

UNIT II STRUCTURED QUERY LANGUAGE

Basics of SQL, DDL, DML, DCL, TCL – creation, alteration, defining constraints – Functions – aggregate functions, Built-in functions – Views – Joins – Procedure

1. SQL FUNDAMENTALS

IBM developed the original version of SQL, originally called Sequel, as part of the System R project in the early 1970s. The Sequel language has evolved since then, and its name has changed to SQL (Structured Query Language)

1.1 Data Definition Language (DDL)

The Data Definition Language is used to define the overall schema of the database. The DDL is also used to specify additional properties of the data. DDL deals with database schemas and descriptions, of how the data should reside in the database.

Example of DDL are:

- **CREATE** : to create database
- **ALTER** : alters the structure of the existing database
- **DROP** : delete objects from the database
- **TRUNCATE** : remove all records from a table, including all spaces allocated for the records are removed
- **RENAME**: Used to rename a relation

SQL Basic Data Types

The SQL standard supports a variety of built-in types, including:

- **char(*n*)**: A fixed-length character string with user-specified length *n*. The full form, **character**, can be used instead.
- **varchar(*n*)**: A variable-length character string with user-specified maximum length *n*. The full form, **character varying**, is equivalent.
- **int**: An integer (a finite subset of the integers that is machine dependent). The full form, **integer**, is equivalent.
- **smallint**: A small integer (a machine-dependent subset of the integer type).

- **numeric(p , d):** A fixed-point number with user-specified precision. The number consists of p digits (plus a sign), and d of the p digits are to the right of the decimal point. Thus, **numeric**(3,1) allows 44.5 to be stored exactly, but neither 444.5 or 0.32 can be stored exactly in a field of this type.
- **real, double precision:** Floating-point and double-precision floating-point numbers with machine-dependent precision.
- **float(n):** A floating-point number, with precision of at least n digits.

CREATE:

It is used to create a table or relation.

Syntax :

CREATE TABLE tablename ($A_1 D_1, A_2 D_2, \dots, A_n D_n, \langle \text{integrity-constraint}_1 \rangle, \dots$

, $\langle \text{integrity-constraint}_k \rangle$); where

tablename - the name of the table

A_i - the attribute name

D_i - the domain (data type) of the attribute $\text{integrity-constraint}_i$ - data integrity constraint name

Integrity Constraints:

Integrity Constraints enforce rules on the data in a table whenever a row is inserted, updated, or deleted from that table. The constraint must be satisfied for the operation to succeed. It prevents the deletion of a table if there are dependencies from other tables.

Primary Key Constraint: The Primary Key constraint uniquely identifies each record in a table. Primary keys must contain unique values, and cannot contain NULL values. A table can have only one primary key, which may consist of single or multiple fields.

Foreign Key Constraint: A foreign key is a key used to link two tables together. A foreign key is a field (or collection of fields) in one table that refers to the primary key in another table. The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

Unique Constraint: The unique constraint ensures that all values in a column are different. Both the unique and primary key constraints provide a guarantee for uniqueness for a column or set of columns. A primary key constraint automatically has a unique constraint.

NOT NULL Constraint: The NOT NULL constraint enforces a column to not accept NULL values. This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field

Check Constraint: The check constraint is used to limit the value range that can be placed in a column.

Default Constraint: The default constraint is used to provide a default value for a column.

Example

```
CREATE TABLE Employee (Eid
    int,
    Name varchar(255) NOT NULL,
    Designation varchar(255), Salary int ,
    check(Salary>0),
    Phno varchar(255) , PRIMARY KEY(Eid)
);
```

Result:

Table created.

To view the structure of the table, **DESCRIBE table_name**

DESCRIBE Employee

Name	NULL	Type
Eid		int
Name	NOT NULL	varchar(255)
Designation		varchar(255)
Salary		Int
Phno		varchar(255)

ALTER :

The alter table statement is used to add, delete, or modify columns in an existing table.

Add Clause:

The new column becomes the last column. If a table already contains rows when a column is added, then the new column is initially null for all the rows.

Modify Clause:

Column modification can include changes to a column's data type, size, and default value. the data type is changed only if the column contains null values. A change to the default value of a column affects only subsequent insertions to the table.

Drop Clause:

only one column can be dropped at a time. The table must have at least one column remaining in it after it is altered. Once a column is dropped, it cannot be recovered.

Syntax:

```
ALTER TABLE table_name ADD|MODIFY column_name datatype;
```

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

Where,

table_name - the name of the table

column_name – name of the attribute

datatype – domain of the attribute

Example 1:

```
ALTER TABLE employee ADD (Address varchar(255));
```

Result:

Table Altered.

```
DESCRIBE Employee
```

Name	NULL	Type
Eid		int
Name	NOT NULL	varchar(255)
Designation		varchar(255)
Salary		Int
Phno		varchar(255)
Address		varchar(255)

Example 2:

```
ALTER TABLE employee DROP Address;
```

Result:

Table Altered.

DESCRIBE Employee

Name	NULL	Type
Eid		int
Name	NOT NULL	varchar(255)
Designation		varchar(255)
Salary		Int
Phno		varchar(255)

DROP :

The DROP TABLE statement is used to drop an existing table in a database. The drop command deletes the entire table along with its content. The DROP TABLE statement cannot be rolled back.

Syntax:

```
DROP TABLE table_name;
```

Example:

```
DROP TABLE Employee;
```

Result:

Table Dropped.

```
DESC Employee;
```

No table found

RENAME :

It is used to rename a table.

Syntax

```
RENAME old_name TO new_name;
```

Where,

old_name - the old name of the table, view, sequence, or synonym. new_name

- the new name of the table, view, sequence, or synonym.

Example:

```
RENAME Employee TO emp;
```

Result:

Table Renamed.

TRUNCATE :

Truncate is used to remove all rows from a table and to release the storage space used by that table. It does not delete the table. The removed rows can not be rolled back.

Syntax:

```
TRUNCATE TABLE table_name;
```

Example:

TRUNCATE TABLE Employee.

1.2 Data-Manipulation Language(DML)

A data manipulation language (DML) is a language that enables users to access or manipulate data as organized by the appropriate data model.

DML commands are not auto-committed. It means that the changes are not permanent to database, they can be rolled back. Data manipulation is retrieval of information, insertion of new information, deletion of information or modification of information stored in the database.

DML statements are :

1.2.1.1 **SELECT** : retrieve data from the database

1.2.1.2 **INSERT** : insert data into a table

1.2.1.3 **UPDATE** : updates existing data in a table

1.2.1.4 **DELETE** : delete all records from a database

There are two types of DML :

1. **Procedural DML** : it requires a user to specify what data are needed and how to get those data.
2. **Non-Procedural DML** : it requires a user to specify what data are needed without specifying how to get those data.

The basic structure of an SQL query consists of three clauses: select, from, and where. The query takes as its input the relations listed in the from clause, operates on them as specified in the where and select clauses, and then produces a relation as the result. The select clause is used to list the attributes desired in the result of a query. The from clause is a list of the relations to be accessed in the evaluation of the query. The where clause is a predicate involving attributes of the relation in the from clause.

SELECT:

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

Syntax:

SELECT * | { [DISTINCT] column|expression [alias],... }FROM table_name [WHERE condition(s)]
[GROUP BY *group_by_expression*] [HAVING *group_condition*][ORDER BYcolumn];

Where,

* - selects all columns

DISTINCT - suppresses duplicates

column/expression - selects the named column or the expression

alias - gives selected columns different headings

Where Clause : restricts the query to rows that meet a condition
Group by -

Example :

Consider the relation Employee

<u>Eid</u>	Name	Designation	Salary	Phno
125	Kumar	Developer	25000	9489562312
128	Raju	Accountant	27500	9557886321
568	Sachin	HR	33000	8561234568
895	Dravid	Developer	45070	7288635241
222	Dhoni	Accountant	39650	8956235689

Query: Select * from Employee;

Result:

<u>Eid</u>	Name	Designation	Salary	Phno
125	Kumar	Developer	25000	9489562312
128	Raju	Accountant	27500	9557886321
568	Sachin	HR	33000	8561234568
895	Dravid	Developer	45070	7288635241
222	Dhoni	Accountant	39650	8956235689

Query: Select Name, Salary from Employee;

Result:

Name	Salary
Kumar	25000
Raju	27500
Sachin	33000
Dravid	45070
Dhoni	39650

Where Clause

It Restrict the rows returned by select statement.

Comparison Conditions:

- = - Equal to
- > - Greater than
- >= - Greater than or equal to
- < - Less than
- <= - Less than or equal to
- <> - Not equal to

BETWEEN - Between two values (inclusive)

IN(set) - Match any of a list of values (%)

LIKE - Match a character pattern

IS NULL - Is a null value

Logical Conditions:

AND - if all the conditions separated by AND is TRUE

OR - if any of the conditions separated by OR is TRUE

NOT - if the condition(s) is NOT TRUE

Rules of Precedence:

Arithmetic operators

Concatenation operator

Comparison conditions

IS [NOT] NULL, LIKE, [NOT] IN

NOT] BETWEEN

NOT logical condition

AND logical condition

OR logical condition

Query: Select Name, Salary from employee where Salary between 27500 and 40000

Result:

Name	Salary
Raju	27500
Sachin	33000
Dhoni	39650

Query: Select Name, Designation, Salary from Employee where Salary>25000 and Designation='Developer';

Result:

Name	Designation	Salary
Dravid	Developer	45070

Query: Select Eid, Name, Designation from Employee where Name like 'D%'

<u>Eid</u>	Name	Designation
895	Dravid	Developer
222	Dhoni	Accountant

Aggregate Functions

Aggregate functions are functions that take a collection of values as input and return a single value. SQL offers five built-in aggregate functions:

- **avg-** returns the average value of a numeric column.
- **min-** returns the smallest value of the selected column.
- **max-** returns the largest value of the selected column.

- **sum-** returns the total sum of a numeric column.
- **count-** returns the number of rows that matches a specified criteria.

Example:

Query : Select min(Salary) from Employee;

Result:

Min(Salary)
25000

Query: Select avg(Salary) from Employee where designation='Accountant';

Result: shows the average salary of employees with the designation Accountant

Avg(Salary)
33575

Group By Clause:

the GROUP BY clause to divide the rows in a table into groups. The attribute or attributes given in the **group by** clause are used to form groups. By default, rows are sorted by ascending order of the columns included in the GROUP BY list.

Query: select Designation, avg(Salary) from Employee group by Designation;

Result:

Designation	Avg(Salary)
Accountant	33575.0
Developer	36250
HR	33000

Having Clause:

The HAVING clause is used to restrict **groups**. The groups that match the criteria in the HAVING clause are displayed.

Query:

Select Designation, avg(Salary) as avg from Employee group by Designation having avg(Salary)>34000;

Result:

Designation	Avg
Developer	36250

INSERT:

It is used to insert a new row that contains values for each column. **Only one row is inserted at a time.**

Syntax:

INSERT INTO *table* (column1, column2, column3,) VALUES (*value1*, *value2*, *value3*, ...);

If the order of the values is in the same order as the columns in the table

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

Example

Query:

Insert into Employee(Eid, Designation, Name, Salary, Phno)values(587,'Designer','Walter',32570, '8978567780')

Result:

1 row created.

Eid	Name	Designation	Salary	Phno
125	Kumar	Developer	25000	9489562312
128	Raju	Accountant	27500	9557886321
568	Sachin	HR	33000	8561234568
895	Dravid	Developer	45070	7288635241
222	Dhoni	Accountant	39650	8956235689
587	Walter	Designer	32570	8978567780

UPDATE :

The UPDATE command is used to modify attribute values of one or more selected tuples.

Syntax:

UPDATE table_name SET column1 = value1, column2 = value2, ...WHERE condition;

Example:**Query:**

Update Employee set salary=43575 where Eid=587;

Result:

1 row updated.

<u>Eid</u>	Name	Designation	Salary	Phno
125	Kumar	Developer	25000	9489562312
128	Raju	Accountant	27500	9557886321
568	Sachin	HR	33000	8561234568
895	Dravid	Developer	45070	7288635241
222	Dhoni	Accountant	39650	8956235689
587	Walter	Designer	43575	8978567780

DELETE:

The DELETE statement is used to delete existing records in a table. The delete operation is not made permanent until the data transaction is committed. If the WHERE clause is omitted, all rows in the table are deleted.

Syntax:

DELETE FROM table_name WHERE condition;

Example:**Query:**

Delete from Employee where EID=587

Result:

1 row deleted.

<u>Eid</u>	Name	Designation	Salary	Phno
125	Kumar	Developer	25000	9489562312
128	Raju	Accountant	27500	9557886321
568	Sachin	HR	33000	8561234568
895	Dravid	Developer	45070	7288635241
222	Dhoni	Accountant	39650	8956235689

Query:

Delete from Employee;

Result:

5 row deleted.

1.3 Transaction Control Language(TCL)

TCL commands are to keep a check on other commands and their affect on the database. These commands can annul changes made by other commands by rolling back to original state. It can also make changes permanent.

TCL Statements are:

COMMIT : to permanently save

ROLLBACK : to undo change

SAVEPOINT : to save temporarily

COMMIT:

It makes the change permanent.

Example:**Query:**

Delete from Employee where EID=587

Result: Remove Employee 587 in the Employee table.

1 row deleted.

Query:

Commit;

Result: makes the change permanent

Commit complete.

ROLLBACK:

Data changes are undone. The previous state of the data is restored.

Example:

Query:

DELETE FROM Employee;

Result :all rows in the employee table is deleted

5 rows deleted.

Query:

ROLLBACK;

Result: deleted rows in the employee table is Roll backed.

SAVEPOINT:

It is used to create a marker in the current transaction which divides the transaction into smaller sections. Roll back to that marker by using the ROLLBACK TO SAVEPOINT statement.

Query:

Delete from Employee where EID=587

Result: Remove Employee 587 in the Employee table.

1 row deleted.

Query:

savepoint A

Result:

Savepoint created.

Query:

Delete from Employee where EID=222

Result: Remove Employee 222 in the Employee table.

1 row deleted.

Query:

ROLLBACK TO A;

Result:

Rollback complete.

<u>Eid</u>	Name	Designation	Salary	Phno
125	Kumar	Developer	25000	9489562312
128	Raju	Accountant	27500	9557886321
568	Sachin	HR	33000	8561234568
895	Dravid	Developer	45070	7288635241
587	Walter	Designer	43575	8978567780

1.4 Data Control Language (DCL)

It is used to control database access. It gives access to specific objects in the database. Roles, permissions, and referential integrity are created using data control language. Users can be given the privileges to access database. Two types of privileges are there.

System Privilege :

More than 100 distinct system privileges are available for users and roles. System privileges typically are provided by the database administrator.

Example:

```
CREATE SESSION
CREATE TABLE
CREATE SEQUENCE
CREATE VIEW
CREATE PROCEDURE
```

Object Privilege:

An *object privilege* is a privilege or right to perform a particular action on a specific table, view, sequence, or procedure.

Example:

```
ALTER
```

DELETE
EXECUTE
INDEX
INSERT
REFERENCES
SELECT
UPDATE

DCL statements:

GRANT
REVOKE

Create user:

The DBA creates users by using the CREATE USER statement.

Syntax:

```
CREATE USER user_name IDENTIFIED BY password;
```

Example:

Query:

```
CREATE USER Student IDENTIFIED BY welcome;
```

Result:

User created.

Where, user name is Student and the password is welcome.

Role:

A role is a named group of related privileges that can be granted to the user. This method makes it easier to revoke and maintain privileges. The DBA must create the role. Then the DBA can assign privileges to the role and users to the role.

Create role:

Syntax:

```
CREATE ROLE role;
```

Example:

```
CREATE ROLE Manager;
```

Role created.

GRANT:

Database Administrator (DBA) can grant specific privileges to a user or role. DBA can create user and role. Then the privilege is granted.

Grant System Privilege to user:

Syntax:

GRANT *privilege* [, *privilege*...] TO *user* [, *user*/ *role*, *PUBLIC*...];

Example:

Query:

GRANT create session, create table, create sequence, create view TO Student;

Result:

Grant succeeded.

For the student user, privileges (create session, create table, create sequence, create view) are granted.

Grant privileges to a role:

Syntax:

GRANT *privilege*(s) TO *role_name*;

Example:

GRANT create table, create view TO manager;

Grant succeeded.

Grant a role to users:

Syntax:

GRANT *role_name* TO *user_name*;

Example:

GRANT manager TO Student;

Result:

Grant succeeded.

create table, create view privileges are granted to the user student.

Grant Object privilege to user:

Syntax:

GRANT *object_priv* [(*columns*)] ON *object* TO {*user*|*role*|PUBLIC} [WITH GRANT OPTION];

Example:

GRANT select ON Employee TO Student;

Grant succeeded.

REVOKE:

The REVOKE statement is used to revoke privileges granted to other users.

Syntax:

REVOKE {*privilege* [, *privilege*...]|ALL} ON *object* FROM {*user*[, *user*...]|*role*|PUBLIC} [CASCADE CONSTRAINTS];

Example:

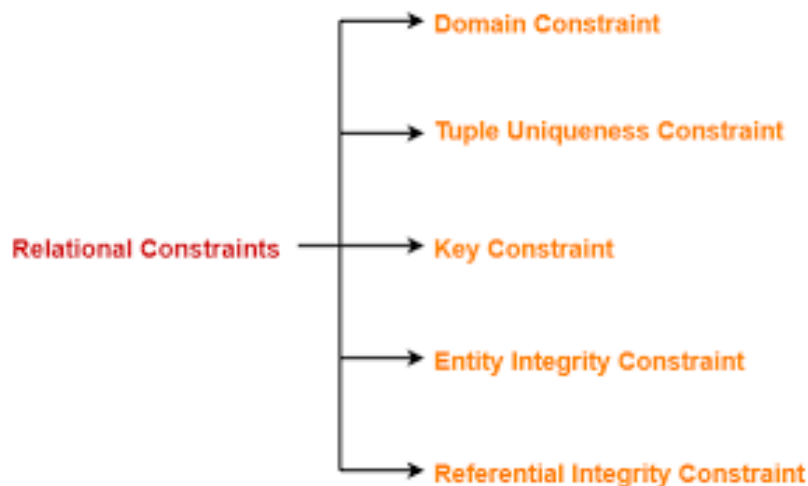
REVOKE select ON Employee FROM Student;

Revoke succeeded.

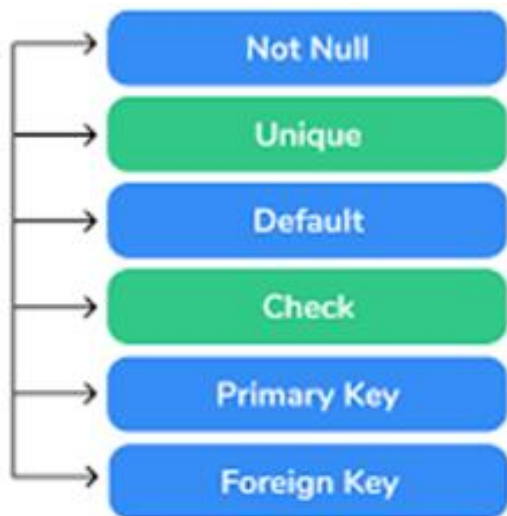
CONSTRAINTS:

Constraints are the rules enforced on the data columns of a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be either on a column level or a table level. The column level constraints are applied only to one column, whereas the table level constraints are applied to the whole table.



Following are some of the most commonly used constraints available in SQL:



1. Domain Constraint

- a. Domain of an attribute specifies all the possible values that attribute can hold
- b. It defines the domain or set of values for an attribute
- c. It specifies that the value taken by the attribute must be the atomic (can't be divided) value from its domain
- d. It contains
 - i. NOT NULL constraint
 - ii. CHECK constraint

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

Not allowed. Because AGE is an integer attribute

RollNo	Name	Age
1	Arya	21
2	Bran	19
3	John	24
4	Max	24

→ Domain

Age must be greater than 18 and must be an integer

2. Tuple Uniqueness Constraint

- a. It specifies that all the tuples must be necessarily unique in any relation.

<u>STU_ID</u>	Name	Age
S001	Akshay	20
S002	Abhishek	21
S003	Shashank	20
S004	Rahul	20

The above relation satisfies Tuple Uniqueness Constraint since all attributes are unique

<u>STU_ID</u>	Name	Age
S001	Akshay	20
S001	Akshay	20
S003	Shashank	20
S004	Rahul	20

The above relation does not satisfies Tuple Uniqueness Constraint since all attributes are not unique

3. Key Constraint

- All the values of primary key must be unique.
- The value of primary key must not be null.

<u>STU_ID</u>	Name	Age
S001	Akshay	20
S001	Akshay	20
S003	Shashank	20
S004	Rahul	20

The above relation does not satisfies Tuple Uniqueness Constraint since all attributes are not unique

4. Entity Integrity Constraint

- It specifies that no attribute of primary key must contain a null value in any relation.
- This is because the presence of null value in the primary key violates the uniqueness property.

<u>STU_ID</u>	Name	Age
S001	Akshay	20
S002	Abhishek	21
S003	Shashank	20
	Rahul	20

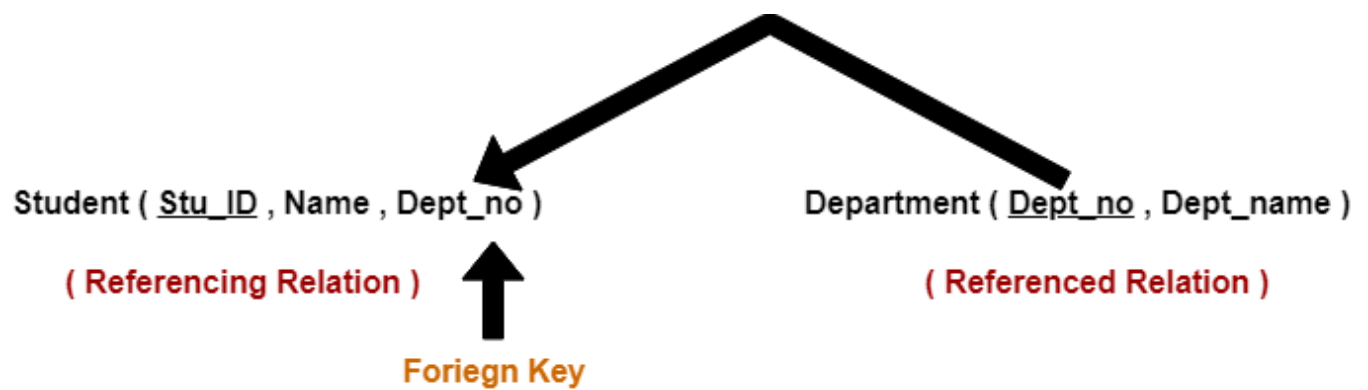
The above relation does not satisfy Entity Integrity Constraint as here the Primary Key contains a NULL value.

5. Referential Integrity Constraint

- This constraint is enforced when a foreign key references the primary key of a relation.
- It specifies that all the values taken by the foreign key must either be available in the relation of the primary key or be null.
- We can not insert a record into a referencing relation if the corresponding record does not exist in the referenced relation.
- We can not delete or update a record of the referenced relation if the corresponding record exists in the referencing relation.

Eg : Consider two relations 'student' and 'Department'

Here relation 'student' references the relation 'Department'



Student

<u>STU_ID</u>	Name	Dept_no
S001	Akshay	D10
S002	Abhishek	D10
S003	Shashank	D11
S004	Rahul	D14

Department

<u>Dept_no</u>	Dept_name
D10	ASET
D11	ALS
D12	ASFL
D13	ASHS

Here,

- The relation 'Student' does not satisfy the referential integrity constraint.
- This is because in relation 'Department', no value of primary key specifies department no. 14.
- Thus, referential integrity constraint is violated.

Constraint and its creation:

1. NOT NULL :

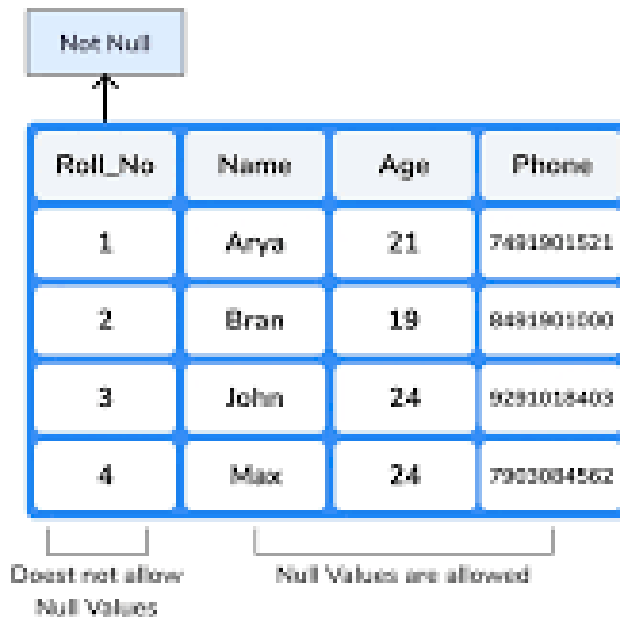
- Values cannot be NULL
- A NULL is not the same as no data, rather, it represents unknown data.

Eg : CREATE TABLE STUDENTS(
 ROLL_NO INT NOT NULL,
 NAME VARCHAR (20) ,
 AGE INT,
 PHONE INT);

- To add NOT NULL constraint to an existing table

ALTER TABLE STUDENTS
 MODIFY AGE INT NOT NULL;

Not Null



2. DEFAULT :

- Set Default Value if not passed
- It provides a default value to a column when the INSERT INTO statement does not provide a specific value.

Eg : CREATE TABLE CUSTOMERS(
 ID_NO INT NOT NULL,
 NAME VARCHAR (20) NOT NULL,
 AGE INT NOT NULL,
 COMPANY CHAR (25),
 COMPANY VARCHAR (18) DEFAULT 'Prepinsta');

- To add DEFAULT constraint to an existing table

ALTER TABLE CUSTOMERS MODIFY SALARY DECIMAL (18, 2) DEFAULT 5000.00;

Default in DBMS

ID_No	Name	Company	Phone
1	Arya	Preplista	7481901521
2	Bran	Preplista	8481901000
3	John	Preplista	9291018403
4	Max	Preplista	7903084562

↓
Row with Default
Value "Preplista"

3. UNIQUE :

- Values cannot match any older values
- It prevents two records from having identical values in a column but accepts NULL values
- It is similar to primary key i.e., used to identify row uniquely but can have NULL values

Eg : CREATE TABLE STUDENTS(
ROLL_NO INT NOT NULL UNIQUE,
NAME VARCHAR (20) ,
AGE INT,
PHONE INT);

- To add UNIQUE constraint to an existing table

ALTER TABLE STUDENTS
MODIFY AGE INT NOT NULL UNIQUE;

- To add UNIQUE constraint to an existing table

ALTER TABLE STUDENTS
ADD CONSTRAINT myUniqueConstraint UNIQUE(AGE, PHONE);

Unique

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

All values are different
Allow duplicate values

4. PRIMARY KEY :

- Used to uniquely identify a row
- A primary key is a field in a table which uniquely identifies each row/record in a database table. Primary keys must contain unique values. A primary key column cannot have NULL values.
- A table can have only one primary key, which may consist of single or multiple fields. When multiple fields are used as a primary key, they are called a composite key.
- If a table has a primary key defined on any field(s), then you cannot have two records having the same value of that field(s).

Eg : CREATE TABLE STUDENTS(
 ROLL_NO INT NOT NULL,
 NAME VARCHAR (20) NOT NULL,
 AGE INT NOT NULL,
 GPA DECIMAL (18, 2),
 PRIMARY KEY (ROLL_NO)
);

- To create PRIMARY KEY constraint on more than one attribute in a table is called Composite

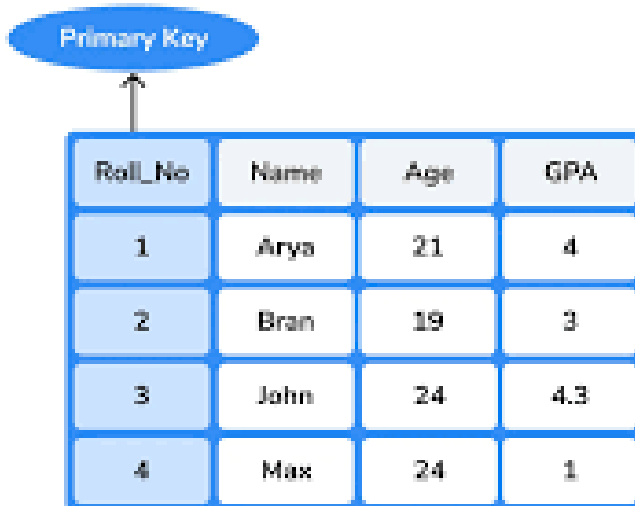
Primary Key

```
CREATE TABLE STUDENTS(
  ROLL_NO INT NOT NULL,
  NAME VARCHAR (20) NOT NULL,
  AGE INT NOT NULL,
  GPA DECIMAL (18, 2),
  PRIMARY KEY (ROLL_NO, NAME));
```

- To add PRIMARY KEY constraint to an existing table

```
ALTER TABLE STUDENTS
  ADD CONSTRAINT PK_ROLLNO PRIMARY KEY (ROLL_NO, NAME);
```

Primary Key



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

5. CHECK :

- Used to validate condition for a new value
- It enables a condition to check the value being entered into a record. If the condition evaluates to false, the record violates the constraint and isn't entered in the table.

```
CREATE TABLE STUDENTS(
  ROLL_NO INT NOT NULL,
  NAME VARCHAR (20) NOT NULL,
  AGE INT CHECK (AGE >= 18),
);
```

- It checks the newly entered value for age column is greater than 18
- To add CHECK constraint to an existing table

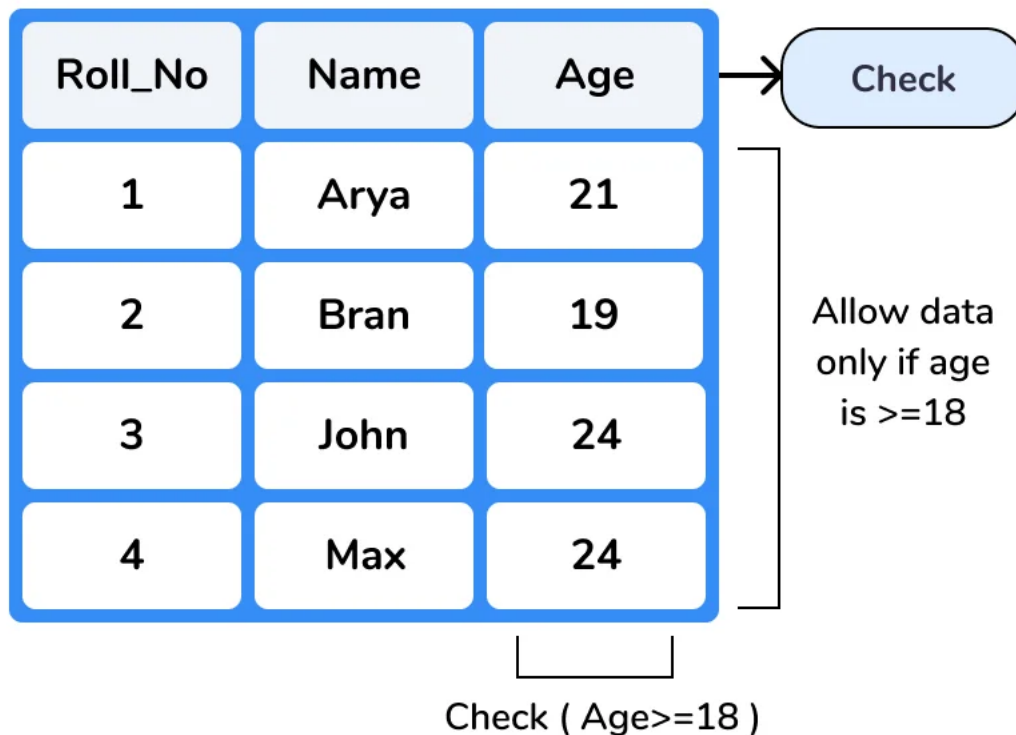
```
ALTER TABLE STUDENTS
  MODIFY AGE INT CHECK (AGE >= 18);
```

OR

```
ALTER TABLE STUDENTS
```

```
ADD CONSTRAINT myCheckConstraint CHECK(AGE >= 18);
```

Check



6. FOREIGN KEY :

- Used to references a row in another table
- Uniquely identifies a row/record in any of the given database table.
- A foreign key is a key used to link two tables together. This is sometimes also called as a referencing key.
- A Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.
- The relationship between 2 tables matches the Primary Key in one of the tables with a Foreign Key in the second table.
- If a table has a primary key defined on any field(s), then you cannot have two records having the same value of that field(s).

Eg: Consider the structure of the following two tables.

STUDENT_DETAILS table

```
CREATE TABLE Student_Details(
  ROLL_NO INT NOT NULL,
  NAME VARCHAR (20) NOT NULL,
  COURSE_ID VARCHAR(10),
  PRIMARY KEY (COURSE_ID)
);
```

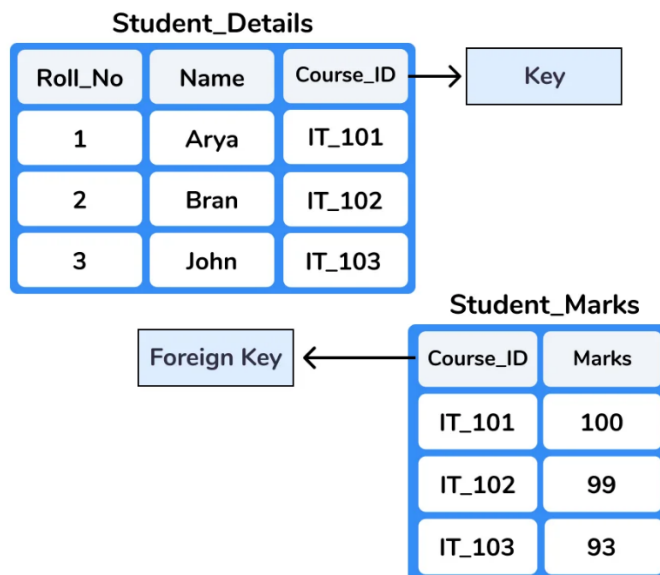
STUDENT_MARKS table

```
CREATE TABLE Student_Marks (
  COURSE_ID INT references Student_Details(COURSE_ID),
  MARKS INT
);
```

If the Student_Marks table has already been created and the foreign key has not yet been set, the use the syntax for specifying a foreign key by altering a table.

```
ALTER TABLE Student_Marks
  ADD FOREIGN KEY (COURSE_ID) REFERENCES Student_Details (COURSE_ID);
```

Foreign Key



FUNCTIONS

- A function in Oracle is a subprogram that is used to perform an action such as complex calculations and returns the result of the action as a value
- SQL is used to handle data in databases. It provides various in-built functions and commands to access and manage databases according to our requirements.
- Functions are methods used to perform data operations. SQL has many in-built functions used to perform string concatenations, mathematical calculations etc.
- SQL function is used to show the functionality of the database records, which we create or

delete, according to our condition. It is a collection of SQL statements, which allows the input parameters, depending on which, it simply takes some types of inputs and displays a result accordingly, nothing more or less than that. The function is executed each time, whenever it was called.

- SQL functions are categorized into the following two categories:

- i. Aggregate Functions

The Aggregate Functions in SQL perform calculations on a group of values and then return a single value. Following are a few of the most commonly used Aggregate Functions:

S.No	Function	Description
1	SUM()	Used to return the sum of a group of values.
2	COUNT()	Returns the number of rows either based on a condition, or without a condition.
3	AVG()	Used to calculate the average value of a numeric column.
4	MAX()	Returns a maximum value of a column.
5	MIN()	This function returns the minimum value of a column.

- ii. Scalar Functions

The Scalar Functions in SQL are used to return a single value from the given input value. Following are a few of the most commonly used Scalar Functions:

S.No	Function	Description
1	LCASE()	Used to convert string column values to lowercase
2	UCASE()	This function is used to convert a string column values to Uppercase.
3	LEN()	Returns the length of the text values in the column.
4	MID()	Extracts substrings in SQL from column values having String data type.
5	ROUND()	Rounds off a numeric value to the nearest integer.
6	NOW()	This function is used to return the current system date and time.
7	FORMAT()	Used to format how a field must be displayed.

To call a function in Oracle:

Following is the syntax to call a function in Oracle:

SELECT <FNAME>(VALUES) FROM DUAL;

Dual in Oracle:

1. It is a pre-define table in oracle.
2. It is having single column & single row
3. It is called a dummy table in oracle.
4. It is used for testing functions (pre-define & user define) functionalities.

Structure of Dual table:

DESC DUAL;


```
SQL> Desc dual;
```

Name	Null?	Type
DUMMY		VARCHAR2(1)

Data of DUAL table:

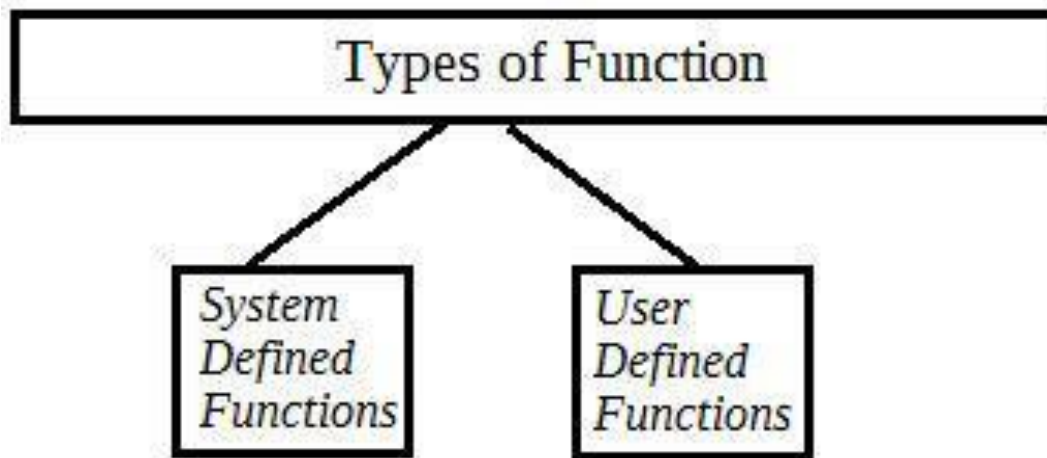
```
SELECT * FROM DUAL;
```

```
SQL> SELECT * FROM DUAL;
```

```
D
-
X
```

TYPES OF FUNCTIONS

1. System Defined Functions - These type of functions, SQL Server already contains, which will be difficult to implement without the System Defined Functions. This function is easily mixed with the complicated and immediately required operations.
2. User Defined Functions - This function returns a value, based on the user-defined condition. User Defined Functions are used to create an easy statement. It generates at the time of execution, which is based on these parts:
 - Scalar-valued function: A user-defined function can return a scalar value, we use this statement in all the database.
 - Inline function: When a user-defined function holds the Select statement, that is updatable or not, the tabular result returned by the function is also updatable. These functions are called inline functions. We can use these functions in the 'Form clause of another query.
 - Table-valued function: When a user-defined function holds the Select statement, that is updatable or not, this Table-valued function returned is not updatable. Table-valued function shows a table and we can use these functions in the 'Form clause of another query.



Advantage of Functions

The SQL function have some advantages like:

- We can use these functions with where, having (clause).
- They can be used in a 'Select statement'.
- These are based on the function oriented.
- These functions are easy to use.

SYSTEM DEFINED FUNCTIONS:

- Also called as Built-in Functions / Pre-defined functions
- It is the functions which are already defined by the oracle system and ready to be used by the user or developer
- It is an expression in which an SQL keyword or special operator executes some operation.
- It uses keywords or special built-in operators.
- They are SQL92 Identifiers and are case-insensitive.
- These are again classified into two categories.
 - Single row functions (scalar functions) - These functions are returning a single row (or) a single value. Following are the examples.
 - Numeric Functions

Function	Description	Syntax	Example	O/P
ABS()	used to convert (-VE) value into (+VE) value	ABS(number)	SELECT ABS(-12) FROM DUAL;	12
CEIL()	used to return a value that is greater than or equal to the given value	CEIL(NUMBER)	SELECT CEIL(9.3) FROM DUAL;	10

FLOOR()	used to return the largest integer equal to or less than n	FLOOR(NUMBER)	SELECT FLOOR(9.8) FROM DUAL;	9
MOD()	used to return the remainder value	MOD(m, n)	SELECT MOD(10,2) FROM DUAL;	0
			SELECT MOD(15,4) FROM DUAL;	3
POWER()	used to return the power of a given expression	POWER(m, n)	SELECT POWER(2, 3) FROM DUAL;	8
ROUND()	returns the nearest value of the given expression	ROUND(NUMBER,[DECIMAL PLACES])	SELECT ROUND(15.253, 1) FROM DUAL;	15.3
			SELECT ROUND(15.253,-1) FROM DUAL;	20
TRUNC()	used to return n1 truncated to n2 decimal places	TRUNC (NUMBER, DECIMAL PLACES)	SELECT TRUNC(5.50) FROM DUAL;	5
			SELECT TRUNC(32.456,2) FROM DUAL;	32.45
			SELECT TRUNC(32.456,-1) FROM DUAL;	30

▪ String Functions

Function	Description	Syntax	Example	O/P
LENGTH()	used to return the length of a given string	LENGTH(String)	SELECT LENGTH('HELLO') FROM DUAL;	5
LOWER()	used to return a specified character expression in lowercase letters	LOWER(Character Expression)	SELECT LOWER('HELLO') FROM DUAL;	hello
UPPER()	used to return a specified character expression in uppercase letters	UPPER(Character Expression)	SELECT UPPER('hello') FROM DUAL;	HELLO
INITCAP()	used to set the first letter of each word in uppercase, and rest all other letters in lowercase	INITCAP(STRING)	SELECT INITCAP('hello') FROM DUAL;	Hello
LTRIM()	used to remove the unwanted spaces (or) unwanted characters from the left side of the given string	LTRIM(STRING1 [,STRING2])	SELECT LTRIM(' ANU') TRIML FROM DUAL;	ANU
			SELECT LTRIM('XXXXXXANU','X') TRIML FROM DUAL;	ANU

RTRIM()	used to remove the unwanted spaces (or) unwanted characters from the right side of the given string	RTRIM(STRING1 [,STRING2])	SELECT RTRIM(' ANU ') TRIMR FROM DUAL;	AN U
			SELECT RTRIM('ANURAGXXXXX','X') TRIMR FROM DUAL;	AN U
			SELECT RTRIM('ANU123','123') TRIMR FROM DUAL;	AN U
CONCAT()	used to return the result (a string) of concatenating two string values	CONCAT(char1, char2)	SELECT CONCAT('Good', 'Morning') ConcatString FROM DUAL;	Go od Mo rnin g
REPLACE()	used to return a string with every occurrence of search_string replaced with replacement_string	REPLACE (String, Search_String [, Replacement_String])	SELECT REPLACE('JACK and JUE','J','BL') "New String" FROM DUAL;	BL AC K and BL UE
TRANSLATE()	used to return a string with all occurrences of each character specified in another string as 2nd argument replaced by its corresponding character specified in the 3rd argument	TRANSLATE(expression, from_string, to_string)	SELECT TRANSLATE('C D T', ' ', ',') AS "New String" FROM DUAL;	C,D ,T
SUBSTR()	Used to return the specified number (substring_length) of characters from a particular position of a given string	SUBSTR (char, position [, substring_length])	SELECT SUBSTR('HELLO',2,3) "New String" FROM DUAL;	EL L
INSTR()	Used to search string for a substring	INSTR(string , substring [, position [, occurrence]])	SELECT INSTR('HELLO WELCOME','O') "New String" FROM DUAL;	5
			SELECT INSTR('HELLO WELCOME','Z') "New String" FROM DUAL;	0

▪ Date Functions

- Date functions can be defined as a set of functions that operate on date and time and allows the developer to retrieve the current date and time in a particular time zone or extract only the date/month/year or more complex actions like extracting the last day of the month/next day/session time zone,

etc.

- It also consists of functions that can be used to convert a Date value to a character string or convert a date that is in a character string to a Date value.

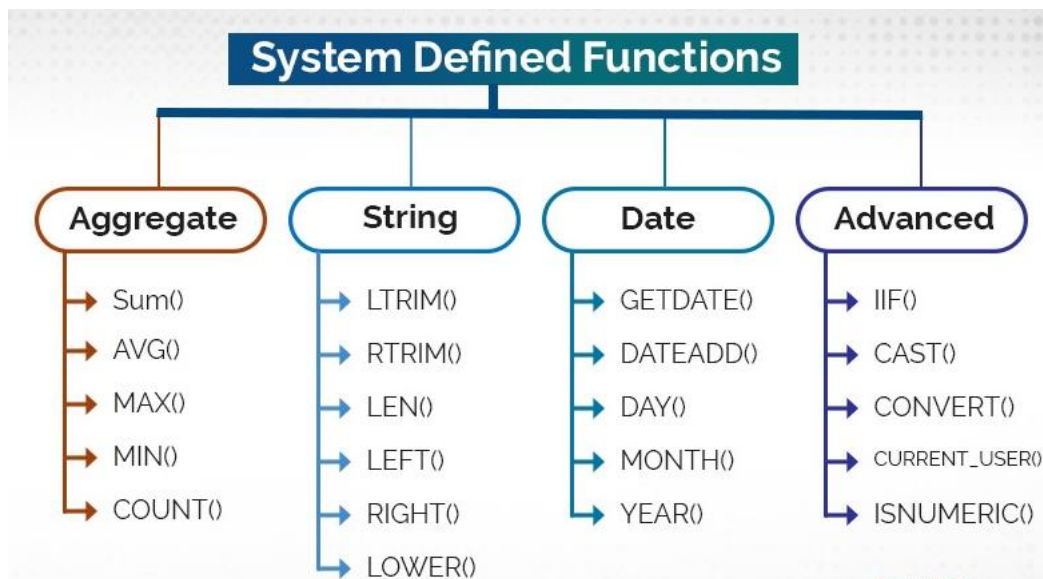
Function	Description	Syntax	Example	O/P
SYSDATE	used to return the current date and time set for the operating system on which the database resides	SYSDATE	SELECT SYSDATE FROM DUAL;	20-AUG-22
			SELECT SYSDATE+10 FROM DUAL;	30-AUG-22
			SELECT TO_CHAR(SYSDATE, 'MM-DD-YYYY HH:MI:SS') "NOW" FROM DUAL;	20-AUG-22 06:57:08
ADD_MONTHS()	used to return the date with a given number of months added (date plus integer months)	ADD_MONTHS(date, integer)	SELECT ADD_MONTHS(SYSDATE, 2) FROM DUAL;	20-OCT-22
LAST_DAY()	used to return the last day of the month that contains a date	LAST_DAY(date)	SELECT LAST_DAY(SYSDATE) FROM DUAL;	31-AUG-22
			SELECT LAST_DAY(SYSDATE) - SYSDATE "Days Left" FROM DUAL;	11
NEXT_DAY()	used to return the date of the first weekday that is later than the date	NEXT_DAY(DATE, '<DAY _NAME>')	SELECT NEXT_DAY(SYSDATE, ' MONDAY') FROM DUAL;	22-AUG-22
MONTHS_BETWEEN()	used to get the number of months between two dates (date1, date2).	MONTHS_BETWEEN(DATE1, DATE2)	SELECT MONTHS_BETWEEN('05-JAN-81', '05-JAN-80') FROM DUAL;	12

■ Conversion Functions

The Oracle Server uses data of one type where it expects data of a different data type. This can happen when the Server automatically converts the data to the expected data type. This data type conversion can be done implicitly by the Server, or explicitly by the user. As a user, we can use the following two functions to convert explicitly.

Function	Description	Syntax	Example	O/P
----------	-------------	--------	---------	-----

TO_CHAR() R()	used to convert a DateTime or interval value of DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, or TIMESTAMP WITH LOCAL TIME ZONE data type to a value of VARCHAR2 datatype in a specified format	TO_CHAR({ Datetime Interval } [, fmt [, 'nlsparam']])	SELECT TO_CHAR(SYSDATE,'Y YYY YY YEAR CC AD') FROM DUAL;	2022 22 TWENTY TWENTY- TWO 21 AD
TO_DATE() E()	used to convert char type to oracle date format type	TO_DATE(STRING[,FROM AT])	INSERT INTO EMP VALUES (7499, 'ALLEN', 'SALESMAN', TO_DATE('20-FEB- 1981', 'DD-MON- YYYY')); SELECT * FROM EMP WHERE TO_CHAR(HIREDATE,' YYYY')=1982;	Row inserted Displays records with hire date year 1982

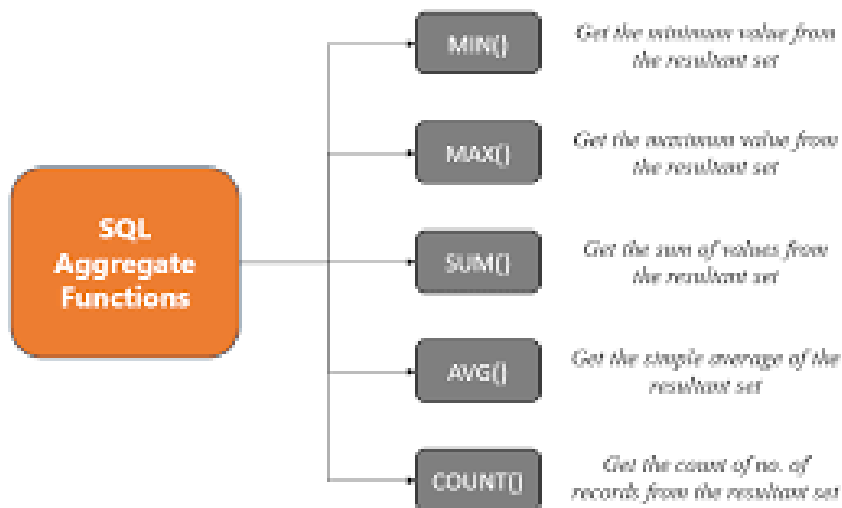


- Multiple row functions (grouping functions)
 - It is also called as Aggregate Functions

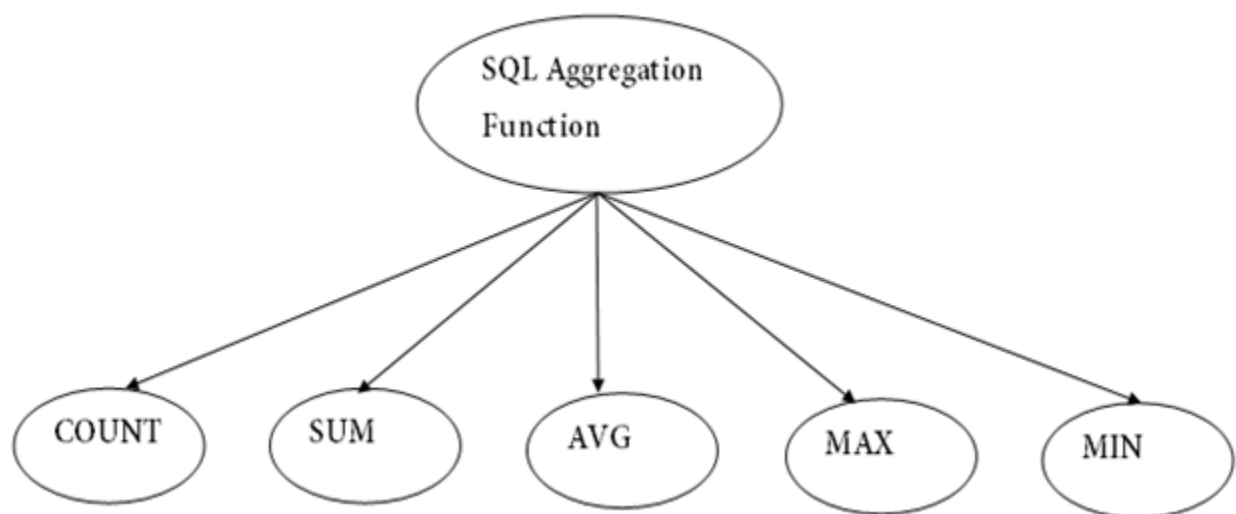
AGGREGATE FUNCTIONS:

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

- Aggregate functions are often used with the GROUP BY and HAVING clauses of the SELECT statement.
- These functions ignore the NULL values except the count function.
- It is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.
- It is also used to summarize the data.



Types of Aggregate Functions:



Now let us understand each Aggregate function with a example:

Id	Name	Salary
1	A	80
2	B	40
3	C	60
4	D	70
5	E	60
6	F	Null

i. COUNT:

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

Eg : Count(*): Returns total number of records .i.e 6.

Count(salary): Return number of Non Null values over the column salary. i.e 5.

Count(Distinct Salary): Return number of distinct Non Null values over the column salary .i.e 4

ii. SUM:

- It is used to calculate the sum of all selected columns. It works on numeric fields only.

Eg : sum(salary): Sum all Non Null values of Column salary i.e., 310

sum(Distinct salary): Sum of all distinct Non-Null values i.e., 250.

iii. AVG:

- It is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Eg : Avg(salary) = Sum(salary) / count(salary) = 310/5

Avg(Distinct salary) = sum(Distinct salary) / Count(Distinct Salary) = 250/4

iv. MAX:

- It is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

Eg : Max(salary): Maximum value in the salary column except NULL i.e., 80.

v. MIN:

- It is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

Eg : Min(salary): Minimum value in the salary column except NULL i.e., 40.

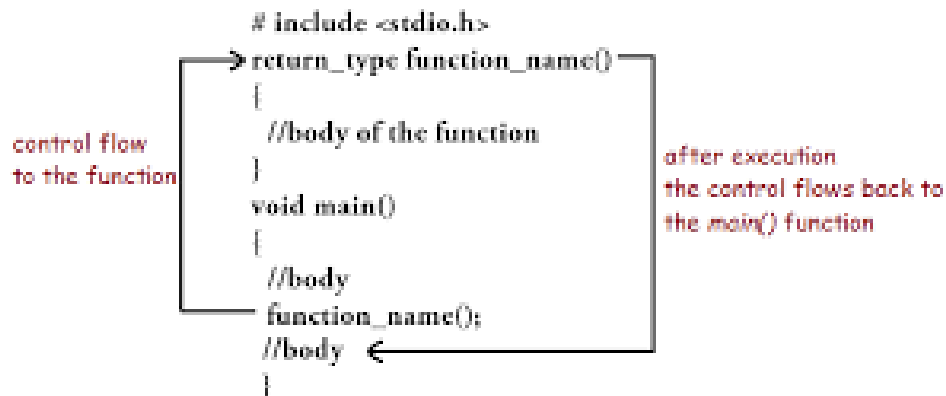
USER DEFINED FUNCTIONS:

- It is the Function is defined by the user or developer
- A PL/SQL function is a reusable program unit stored as a schema object in the Oracle Database

- Its structure is as follows,

```
CREATE [OR REPLACE] FUNCTION function_name (parameter_list)
                                RETURN return_type
IS
    [declarative section]
BEGIN
    [executable section]
    [EXCEPTION]
    [exception-handling section]
END;
```

The flow of executing the function is represented as follows,



A function consists of:

- Function Header:
 - has the function name
 - a RETURN clause that specifies the datatype of the returned value
 - Each parameter of the function can be either in the IN, OUT, or INOUT mode
- Function Body : It has three sections
 - Declarative Section
 - section is between the IS and BEGIN keywords
 - It is where you declare variables, constants, cursors, and user-defined types.
 - