NAME,FORMAT(NOW(),'YYYY-MM-DD') AS DATE FROM STUDENTS;

NAME	Date
HARSH	2017-01-13

SQL DATE AND TIME FUNCTIONS:

1. Sysdate(): returns the current date and time of the system.

```
SQL> SELECT SYSDATE FROM DUAL;

SYSDATE
---------
20-FEB-24
```

2. Months_between(x,y): This function takes two values namely x and y which are in the form of months. It returns the number of months between x and y.

Example: SELECT MONTHS_BETWEEN (SYSDATE, EMP_JOIN_DATE) FROM EMP;

Consider the Employee joining date as 1-January-2018 and the system date as 1-August-2018. Therefore the above returns 7

3. ADD_MONTHS (d, n):

This function gives the same day as d, n number of months away. The value of n can be positive or negative.

Example: SELECT SYSDATE, ADD_MONTHS (SYSDATE,2)FROM DUAL;

This function will return the sysdate and the date 2 months after the sysdate i.e. '1-August-2018' and '1-October-2018'.

4. LAST_DAY(d):

This function returns the last day of the month for the specific month d provided in the function.

Example: SELECT SYSDATE, LAST_DAY (SYSDATE)FROM DUAL;

This returns the system date and the last day of the particular month for the system date i.e.

'1-August-2018' and '31-August-2018'.

7. SQL Numeric Functions

- ABS(X): Returns absolute value of X
- MOD(X,Y): remainder is returned
- SIGN(X): returns +1 when the number is positive and -1 for negative
- FLOOR(X):returns the largest int value that is either less than X or equal to it
- CEIL(X): returns the smallest int value that is greater than or equal to it.
- POWER(X,Y): returns the value of X raised to the power of Y

```
SQL> SELECT ABS(-110) FROM DUAL;

 ABS(-110)
----------
       110

SQL> SELECT MOD(7,2) FROM DUAL;

  MOD(7,2)
----------
         1

SQL> SELECT SIGN(-11) FROM DUAL;

 SIGN(-11)
----------
        -1

SQL> SELECT SIGN(11) FROM DUAL;

  SIGN(11)
----------
         1

SQL> SELECT FLOOR(10.34) FROM DUAL;

FLOOR(10.34)
------------
          10
```

```
SQL> SELECT CEIL(10.34) FROM DUAL;

CEIL(10.34)
-----------
         11

SQL> SELECT POWER(3,2) FROM DUAL;

POWER(3,2)
----------
         9

SQL> SELECT TRUNC(324.325) FROM DUAL;

TRUNC(324.325)
--------------
           324

SQL> SELECT ROUND(34.35,0) FROM DUAL;

ROUND(34.35,0)
--------------
            34

SQL> SELECT ROUND(34.35,1) FROM DUAL;

ROUND(34.35,1)
--------------
          34.4
```

8. String conversion functions

- TO_CHAR()
- TO_NUMBER()
- TO_DATE()

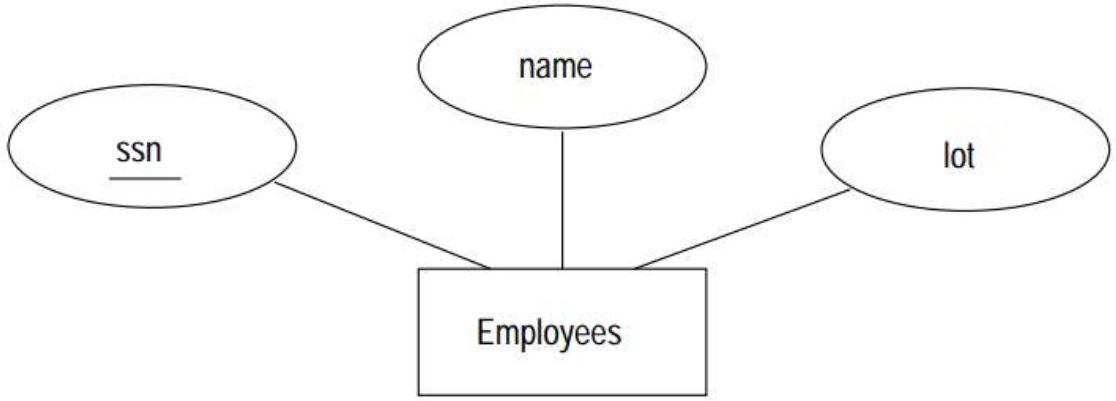
```
SQL> SELECT TO_DATE('102118','MMDDYY') FROM DUAL;

TO_DATE('
---------
21-OCT-18

SQL> SELECT SYSDATE FROM DUAL;

SYSDATE
---------
20-FEB-24

SQL> SELECT TO_CHAR(SYSDATE,'DD-MM-YYYY') FROM DUAL;

TO_CHAR(SY
----------
20-02-2024

SQL> SELECT TO_NUMBER('12','99D99') FROM DUAL;

TO_NUMBER('12','99D99')
-----------------------
                     12
```

