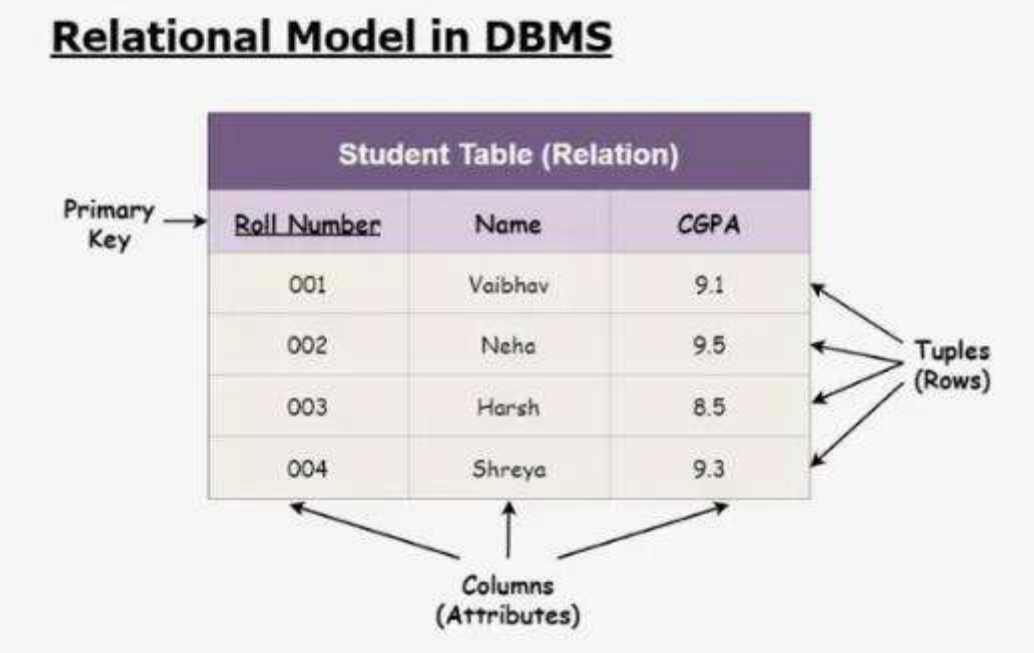
UNIT - 2
Introduction to the Relational Model

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

1. Attribute: Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME,etc.
2. 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.
3. Tuple – It is nothing but a single row of a table, which contains a single record.
4. Relation Schema: A relation schema represents the name of the relation with its attributes.
5. Degree: The total number of attributes which in the relation is called the degree of the relation.
6. Cardinality: Total number of rows present in the Table.
7. Column: The column represents the set of values for a specific attribute.
8. Relation instance – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
9. Relation key – Every row has one, two or multiple attributes, which is called relation key.

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

Advantages of Realation Model

- Structural Independence: Structural independence is an ability that allows us to make changes in one database structure without affecting other. The relational model have structural independence. Hence making required changes in the database is convenient in relational database model.
- Conceptual Simplicity: The relational model allows the designer to simply focus on logical design and not on physical design. Hence relational models are conceptually simple to understand.
- Query Capability: Using simple query language (such as SQL) user can get information from the database or designer can manipulate the database structure.
- Easy design,maintenance and usage: The relational models can be designed logically hence they are easy to maintain and use.

✓ Keys in DBMS

KEYS in DBMS is an attribute or set of attributes which helps you to identify a row(tuple) in a relation(table). They allow you to find the relation between two tables. Keys help you uniquely identify a row in a table by a combination of one or more columns in that table.

Employee ID	FirstName	LastName
11	Andrew	Johnson
22	Tom	Wood
33	Alex	Hale

In the above-given example, employee ID is a primary key because it uniquely identifies an employee record. In this table, no other employee can have the same employee ID.

Types of Keys in DBMS

There are seven different types of Keys in DBMS and each key has it's different functionality:

1. Super Key
2. Primary Key
3. Candidate Key
4. Alternate Key
5. Foreign Key
6. Composite Key
7. Surrogate Key

Super key:

- A superkey is a group of single or multiple keys which identifies rows in a table. Super Key is a superset of Candidate key.

Emp_SSN	Emp_Number	Emp_Name
123456789	226	Steve
999999321	227	Ajeet
888997212	228	Chaitanya
777778888	229	Robert

- All of the following sets of super key are able to uniquely identify a row of the employee table.
- {Emp_SSN}
- {Emp_Number}
- {Emp_SSN, Emp_Number}
- {Emp_SSN, Emp_Name}
- {Emp_SSN, Emp_Number, Emp_Name}
- {Emp_Number, Emp_Name}

Candidate Key:

CANDIDATE KEY in SQL is a set of attributes that uniquely identify tuples in a table. Candidate Key is a super key with no repeated attributes. The Primary key should be selected from the candidate keys. Every table must have at least a single candidate key. A table can have multiple candidate keys but only a single primary key.

Emp_SSN	Emp_Number	Emp_Name
123456789	226	Steve
999999321	227	Ajeet
888997212	228	Chaitanya
777778888	229	Robert

Super keys are:

1. {Emp_Id}
2. {Emp_Number}
3. {Emp_Id, Emp_Number}
4. {Emp_Id, Emp_Name}
5. {Emp_Id, Emp_Number, Emp_Name}
6. {Emp_Number, Emp_Name}

- The candidate keys we have selected are:
    - {Emp_Id}
    - {Emp_Number}

A primary key is selected from the set of candidate keys. That means we can either have Emp_Id or Emp_Number as primary key. The decision is made by DBA.

Primary Key:

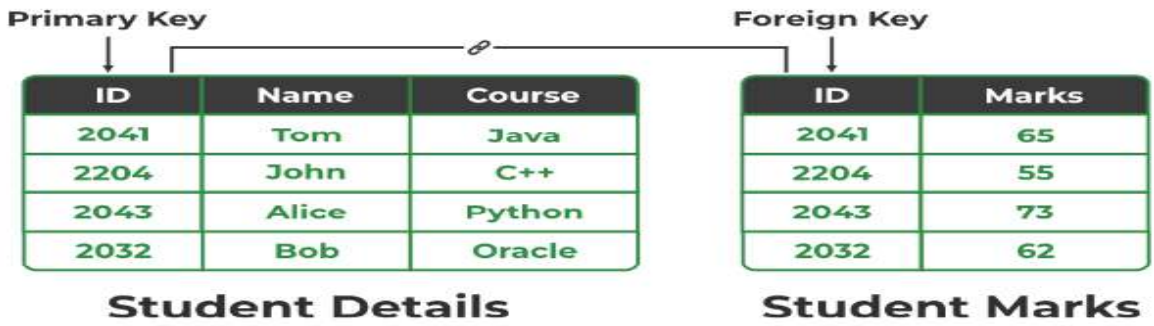
PRIMARY KEY in DBMS is a column or group of columns in a table that uniquely identify every row in that table. The Primary Key can't be a duplicate meaning the same value can't appear more than once in the table. A table cannot have more than one primary key.

In the following example, StudID is a primary key.

StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com

Foreign key:

FOREIGN KEY is a column that creates a relationship between two tables. The purpose of Foreign keys is to maintain data integrity and allow navigation between two different instances of an entity. It acts as a cross-reference between two tables as it references the primary key of another table.



Composite Key:

Sometimes, a table might not have a single column/attribute that uniquely identifies all the records of a table. To uniquely identify rows of a table, a combination of two or more

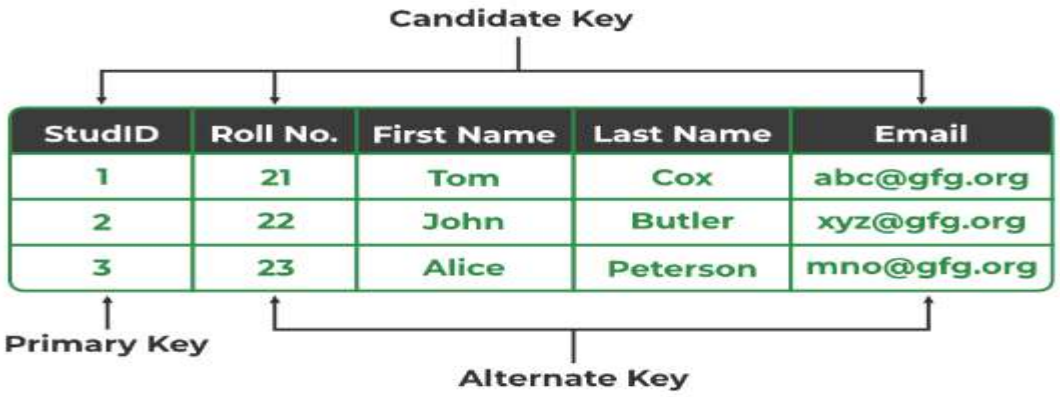
columns/attributes can be used. It still can give duplicate values in rare cases. So, we need to find the optimal set of attributes that can uniquely identify rows in a table.

- It acts as a primary key if there is no primary key in a table
- Two or more attributes are used together to make a composite key.
- Different combinations of attributes may give different accuracy in terms of identifying the rows uniquely.

Alternate Key:

The candidate key other than the primary key is called an alternate key.

- All the keys which are not primary keys are called alternate keys.
- It is a secondary key.
- It contains two or more fields to identify two or more records.
- These values are repeated.

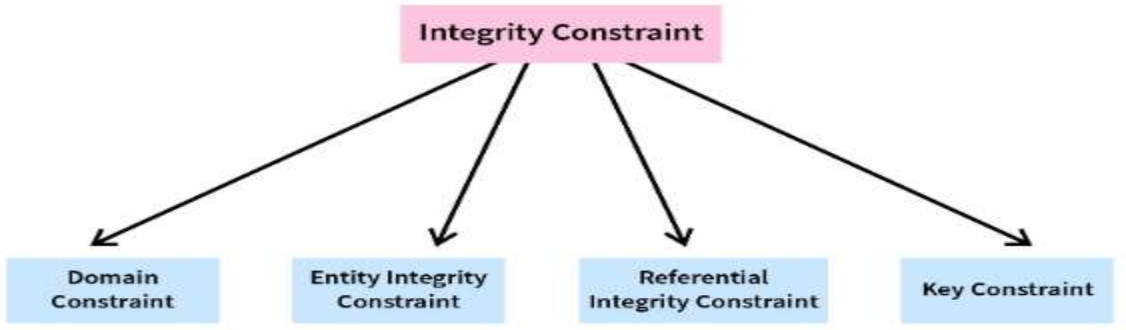


Surrogate key:

SURROGATE KEYS is An artificial key which aims to uniquely identify each record is called a surrogate key. This kind of partial key in dbms is unique because it is created when you don't have any natural primary key.

✓ Integrity Constraints in DBMS

In database management systems (DBMS) there is a certain set of rules which are used to maintain the quality and consistency of data in the database. Every time there is an insertion, deletion, or updating of data in the database it is the responsibility of these integrity constraints to maintain the integrity of data and thus help to prevent accidental damage to the database.



There are four types of integrity constraints in DBMS:

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

1. Domain Constraints

1. Every domain must contain atomic values(smallest indivisible units) which means composite and multi-valued attributes are not allowed.
2. We perform a datatype check here, which means when we assign a data type to a column we limit the values that it can contain. Eg. If we assign the datatype of attribute age as int, we can't give it values other than int datatype.

Example:

EID	Name	Phone
01	Bikash Dutta	123456789 234456678

Explanation: In the above relation, Name is a composite attribute and Phone is a multi-values attribute, so it is violating domain constraint.

Example:

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 Constraint

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

Example:

EID	Name	Phone
01	Bikash	9000900099
02	Paul	600000009
NULL	Sony	9234567892

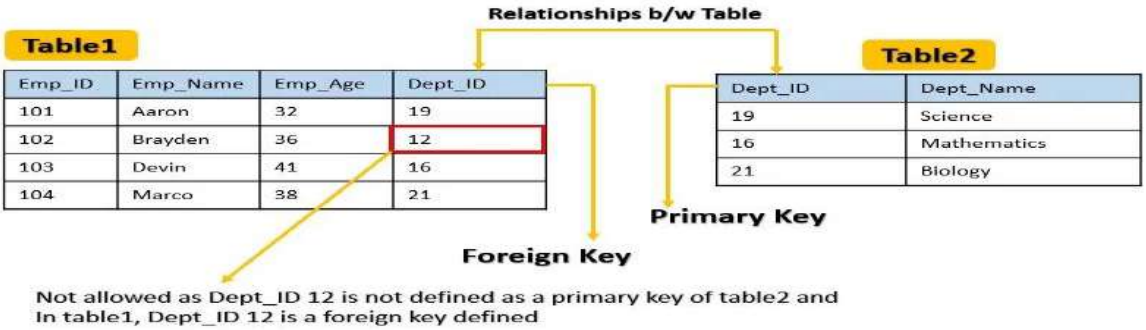
Explanation: In the above relation, EID is made the primary key, and the primary key can't take NULL values but in the third tuple, the primary key is null, so it is violating Entity Integrity constraints.

Emp_ID	Emp_Name	Emp_City	Emp_Age
101	Aaron	New York	32
102	Brayden	California	36
103	Devin	Chicago	41
	Marco	Houston	38

3. Referential Integrity Constraint

Referential Integrity Constraint ensures that there must always exist a valid relationship between two relational database tables. This valid relationship between the two tables confirms that a foreign key exists in a table. It should always reference a corresponding value or attribute in the other table or be null.

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 set of entities that are used to identify an entity within its entity set uniquely. There could be multiple keys in a single entity set, but out of these multiple keys, only one key will be the primary key. A primary key can only contain unique and not null values in the relational database table.

NOT NULL:

NOT NULL constraint makes sure that a column does not hold NULL value. When we don't provide value for a particular column while inserting a record into a table, it takes NULL value by default. By specifying NULL constraint, we can be sure that a particular column(s) cannot have NULL values.

Example:

```
CREATE TABLE STUDENT(
ID INT NOT NULL,
NAME VARCHAR (35) NOT NULL,
AGE INT NOT NULL,
PRIMARY KEY (ID)
);
```

```
SQL> select * from student;

        ID NAME                           AGE
---------- --------------- ----------
         1 bc                             23
         2 abc                            24
         3 abcd                           25
         4 bcd                            26
         5 cd                             30

SQL> insert into student values(NULL,'venkatesh',26);
insert into student values(NULL,'venkatesh',26)
                           *
ERROR at line 1:
ORA-01400: cannot insert NULL into ("SYSTEM"."STUDENT"."ID")
```

UNIQUE:

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

```
SQL> create table employee(id int PRIMARY KEY,name varchar(20),sal int,age int check(age>18));

Table created.
```

```
SQL> select * from employee;

        ID NAME                           SAL        AGE
---------- -------------------- ---------- ----------
         1 venkatesh                   40000         21
         2 krishna                     50000         27

SQL> insert into employee values(1,'sagar',30000,27);
insert into employee values(1,'sagar',30000,27)
*
ERROR at line 1:
ORA-00001: unique constraint (SYSTEM.SYS_C004046) violated
```

CHECK:

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

```
SQL> create table employee(id int PRIMARY KEY,name varchar(20),sal int,age int check(age>18));

Table created.
```

```
SQL> select * from employee;

        ID NAME                           SAL        AGE
---------- -------------------- ---------- ----------
         1 venkatesh                   40000         21
         2 krishna                     50000         27

SQL> insert into employee values(2,'krishna',50000,17);
insert into employee values(2,'krishna',50000,17)
*
ERROR at line 1:
ORA-02290: check constraint (SYSTEM.SYS_C004045) violated
```

DEFAULT:

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

```
SQL> alter table employee add address varchar(10) default 'hyd';

Table altered.

SQL> select * from employee;

        ID NAME                           SAL        AGE ADDRESS
---------- -------------------- ---------- ---------- ----------
         1 venkatesh                   40000         21 hyd
         2 krishna                     50000         27 hyd
```

PRIMARY KEY:

Primary key uniquely identifies each record in a table. It must have unique values and cannot contain nulls. The primary key has automatically UNIQUE and NOT NULL constraints applied to it. It is usually used to index the table or uniquely identify each tuple in the table.
We can say that:
PRIMARY KEY CONSTRAINT = UNIQUE CONSTRAINT + NOTNULL CONSTRAINT

```
SQL> create table employee(id int PRIMARY KEY,name varchar(20),sal int,age int check(age>18));

Table created.
```

```
SQL> select * from employee;

        ID NAME                           SAL        AGE ADDRESS
---------- -------------------- ---------- ---------- ----------
         1 venkatesh                   40000         21 hyd
         2 krishna                     50000         27 hyd

SQL> insert into employee values(2,'krishna',50000,27,default);
insert into employee values(2,'krishna',50000,27,default)
*
ERROR at line 1:
ORA-00001: unique constraint (SYSTEM.SYS_C004046) violated


SQL> insert into employee values(NULL,'krishna',50000,27,default);
insert into employee values(NULL,'krishna',50000,27,default)
                            *
ERROR at line 1:
ORA-01400: cannot insert NULL into ("SYSTEM"."EMPLOYEE"."ID")
```

FOREIGN KEY Constraint:

Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables. So FOREIGN KEY constraint prevents operations that can destroy the link between the tables. Foreign Keys are used in situations where we do not want the deletion of data from one table to hamper the data in the other table.

One important thing to note here is that - the Foreign Key of one table points to or references the Primary Key of the second table. So, when we delete data from the main table that points to data in the other table, we will be shown the error - **Record in child table exists**.

```
SQL> CREATE TABLE Customers1( Cid INT PRIMARY KEY, first_name VARCHAR(40), age INT, country VARCHAR(10));

Table created.

SQL> CREATE TABLE Orders (orderid INT,product VARCHAR(40),FOREIGN KEY (orderid) REFERENCES Customers1(cid));

Table created.
```

```
SQL> insert into customers1 values(101,'mahesh',28,'india');

1 row created.

SQL> insert into customers1 values(102,'kalyan',29,'india');

1 row created.

SQL> insert into customers1 values(104,'venkatesh',27,'india');

1 row created.

SQL> select * from customers1;

       CID FIRST_NAME                                        AGE COUNTRY
---------- ---------------------------------------- ---------- ----------
       101 mahesh                                           28 india
       102 kalyan                                           29 india
       104 venkatesh                                        27 india
```

```
SQL> insert into orders values(101,'mobile');
1 row created.
SQL> insert into orders values(102,'Laptop');
1 row created.
SQL> insert into orders values(102,'hardisk');

1 row created.
```

```
SQL> select * from orders;

   ORDERID PRODUCT
---------- ----------------------------------------
       101 mobile
       102 Laptop
       102 hardisk

SQL> insert into orders values(103,'Laptop');
insert into orders values(103,'Laptop')
*
ERROR at line 1:
ORA-02291: integrity constraint (SYSTEM.SYS_C004049) violated - parent key not
found
```

Querying Relational Data in DBMS

A relational database query is a question about the data, and the answer consists of a new relation containing the result. For example, we might want to find all students AGE less than 18 or all students enrolled in perticular course.
The ***SELECT*** statement is used to fetch the data from a database table which returns this data in the form of a result table. These result tables are called result-sets.

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

```
SELECT * FROM table_name;
```

The symbol ´*****´ means that we retain all fields of selected tuples in the result.

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

```
SELECT * FROM Students WHERE age < 18;
```

The condition **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.

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 student_id and First_name of students who are younger than 18 with the following query:

```
SELECT ID,FirstName FROM Students WHERE age < 18;
```

SQL Aliases

Aliases are the temporary names given to tables or columns. An alias is created with the **AS** keyword.

Alias Column Syntax :

```
SELECT column_name AS alias_name FROM table_name;
```

Alias Table Syntax :

```
SELECT column_name(s) FROM table_name AS alias_name;
```

Example:

```
SELECT studentID AS ID, FROM students AS S;
```

Aliases can be useful when:

- There are more than one table involved in a query
- Functions are used in the query
- Column names are big or not very readable
- Two or more columns are combined togeth

SELECT data from Multiple Tables:

We can also combine information from multiple tables.

Syntax :

```
SELECT table1.column1, table2.column2
FROM table1, table2
WHERE table1.column1 = table2.column1;
```

Example:

```
SELECT S.name, E.cid
FROM Students AS S, Enrolled AS E
WHERE S.sid = E.sid;
```

Views in DBMS:

- Writing Complex queries and Securing Database access is very challenging for any Database Developer and Database Administrator.
- Sometimes SQL queries get very complicated by joins, Group By clauses, and other referential dependencies, So those Types of queries can be simplified to proxy data or virtual data which simplifies the queries.
- Suppose, the user needs only 2 columns of data, so instead of giving him access to the whole table in the database, the Database Administrator can easily create a virtual table of 2 columns that the user needs using the views.
- This will not give full access to the table and the user is only seeing the projection of only 2 columns and it keeps the database secure.
- View is a virtual table that contains rows and columns, just like a real table.
- It is used to hide complexity of query from user.
- A virtual table does not exist physically, it is created by a SQL statement that joins one or more tables.
- Views in DBMS can be visualized as virtual tables that are formed by original tables from the database.

Creating Views:

- Database views are created using the CREATE VIEW statement. Views can be created from a single table, multiple tables or another view.
- To create a view, a user must have the appropriate system privilege according to the specific implementation.

Syntax:

- CREATE VIEW view_name AS SELECT column1, column2..... FROM table_name WHERE [condition];

```
SQL> create view studview as select * from student;

View created.

SQL> select * from studview;

        ID NAME                          AGE          FEE
---------- ---------------- ---------- ----------
         1 bc                           23        60000
         2 abc                          24        60000
         3 abcd                         25        60000
         4 bcd                          26        60000
         5 cd                           30        60000
         1 venkatesh                    26        60000

6 rows selected.
```

Creating View from multiple tables:

- View from multiple tables can be created by simply include multiple tables in the SELECT statement.

Syntax:

- CREATE VIEW MarksView AS SELECT Student_Detail.NAME, Student_Detail.ADDRESS, Student_Marks.MARKS FROM Student_Detail, Student_Mark WHERE Student_Detail.NAME = Student_Marks.NAME;

Deleting View

A view can be deleted using the Drop View statement.

Syntax

DROP VIEW view_name;

Example:

If we want to delete the View **MarksView**, we can do this as:

DROP VIEW MarksView;

```
SQL> drop view studview;

View dropped.
```

Inserting a Row in a View:

- Inserting a row in the view takes the same syntax as we use to insert a row in a simple table.

Syntax:

- INSERT INTO ViewName(column1, column2,..) VALUES(value1, value2,..);

```
SQL> insert into studview values(10,'krishna',23,50000);

1 row created.

SQL> select * from studview;

        ID NAME                          AGE        FEE
---------- --------------- ---------- ----------
         1 bc                          23      60000
         2 abc                         24      60000
         3 abcd                        25      60000
         4 bcd                         26      60000
         5 cd                          30      60000
        10 krishna                     23      50000
         1 venkatesh                   26      60000

7 rows selected.
```

Deleting a Row in a View:

- Deleting a row in the view takes the same syntax as we use to delete a row in a simple table.

Syntax:

- DELETE FROM ViewName WHERE condition;

```
SQL> delete from studview where name='abcd';

1 row deleted.
```

UPDATING VIEWS:

There are certain conditions needed to be satisfied to update a view. If any one of these conditions is **not** met, then we will not be allowed to update the view.

1. The SELECT statement which is used to create the view should not include GROUP BY clause or ORDER BY clause.
2. The SELECT statement should not have the DISTINCT keyword.
3. The View should have all NOT NULL values.
4. The view should not be created using nested queries or complex queries.
5. The view should be created from a single table. If the view is created using multiple tables then we will not be allowed to update the view.

We can use the **CREATE OR REPLACE VIEW** statement to add or remove fields from a view.

Syntax:

CREATE OR REPLACE VIEW view_name AS SELECT column1,column2,.. FROM table_name WHERE condition;

For example, if we want to update the view **MarksView** and add the field AGE to this View from **StudentMarks** Table, we can do this as:

CREATE OR REPLACE VIEW MarksView AS SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS, StudentMarks.AGE FROM StudentDetails, StudentMarks WHERE StudentDetails.NAME = StudentMarks.NAME;

WITH CHECK OPTION:

The WITH CHECK OPTION clause in SQL is a very useful clause for views. It is applicable to an updatable view. If the view is not updatable, then there is no meaning of including this clause in the CREATE VIEW statement.

- The WITH CHECK OPTION clause is used to prevent the insertion of rows in the view where the condition in the WHERE clause in CREATE VIEW statement is not satisfied.
- If we have used the WITH CHECK OPTION clause in the CREATE VIEW statement, and if the UPDATE or INSERT clause does not satisfy the conditions then they will return an error.

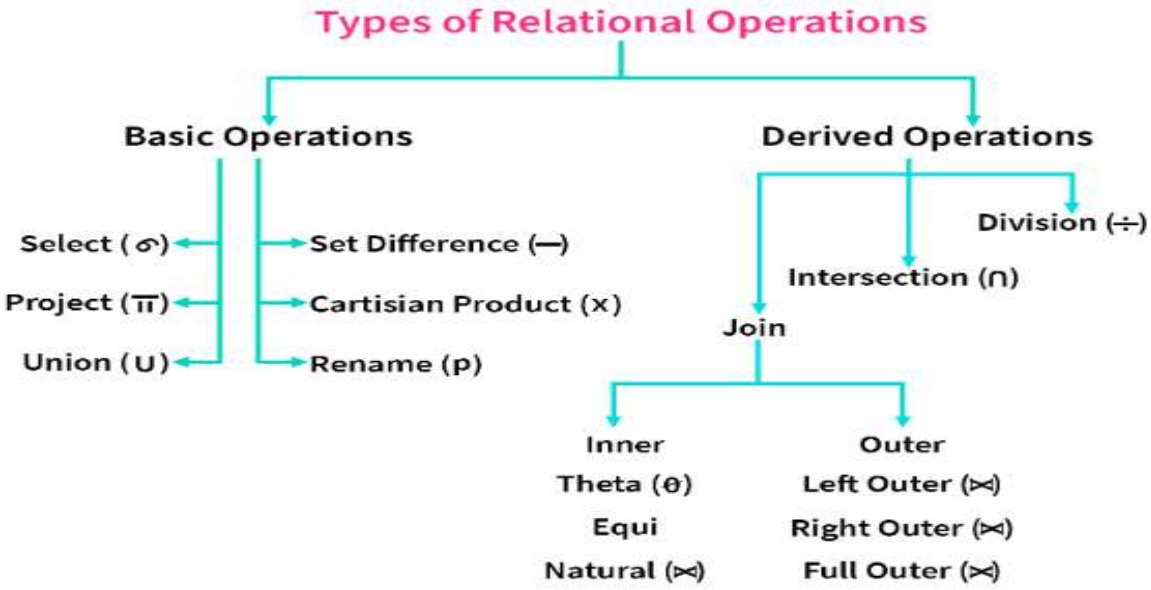
Example: In the below example we are creating a View SampleView from StudentDetails Table with WITH CHECK OPTION clause.

CREATE VIEW SampleView AS SELECT S_ID, NAME FROM StudentDetails WHERE NAME IS NOT NULL WITH CHECK OPTION;

In this View if we now try to insert a new row with null value in the NAME column then it will give an error because the view is created with the condition for NAME column as NOT NULL.

Relational Algebra in DBMS

- Relational algebra refers to a procedural query language that takes relation instances as input and returns relation instances as output.
- Queries in relational algebra are performed using operators. A binary or unary operator can be used.
- A programmer/user has to mention two things, "What to Do" and "How to Do".
- Relational algebra is follows a particular syntax with the help of which, data can be accessed and retrieved very easily from single as well as multiple table/data sources.



Select (σ):

- Select operation is done by Selection Operator which is represented by "sigma"(σ).

Syntax : σ <Condition> (Relation Name)

- It is used to retrieve tuples(rows) from the table where the given condition is satisfied.
- It is a **unary operator** means it requires only one operand.
- Where σ is used to represent SELECTION,R is used to represent RELATION

Example:

Student_Details

Roll No	Name	Age	Marks
1	**Anoop**	**22**	**30**
2	**Anurag**	**23**	**32**
3	**Ganesh**	**21**	**31**

Query Used :

σ Name and Age>21 (Student_Details)

Name	Age
Anoop	**22**
Anurag	**23**

Ex 2:

```
Table: CUSTOMER
---------------

Customer_Id      Customer_Name       Customer_City
-----------      -------------       -------------
C10100            Steve               Agra
C10111            Raghu               Agra
C10115            Chaitanya           Noida
C10117            Ajeet               Delhi
C10118            Carl                Delhi
```

Query:

σ Customer_City="Agra" (CUSTOMER)

```
Customer_Id    Customer_Name     Customer_City
-----------    -------------     -------------
C10100         Steve             Agra
C10111         Raghu             Agra
```

Project (∏):

- Project operation is done by Projection Operator which is represented by "pi"(∏).
- Using project operation one can simply fetch all the tuples corresponding to a single attribute or multiple attributes.

- It is also known as vertical partitioning as it separates the table vertically. It is also a unary operator.

Syntax : $\pi_{<attribute>}$(Relation Name)

Example:

Student_Details

Roll No	Name	Age	Marks
1	**Anoop**	**22**	**30**
2	**Anurag**	**23**	**32**
3	**Ganesh**	**21**	**31**

Query Used :

π_{Marks}(Student_Details)

Query Output

Marks
30
32
31

Example 2:

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra
C10115	Chaitanya	Noida
C10117	Ajeet	Delhi
C10118	Carl	Delhi

Query:

∏ Customer_Name, Customer_City (CUSTOMER)

Output

Customer_Name	Customer_City
Steve	Agra
Raghu	Agra
Chaitanya	Noida
Ajeet	Delhi
Carl	Delhi

Union (∪):

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

- It is a binary operator as it requires two operands.
- Also, two things need to keep in mind while applying union operation are :
  - ❖ Both the relations compulsory to have same number of attributes.
  - ❖ Both the relations compulsory to have same domain for attributes.

Consider the two tables with relations X_1(Name, Age) and X_2(Name, Age).

- If we wish to apply the union operation, then it can be done by

X1(Name, Age)

Name	Age
Anoop	22
Anurag	23

∪

X2(Name, Age)

Name	Age
Ganesh	21
Saurav	22
Rakesh	20

Query Output(X1 U X2)

Name	Age
Anoop	22
Anurag	23
Ganesh	21
Saurav	22
Rakesh	20

Set Difference (-):

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

Syntax of Set Difference (-)

```
table_name1 - table_name2
```

Example:

Table 1: COURSE

Course_Id	Student_Name	Student_Id
C101	Aditya	S901
C104	Aditya	S901
C106	Steve	S911
C109	Paul	S921
C115	Lucy	S931

Table 2: STUDENT

Student_Id	Student_Name	Student_Age
S901	Aditya	19
S911	Steve	18
S921	Paul	19
S931	Lucy	17
S941	Carl	16
S951	Rick	18

Query:

∏ Student_Name (STUDENT) - ∏ Student_Name (COURSE)

Output:

Student_Name
Carl
Rick

Cartesian product (X):

Cartesian product is denoted by the "X" symbol. Let's say we have two relations R and S. Cartesian product will combine every tuples (row) from R with all the tuples from S. It combines the information of two or more relations in one single relation.

Notation: R X S

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

Example:

Table 1: R

Col_A	Col_B
AA	100
BB	200
CC	300

Table 2: S

Col_X	Col_Y
XX	99
YY	11
ZZ	101

Query:

Lets find the cartesian product of table R and S.

R X S

Output:

Col_A	Col_B	Col_X	Col_Y
AA	100	XX	99
AA	100	YY	11
AA	100	ZZ	101
BB	200	XX	99
BB	200	YY	11
BB	200	ZZ	101
CC	300	XX	99
CC	300	YY	11
CC	300	ZZ	101

Intersection Operator (∩):

Intersection operator is denoted by ∩ symbol and it is used to select common rows (tuples) from two tables (relations).

Syntax of Intersection Operator (∩)

table_name1 ∩ table_name2

Table 1: COURSE

Course_Id	Student_Name	Student_Id
C101	Aditya	S901
C104	Aditya	S901
C106	Steve	S911
C109	Paul	S921
C115	Lucy	S931

Table 2: STUDENT

Student_Id	Student_Name	Student_Age
S901	Aditya	19
S911	Steve	18
S921	Paul	19
S931	Lucy	17
S941	Carl	16
S951	Rick	18

Query:

∏ Student_Name (COURSE) ∩ ∏ Student_Name (STUDENT)

Output:

Student_Name
Aditya
Steve
Paul
Lucy

JOINS in DBMS

Databases usually have more than one table. JOINs are an SQL construction used to join data from two or more tables. When you want to use columns from two tables in a result table, the easiest way to do it is to write a JOIN query. JOIN Keyword is used in SQL queries for joining two or more tables.

The syntax of an SQL JOIN is:

```
SELECT * FROM table1 JOIN table2 ON table1.id1=table2.id2
```

Types of JOIN

Following are the types of JOIN that we can use in SQL:

- Inner Join
- Outer Join
  - Left outer Join
  - Right Outer Join
  - Full Outer Join

Inner Join:

This is a simple JOIN in which the result is based on matched data as per the equality condition specified in the SQL query.

Inner Join Syntax is,

```
SELECT column-name-list FROM table-name1 INNER JOIN table-name2 WHERE table-name1.column-name = table-name2.column-name;
```

Example;

Consider a **class** table,

ID	NAME
1	abhi
2	adam
3	alex
4	anu

and the **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

Inner JOIN query will be,

```
SELECT * from class INNER JOIN class_info where class.id = class_info.id;
```

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI

Natural JOIN

Natural Join is a type of Inner join which is based on column having same name and same datatype present in both the tables to be joined.
The syntax for Natural Join is,

```
SELECT * FROM table-name 1 NATURAL JOIN table-name 2;
```

Example:

Consider a **class** table,

ID	NAME
1	abhi
2	adam
3	alex
4	anu

and the **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

Natural join query will be,

```
SELECT * from class NATURAL JOIN class_info;
```

Output:

ID	NAME	Address
1	abhi	DELHI
2	adam	MUMBAI
3	alex	CHENNAI

OUTER JOIN

Outer Join is based on both matched and unmatched data. Outer Joins subdivide further into,

1. Left Outer Join

2. Right Outer Join
3. Full Outer Join

LEFT Outer Join

The left outer join returns a resultset table with the **matched data** from the two tables and then the remaining rows of the **left** table and null from the **right** table's columns.
Syntax:

```
SELECT column-name FROM table1 LEFT OUTER JOIN table-2 ON table-1.column-name = table-2.column-name;
```

Table : song

song_id	song_title	artist_id
100	Dancing Queen	04
101	Imagine	02
102	I Will Survive	01
103	Rolling in the deep	06
104	Respect	03
105	Mamma Mia	04

Table : Artist

artist_id	artist_name
01	Gloria Gaynor
02	John Lennon
03	Aretha Franklin
04	ABBA
05	Michael Jackson

Query:

```
SELECT * FROM song LEFT JOIN artist ON song.artist_id=artist.artist_id
```

song_id	song_title	artist_id	artist_id	artist_name
100	Dancing Queen	04	04	ABBA
101	Imagine	02	02	John Lennon
102	I Will Survive	01	01	Gloria Gaynor
103	Rolling in the deep	06	NULL	NULL
104	Respect	03	03	Aretha Franklin
105	Mamma Mia	04	04	ABBA

RIGHT Outer Join

The right outer join returns a resultset table with the **matched data** from the two tables being joined, then the remaining rows of the **right** table and null for the remaining **left** table's columns.
Syntax for Right Outer Join is,

```
SELECT column-name  FROM  table1 RIGHT OUTER JOIN table-2 ON table1.column-name = table2.column-name;
```

Query:
SELECT * FROM song RIGHT JOIN artist ON song.artist_id=artist.artist_id

Output:

song_id	song_title	artist_id	artist_id	artist_name
100	Dancing Queen	04	04	ABBA
101	Imagine	02	02	John Lennon
102	I Will Survive	01	01	Gloria Gaynor
104	Respect	03	03	Aretha Franklin
105	Mamma Mia	04	04	ABBA
NULL	NULL	NULL	05	Michael Jackson

Full Outer Join

The full outer join returns a resultset table with the **matched data** of two table then remaining rows of both **left** table and then the **right** table.

Syntax:

```
SELECT column-name FROM table1 FULL OUTER JOIN table2 ON table1.column-name = table2.column-name;
```

Query:
SELECT * FROM song FULL JOIN artist ON song.artist_id=artist.artist_id

Output:

song_id	song_title	artist_id	artist_id	artist_name
100	Dancing Queen	04	04	ABBA
101	Imagine	02	02	John Lennon
102	I Will Survive	01	01	Gloria Gaynor
103	Rolling in the deep	06	NULL	NULL
104	Respect	03	03	Aretha Franklin
105	Mamma Mia	04	04	ABBA
NULL	NULL	NULL	05	Michael Jackson

Relational Calculus

Procedural Language - Those Languages which clearly define how to get the required results from the Database are called Procedural Language.
Relational algebra is a Procedural Language.
Declarative Language - Those Language that only cares about What to get from the database without getting into how to get the results are called Declarative Language.
Relational Calculus is a Declarative Language.

Relational calculus is a non-procedural query language, and instead of algebra, it uses mathematical predicate calculus.
Relational Calculus in database management system (DBMS) is all about ***"What you want ?"***.
Relational calculus does not tell us how to get the results from the Database, but it just cares about what we want.
The relational calculus tells what to do but never explains how to do.

Relational Calculus is of Two Types:

- Tuple Relational Calculus (TRC)
- Domain Relational Calculus (DRC)

Tuple Relational Calculus (TRC):

- **Tuple Relational Calculus (TRC)** is a non-procedural query language used in relational database management systems (RDBMS) to retrieve data from tables.
- TRC is a declarative language, meaning that it specifies what data is required from the database, rather than how to retrieve it.
- Tuple Relational Calculus in DBMS uses a tuple variable (t) that goes to each row of the table and checks if the predicate is true or false for the given row.
- Depending on the given predicate condition, it returns the row or part of the row.

- Table: Student

First_Name	Last_Name	Age
Ajeet	Singh	30
Chaitanya	Singh	31
Rajeev	Bhatia	27
Carl	Pratap	28

- Query to display the last name of those students where age is greater than 30

- { t.Last_Name | Student(t) AND t.age > 30 }
- In the above query you can see two parts separated by | symbol. The second part is where we define the condition and in the first part we specify the fields which we want to display for the selected tuples.
- The result of the above query would be:

Last_Name
Singh

- Query to display all the details of students where Last name is ‘Singh’

- { t | Student(t) AND t.Last_Name = 'Singh' }

Output:

First_Name	Last_Name	Age

Ajeet	Singh	30
Chaitanya	Singh	31

2. Domain Relational Calculus (DRC)

- In domain relational calculus the records are filtered based on the domains. Again we take the same table to understand how DRC works.
  Table: Student

First_Name	Last_Name	Age
Ajeet	Singh	30
Chaitanya	Singh	31
Rajeev	Bhatia	27
Carl	Pratap	28

- Query to find the first name and age of students where student age is greater than 27

$$\{< \text{First\_Name}, \text{Age} > \mid \in \text{Student} \land \text{Age} > 27\}$$

Note:
The symbols used for logical operators are: ∧ for AND, ∨ for OR and ¬ for NOT.

- **Output:**

First_Name	Age
Ajeet	30
Chaitanya	31
Carl	28