Logical database Design

1. Entity Sets to Tables
2. Translating Relationship Sets with Key Constraints
3. Translating Weak Entity Sets

Entity Sets to Tables

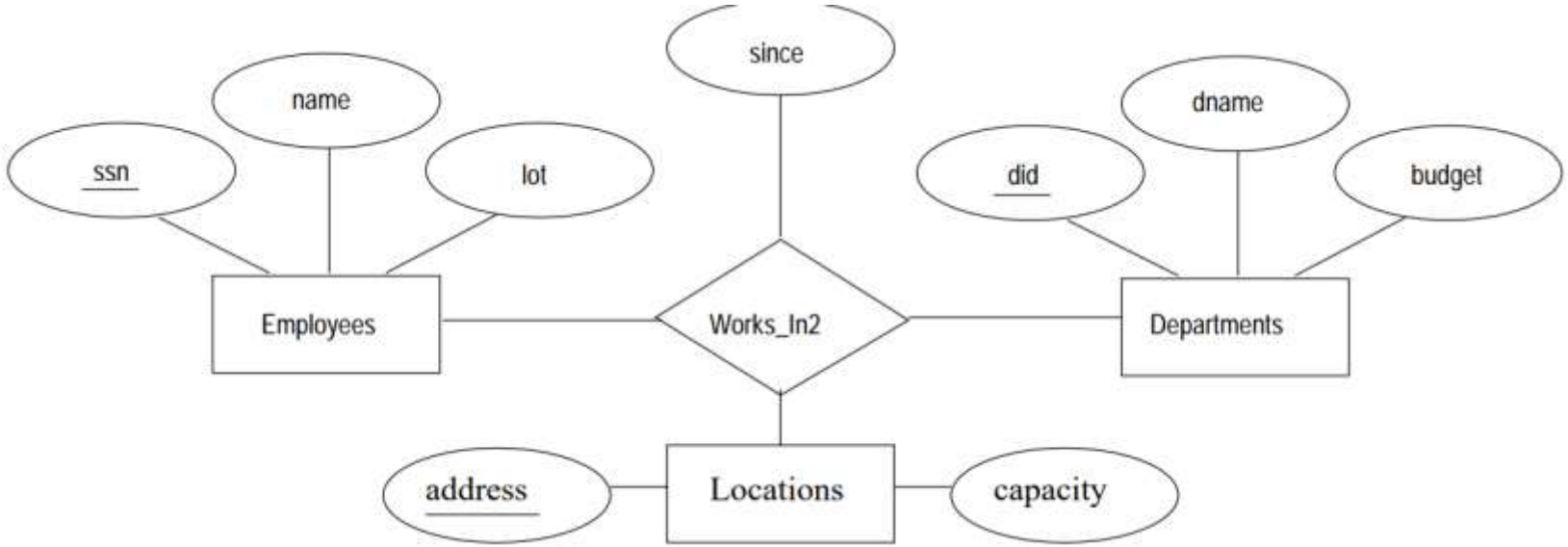
- An entity set is mapped to a relation in a straightforward way: Each attribute of the entity set becomes an attribute of the table. Note that we know both the domain of each attribute and the (primary) key of an entity set.



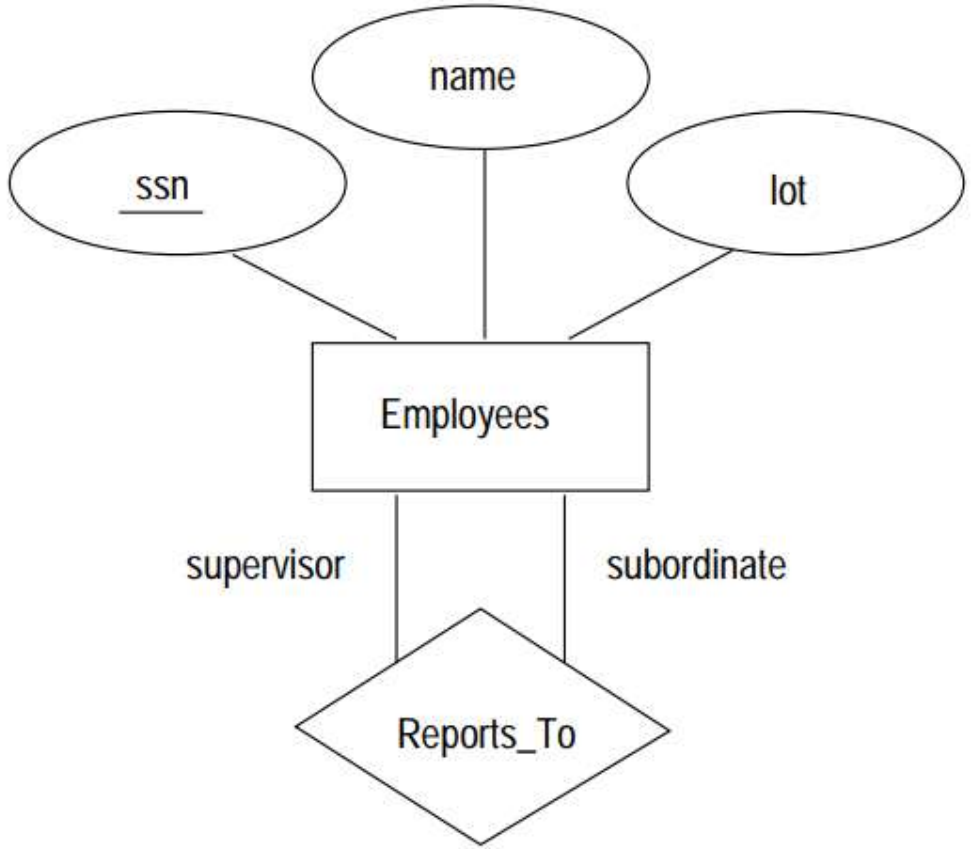

```
CREATE TABLE Employees ( ssn      CHAR(11),
                         name     CHAR(30),
                         lot      INTEGER,
                         PRIMARY KEY (ssn) )
```

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

Relationship Sets with key constraints to Tables

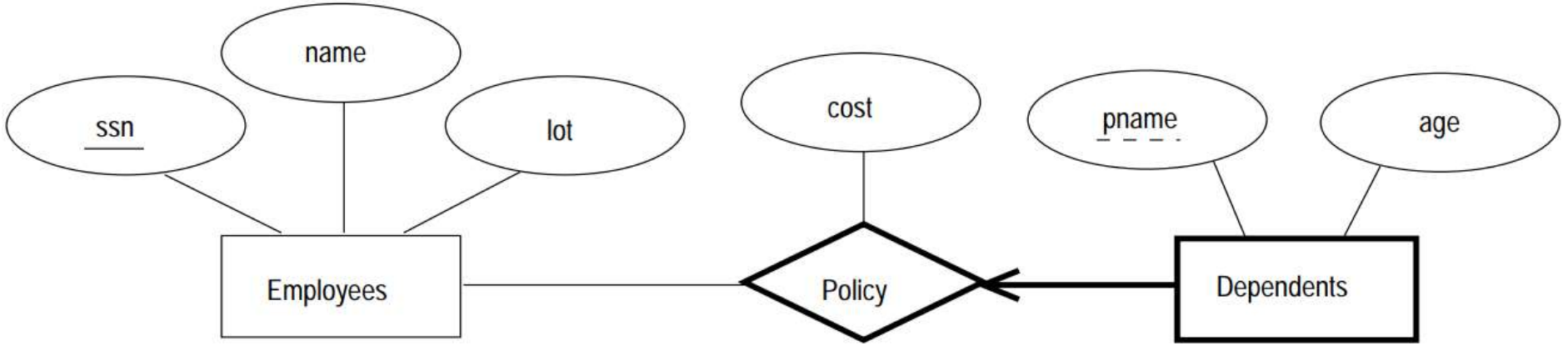



```
CREATE TABLE Works_In2 ( ssn       CHAR(11),
                         did       INTEGER,
                         address   CHAR(20),
                         since     DATE,
                         PRIMARY KEY (ssn, did, address),
                         FOREIGN KEY (ssn) REFERENCES Employees,
                         FOREIGN KEY (address) REFERENCES Locations,
                         FOREIGN KEY (did) REFERENCES Departments )
```




```
CREATE TABLE Reports_To (
        supervisor_ssn    CHAR(11),
        subordinate_ssn   CHAR(11),
        PRIMARY KEY (supervisor_ssn, subordinate_ssn),
        FOREIGN KEY (supervisor_ssn) REFERENCES Employees(ssn),
        FOREIGN KEY (subordinate_ssn) REFERENCES Employees(ssn) )
```

Translating Weak Entity Sets




```
CREATE TABLE Dep_Policy ( pname    CHAR(20),
                          age      INTEGER,
                          cost     REAL,
                          ssn      CHAR(11),
                          PRIMARY KEY (pname, ssn),
                          FOREIGN KEY (ssn) REFERENCES Employees
                                  ON DELETE CASCADE )
```

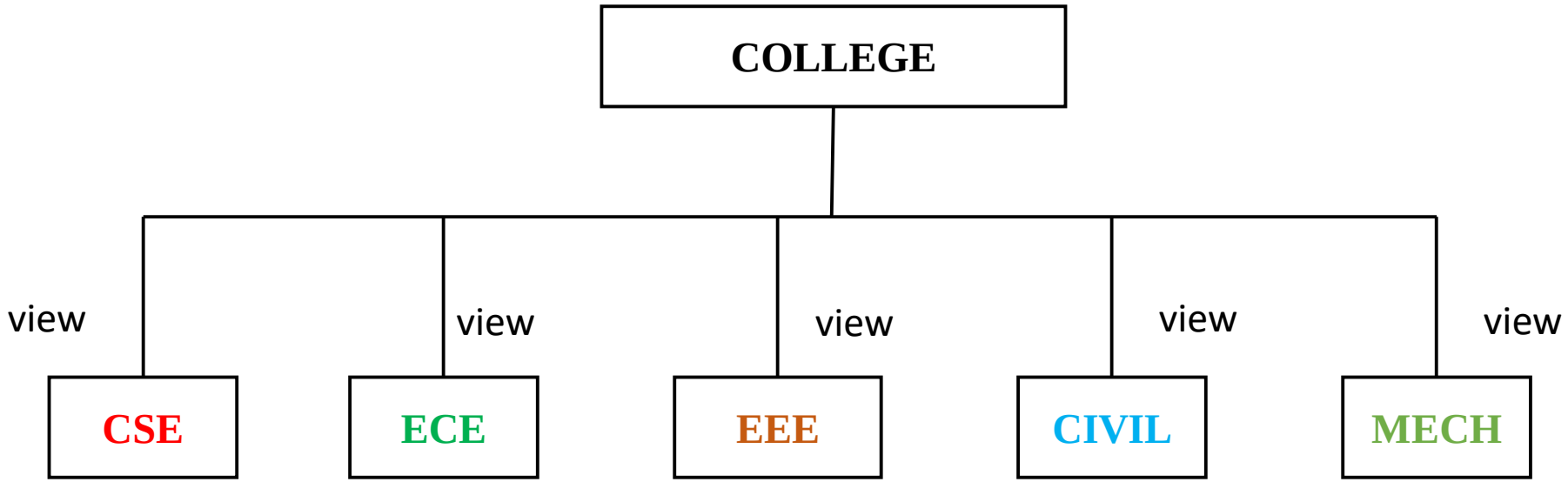
Introduction to Views

- A view is a table whose rows are not explicitly stored, a view is a virtual table constructed from the existing tables.
- A view can be created from one or many tables which depends on the written SQL query to create a view.
- A view is generated to show the information that the end-user requests the data according to specified needs rather than complete information of the table.
- Changes made in views reflects on table and vice versa.

Advantages:

- Using Views, we can join multiple tables into a single virtual table.
- In the database, views take less space than tables for storing data because the database contains only the view definition.
- Views indicate the subset of that data, which is contained in the tables of the database.

Example



College maintains database of different branches. When we try to modify the data in any branch it will automatically reflects in the college data base by using VIEWS.

Syntax for creation of View

```
create view view_name as
    select column1,column2, ……..
    from table_name where condition;
```

Example:

```
create view cse_stud as
    select * from student where branch=‘cse’;
```

Student table

```
SQL> select * from student;

       SID SNAME                          PERC GENDE BRANCH
---------- -------------------- ---------- ----- ----------
       101 hari                         99 male  cse
       102 monu                         90 femal ece
       103 sandeep                      95 male  cse
       104 janu                         78 femal eee
       105 charan                       88 male  cse
       106 yamini                       98 femal civil
       107 ramu                         60 male  ece
       108 gree                         55 male  cse
       109 sree                         55 male  ece
       110 pramila                      67 femal ece
```

Creation of a view from one table

```
SQL> create view cse_stud as
  2  select * from student
  3  where branch='cse';

View created.
```

```
SQL> select * from cse_stud;

       SID SNAME                          PERC GENDE BRANCH
---------- -------------------- ---------- ----- -------
       101 hari                         99 male  cse
       103 sandeep                      95 male  cse
       105 charan                       88 male  cse
       108 gree                         55 male  cse
```

Modifying views/ modifying master tables

```
SQL> select * from student;

       SID SNAME                          PERC GENDE BRANCH
---------- -------------------- ---------- ----- -------
       101 hari                         99 male  cse
       102 monu                         90 femal ece
       103 sandeep                      95 male  cse
       104 janu                         78 femal eee
       105 charan                       88 male  cse
       106 yamini                       98 femal civil
       107 ramu                         60 male  ece
       108 gree                         55 male  cse
       109 sree                         55 male  ece
       110 pramila                      67 femal ece
```

```
SQL> update student set branch ='ece'
  2  where sid=108;

1 row updated.
```

```
SQL> select * from cse_stud;

       SID SNAME                          PERC GENDE BRANCH
---------- -------------------- ---------- ----- -------
       101 hari                         99 male  cse
       103 sandeep                      95 male  cse
       105 charan                       88 male  cse
       108 gree                         55 male  cse
```

```
SQL> select * from cse_stud;

       SID SNAME                          PERC GENDE BRANCH
---------- -------------------- ---------- ----- -------
       101 hari                         99 male  cse
       103 sandeep                      95 male  cse
       105 charan                       88 male  cse
```

```
SQL> select * from ece_stud;

       SID SNAME                          PERC GENDE BRANCH
---------- -------------------- ---------- ----- -------
       102 monu                         90 femal ece
       107 ramu                         60 male  ece
       108 gree                         55 male  ece
       109 sree                         55 male  ece
       110 pramila                      67 femal ece
```

Creating views from more than one table

```
SQL> select * from student_marks;

       SID NAME                           MARKS        AGE
---------- -------------------- ---------- ----------
         1 harini                         96         20
         2 manisha                        90         19
         3 divya                          94         21
         4 kushi                          93         19
         5 amitha                         95         21
```

```
SQL> select * from student_details;

       SID NAME                 ADDRESS
---------- -------------------- ----------
         1 harini               kolkata
         2 preity               hyderabad
         3 divya                chennai
         4 kushi                mumbai
         5 amitha               banglore
```

```
SQL> create view marks_view as select
  2  student_details.name,student_details.address,student_marks.marks
  3  from student_details,student_marks
  4  where student_details.name=student_marks.name;

View created.
```

```
SQL> select * from marks_view;

NAME                 ADDRESS                              MARKS
-------------------- ------------------------------ ----------
harini               kolkata                                96
divya                chennai                                94
kushi                mumbai                                 93
amitha               banglore                               95
```

```
SQL> update student_details set address='hyderabad' where  name='kushi';

1 row updated.

SQL> select * from marks_view;

NAME                 ADDRESS                          MARKS
-------------------- ------------------------------ ----------
harini               kolkata                            96
divya                chennai                            94
kushi                hyderabad                          93
amitha               banglore                           95

SQL> select * from student_details;

       SID NAME                 ADDRESS
---------- -------------------- ------------------------------
         1 harini               kolkata
         2 preity               hyderabad
         3 divya                chennai
         4 kushi                hyderabad
         5 amitha               banglore
```

- When we try to modify the marks_view table it displays an error message “cannot modify” because the marks_view table is created from the two existing tables called “student_details’ and “student_marks” .

```
SQL> update marks_view set address='mumbai' where  name='kushi';
update marks_view set address='mumbai' where  name='kushi'
                      *
ERROR at line 1:
ORA-01779: cannot modify a column which maps to a non key-preserved table
```

Criteria for View Updating

- The **select statement** used in the create view statement **should not include group by clause or order by clause**
- The select statement **must not contain distinct keyword**
- A view **should not be created from nested or Complex queries**
- A view should be **created from a single table** but **if the view is created from more than one table then it is not allowed for updating**

Create / Replace Views

Syntax:

CREATE OR REPLACE VIEW VIEW_NAME AS

SELECT COLUMN1,COLUMN2,…

FROM TABLE_NAME WHERE CONDITION;

```
SQL> create or replace view marks_view as
  2  select student_details.name,student_details.address,student_marks.marks,student_marks.age
  3  from student_details,student_marks
  4  where student_details.name=student_marks.name;

View created.
```

```
SQL> create or replace view marks_view as
  2  select student_details.name,student_details.address,student_marks.marks,student_marks.age
  3  from student_details,student_marks
  4  where student_details.name=student_marks.name;

View created.

SQL> select * from marks_view;

NAME                 ADDRESS                         MARKS        AGE
-------------------- ------------------------------ ---------- ----------
harini               kolkata                            96         20
divya                chennai                            94         21
kushi                mumbai                             93         19
amitha               banglore                           95         21
```

Deleting and updating views in DBMS

- Delete a view by using the DROP statement.

Syntax:

DROP VIEW VIEW_NAME;

Example:

DROP VIEW CSE_STUD;

DROP VIEW MARKS_VIEW;

```
SQL> DROP VIEW CSE_STUD;

View dropped.

SQL> DROP VIEW MARKS_VIEW;

View dropped.
```

Updating view:

- Views are ***updated only if certain conditions are met*** otherwise if any one of the conditions are not met views will not be updated.

Inserting a row into a views

- To insert a row in a view just like inserting a row in an ordinary table.

```
SQL> insert into cse_stud values(111,'deepu',77,'femal','cse');

1 row created.

SQL> select * from cse_stud;

       SID SNAME                          PERC GENDE BRANCH
---------- -------------------- ---------- ----- ----------
       101 hari                         99 male  cse
       103 sandeep                      95 male  cse
       105 charan                       88 male  cse
       111 deepu                        77 femal cse

SQL> select * from student;

       SID SNAME                          PERC GENDE BRANCH
---------- -------------------- ---------- ----- ----------
       101 hari                         99 male  cse
       102 monu                         90 femal ece
       103 sandeep                      95 male  cse
       104 janu                         78 femal eee
       105 charan                       88 male  cse
       106 yamini                       98 femal civil
       107 ramu                         60 male  ece
       108 gree                         55 male  ece
       109 sree                         55 male  ece
       110 pramila                      67 femal ece
       111 deepu                        77 femal cse

11 rows selected.
```

Deleting a row from a view

- A row in a view **can be deleted just like simply deleting rows from a Table using delete statement.**
- But remember a row in a view **can be deleted only if the row is actually deleted in the original table from which it is created.**

```
SQL> delete from cse_stud where sname='deepu';

1 row deleted.

SQL> select * from student;
```

```
SQL> select * from cse_stud;

       SID SNAME                          PERC GENDE BRANCH
---------- ------------------------------ ---------- ----- --------
       101 hari                               99 male  cse
       103 sandeep                            95 male  cse
       105 charan                             88 male  cse
```

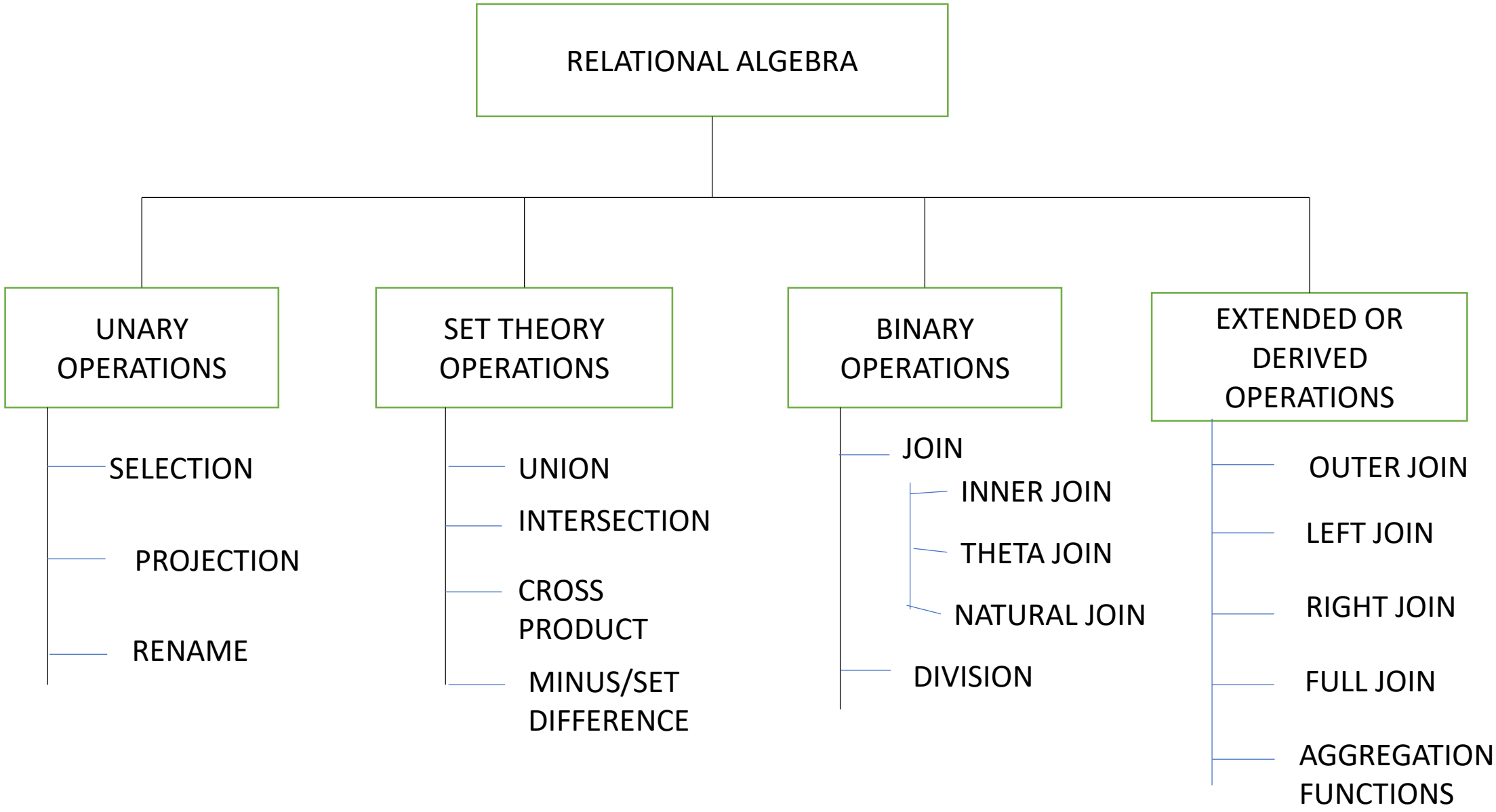
Syntax:

Delete from view_name

where condition;

Relational Algebra

- Relational algebra is a procedural query language. It gives a step by step process to obtain the result of the query. It uses operators to perform queries.
- It is a theoretical model and base for SQL.
- In relational mode, the data can retrieved theoretically or practically.
- Theoretical means -> Relational Algebra, Relational Calculus
- Practical means-> Through SQL queries


RELATIONAL ALGEBRA
UNARY OPERATIONS
SELECTION
PROJECTION
RENAME
SET THEORY OPERATIONS
UNION
INTERSECTION
CROSS PRODUCT
MINUS/SET DIFFERENCE
BINARY OPERATIONS
JOIN
INNER JOIN
THETA JOIN
NATURAL JOIN
DIVISION
EXTENDED OR DERIVED OPERATIONS
OUTER JOIN
LEFT JOIN
RIGHT JOIN
FULL JOIN
AGGREGATION FUNCTIONS

UNARY OPERATIONS

- Unary operations performed on single relation / single table
- **Selection operation**
  - Represented with sigma (σ)
  - Horizontal subset of a relation (gives row /tuple as a result)
  - Syntax: σ (condtion) (relation_name)

Examples:

σ (student) – displays all rows

σ (course =‘cs101’ and grade=‘A’) (student) -> display the entire row

σ (perc < 90) (student)

Student

Name	Course	Grade
Alice	CS101	A
Bob	CS101	B+
Alice	CS102	A-
Charlie	CS101	C+

- **Projection:**
  - used to select a subset of columns from a table.
  - Projection (Π/ π)
  - Vertical subset of a relation (columns)
  - Syntax: π (col1,col2….coln) (relation)

Example:
Π (Name) (student)
Π (Roll, per)(student)
Π (roll,per) (σ per>90) (student)

Name	Course	Grade
Alice	CS101	A
Bob	CS101	B+
Alice	CS102	A-
Charlie	CS101	C+

- **Rename operation**
  - Represented with symbol RHO(ρ)
  - Syntax: ρ (new_name) (relation_name)

Example:
ρ (final_students) (student)

Student

Name	Course	Grade
Alice	CS101	A
Bob	CS101	B+
Alice	CS102	A-
Charlie	CS101	C+

SET THEORY OPERATIONS

- Set Theory Operations can be performed on 2 tables or relations.

UNION(U) :

- Let R and S be two relations.
- Then-
  - R ∪ S is the set of all tuples belonging to either R or S or both.
  - In R ∪ S, duplicates are automatically removed.

Relation R

ID	Name	Subject
100	Ankit	English
200	Pooja	Maths
300	Komal	Science

Relation S

ID	Name	Subject
100	Ankit	English
400	Kajol	French

Relation R ∪ S

ID	Name	Subject
100	Ankit	English
200	Pooja	Maths
300	Komal	Science
400	Kajol	French

Intersection (∩):

- Let R and S be two relations.
- Then-
  - R ∩ S is the set of all tuples belonging to both R and S.
  - In R ∩ S, duplicates are automatically removed.
  - Intersection operation is both commutative and associative.

Relation R

ID	Name	Subject
100	Ankit	English
200	Pooja	Maths
300	Komal	Science

Relation S

ID	Name	Subject
100	Ankit	English
400	Kajol	French

Relation R ∩ S

ID	Name	Subject
100	Ankit	English

Minus/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.

ID	Name	Subject
100	Ankit	English
200	Pooja	Maths
300	Komal	Science

ID	Name	Subject
100	Ankit	English
400	Kajol	French

ID	Name	Subject
200	Pooja	Maths
300	Komal	Science

Relation R – S

Cross product operation (X):

- The Cartesian product is used to combine each row in one table with each row in the other table. It is also known as a cross product.
- Notation: E X D
- Example: (Employee X DEPARTMENT)

EMPLOYEE

EMP_ID	EMP_NAME	EMP_DEPT
1	SMITH	A
2	HARRY	C
3	JOHN	B

DEPARTMENT

DEPT_NUM	DEPT_NAME
A	MARKETING
B	SALES
C	LEGAL

Output:

EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO	DEPT_NAME
1	Smith	A	A	Marketing
1	Smith	A	B	Sales
1	Smith	A	C	Legal
2	Harry	C	A	Marketing
2	Harry	C	B	Sales
2	Harry	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales
3	John	B	C	Legal

JOINS

- Joins (⋈) in DBMS is used to combine tables.
- There are three types of joins: **inner joins and outer joins**.
- Inner joins are classified into three types: Theta Join(for relational operators), natural join and Equi Join(for Equality).
- There are three types of outer joins in DBMS: left outer join, right outer join, and full outer join.
- Natural join is only performed when at least one matching attribute exists in both tables.
- No matter the Join condition, a left outer join always returns every row from the left table.
- Regardless of the Join condition, Right Outer Join always returns all rows from the right table.
- Regardless of the join condition, Complete Outer Join always returns all rows from both tables.

Join = cartesian product + selection

RELATION R

A	B
1	2
3	4

RELATION S

C	D
5	6
7	8

RELATION R X S

A	B	C	D
1	2	5	6
1	2	7	8
3	4	5	6
3	4	7	8

Inner join

Theta join: A Theta Join uses a condition other than equality to join two tables. In this relations will be join with respect to condition and type of attribute. The attributes names may not be equal. [⋈(condition)]

- Example: C ⋈ J.price > C.price J

Jeep_Model	Price	Car_Model	Price
Jeep1	600000	Car1	400000
Jeep1	600000	Car2	500000
Jeep2	1000000	Car1	400000
Jeep2	1000000	Car2	600000
Jeep3	1000000	Car3	800000

Car_Model	Price
Car1	400000
Car2	500000
Car3	800000

C

Jeep_Model	Price
Jeep1	600000
Jeep2	1000000

J

Natural Join: Natural join can join tables based on the common columns in the tables being joined. A natural join returns all rows by matching values in common columns having same name and data type of columns and that column should be present in both tables.

Example: R ⋈ S

SID	SNAME	DeptID
101	RAJU	1
102	RAVI	2
103	HARI	1
104	RAMU	3

DeptID	DNAME
1	CSE
2	ECE
3	EEE
4	MECH

SID	SNAME	DeptID	DNAME
101	RAJU	1	CSE
102	RAVI	2	ECE
103	HARI	1	CSE
104	RAMU	3	EEE

R ⋈ S

Equi Join: R1 ⋈ condition R2

- R1 ⋈ (R1. DeptID = R2.DeptID) R2

SID	SNAME	DeptID
101	RAJU	1
102	RAVI	2
103	HARI	1
104	RAMU	3

DeptID	DNAME
1	CSE
2	ECE
3	EEE
4	MECH

SID	SNAME	DeptID	DNAME
101	RAJU	1	CSE
102	RAVI	2	ECE
103	HARI	1	CSE
104	RAMU	3	EEE

Outer Join

Left Outer Join: A SQL operation that combines two tables, showing all rows from the left table and matching rows from the right table. If there is no match in the right table, it displays null values.(S ⟕ R).

SID	SNAME	DeptID
101	RAJU	1
102	RAVI	2
103	HARI	1
104	RAMU	3
105	SURESH	5

DeptID	DNAME
1	CSE
2	ECE
3	EEE
4	MECH

S ⟕ R

SID	SNAME	DeptID	DNAME
101	RAJU	1	CSE
102	RAVI	2	ECE
103	HARI	1	CSE
104	RAMU	3	EEE
105	SURESH	5	NULL

- **Right outer join(⟖):** RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join. (S ⟖ R)

SID	SNAME	DeptID
101	RAJU	1
102	RAVI	2
103	HARI	1
104	RAMU	3
105	SURESH	5

DeptID	DNAME
1	CSE
2	ECE
3	EEE
4	MECH

S ⟖ R

SID	SNAME	DeptID	DNAME
101	RAJU	1	CSE
102	RAVI	2	ECE
104	RAMU	3	EEE
NULL	NULL	4	MECH
103	HARI	1	CSE

- **Full Join (⟗)**: FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both tables. For the rows for which there is no matching, the result-set will contain *NULL* values. (S ⟗ R)

SID	SNAME	DeptID
101	RAJU	1
102	RAVI	2
103	HARI	1
104	RAMU	3
105	SURESH	5

DeptID	DNAME
1	CSE
2	ECE
3	EEE
4	MECH

S ⟗ R

SID	SNAME	DeptID	DNAME
101	RAJU	1	CSE
102	RAVI	2	ECE
103	HARI	1	CSE
104	RAMU	3	EEE
105	SURESH	5	NULL
NULL	NULL	4	MECH

EXAMPLE 2 on joins

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	HARSH	DELHI	1234	18
2	PARTIK	BIHAR	1243	19
3	RIYANKA	SILGURI	5433	30
4	DEEP	RAMNAGAR	6444	47
5	SAPTARHI	KOLKATA	6666	39
6	DHANRAJ	BARABAJAR	4444	28
7	ROHIT	BALURGHAT	7777	56
8	NIRAJ	ALIPUR	3333	32

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

INNER JOIN

ROLL_NO	NAME	ADDRESS	PHONE	AGE	COURSE_ID
1	HARSH	DELHI	1234	18	1
2	PARTIK	BIHAR	1243	19	2
3	RIYANKA	SILGURI	5433	30	2
4	DEEP	RAMNAGAR	6444	47	3
5	SAPTARHI	KOLKATA	6666	39	1

OUTER JOIN – LEFT OUTER JOIN

ROLL_NO	NAME	ADDRESS	PHONE	AGE	COURSE_ID
1	HARSH	DELHI	1234	18	1
2	PARTIK	BIHAR	1243	19	2
3	RIYANKA	SILGURI	5433	30	2
4	DEEP	RAMNAGAR	6444	47	3
5	SAPTARHI	KOLKATA	6666	39	1
6	DHANRAJ	BARABAJAR	4444	28	NULL
7	ROHIT	BALURGHAT	7777	56	NULL
8	NIRAJ	ALIPUR	3333	32	NULL

OUTER JOIN – RIGHT OUTER JOIN

ROLL_NO	NAME	ADDRESS	PHONE	AGE	COURSE_ID
1	HARSH	DELHI	1234	18	1
2	PARTIK	BIHAR	1243	19	2
3	RIYANKA	SILGURI	5433	30	2
4	DEEP	RAMNAGAR	6444	47	3
5	SAPTARHI	KOLKATA	6666	39	1
9	NULL	NULL	NULL	NULL	4
10	NULL	NULL	NULL	NULL	5
11	NULL	NULL	NULL	NULL	4

OUTER JOIN – FULL JOIN

ROLL_NO	NAME	ADDRESS	PHONE	AGE	COURSE_ID
1	HARSH	DELHI	1234	18	1
2	PARTIK	BIHAR	1243	19	2
3	RIYANKA	SILGURI	5433	30	2
4	DEEP	RAMNAGAR	6444	47	3
5	SAPTARHI	KOLKATA	6666	39	1
6	DHANRAJ	BARABAJAR	4444	28	NULL
7	ROHIT	BALURGHAT	7777	56	NULL
8	NIRAJ	ALIPUR	3333	32	NULL
9	NULL	NULL	NULL	NULL	4
10	NULL	NULL	NULL	NULL	5
11	NULL	NULL	NULL	NULL	4

DIVISON OPERATION

- The division operator is used for queries which involve the 'all'.
- R1 ÷ R2 = tuples of R1 associated with all tuples of R2.

Example

- Retrieve the name of the subject that is taught in all courses.

Name	Course
System	Btech
Database	Mtech
Database	Btech
Algebra	Btech

÷

Course
Btech
Mtech

=

Name
database

Example : Retrieve names of employees who work on all the projects that John Smith works on.

Consider the Employee table given below –

Name	Eno	Pno
John	123	P1
Smith	123	P2
A	121	P3

÷

Works on the following –

Eno	Pno	Pname
123	P1	Market
123	P2	Sales

=

The result is as follows

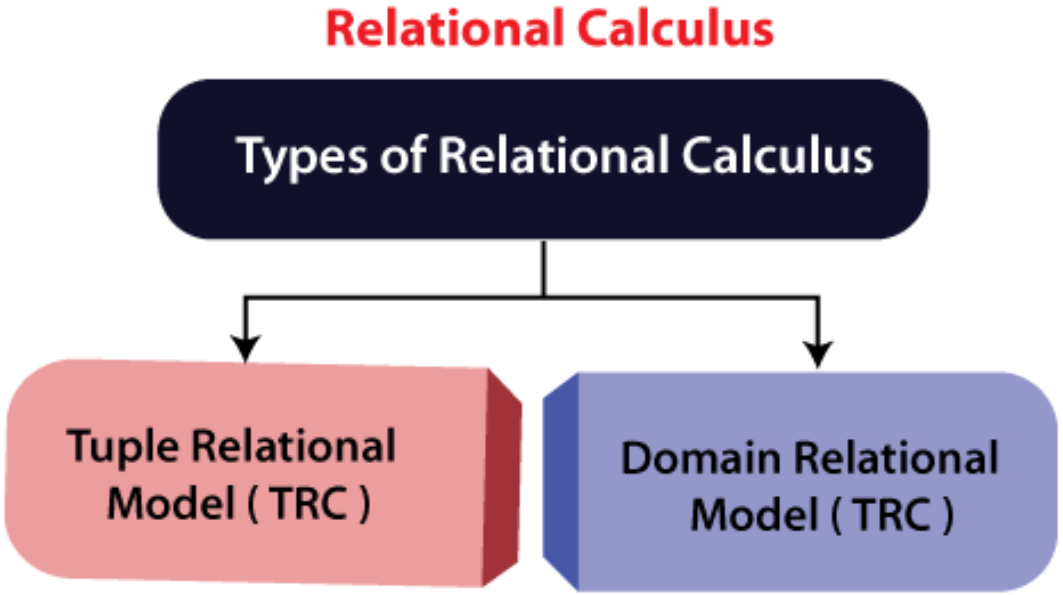
Eno
123

Relational calculus

- Relational calculus in DBMS is a non-procedural language, which only focuses on what kind of data is required and it does not care about how to get these data.

Example: Let's take an example to understand what is relational calculus in DBMS.

SELECT the tuples from **EMPLOYEE** relation with **DEPARTMENT= 'NETWORKING'**



Tuple Relational Calculus (TRC)

- TRC in DBMS is based on the concept of selecting tuples (rows) from a relation (table) that satisfies certain conditions.
- Tuple Relational Calculus uses variables to represent tuples and logical predicates to specify the conditions that must be met for the tuples to be selected.
- The resulting expression is a formula that describes a set of tuples that meet the specified conditions.
- The general syntax for Tuple Relational Calculus is:

```
{ t | P (t) } or { t | Condition (t) }
```

- t is the resulting tuples, and P(t) is the condition used to get t

Example: Now, we will take the database table Employee and try to apply the Tuple Relational Calculus expression on that table

Emp_id	Emp_name	Department
101	Naimish	Computer
102	Sahil	Finance
103	Divyesh	Computer
104	Nikunj	Account
105	Gautam	Computer
106	Vishal	HR
107	Meet	Business
108	Milan	Business

output

Emp_id	Emp_name	Department
101	Naimish	Computer
103	Divyesh	Computer
105	Gautam	Computer

TRC Query : {t | t ∈ Employee ∧ t.Department = ’Computer’} or

TRC Query: {t | Employee(t) ∧ t[Department] = ‘Computer’}

Domain Relational Calculus :

- Domain Relational Calculus in DBMS is based on the concept of selecting values from a relation (table) that satisfy certain conditions.
- Domain Relational Calculus uses variables to represent individual values, and logical predicates to specify the conditions that must be met for the values to be selected.
- The resulting expression is a formula that describes a set of values that meet the specified conditions.
- General Syntax:

$$\{ < x_1, x_2, x_3, \ldots, x_n > \mid P (x_1, x_2, x_3, \ldots, x_n) \}$$

- where, $< x_1, x_2, x_3, \ldots, x_n >$ represents resulting domains variables and $P (x_1, x_2, x_3, \ldots, x_n)$ represents the condition or formula equivalent to the Predicate calculus

Example : Now, we will take the database table Employee and try to apply the Domain Relational Calculus expression on that table.

Emp_id	Emp_name	Department
101	Naimish	Computer
102	Sahil	Finance
103	Divyesh	Computer
104	Nikunj	Account
105	Gautam	Computer
106	Vishal	HR
107	Meet	Business
108	Milan	Business

Query:

```
{ \|  ∈ Employee∧ x3 = ‘Computer’}
```

Emp_id	Emp_name	Department
101	Naimish	Computer
103	Divyesh	Computer
105	Gautam	Computer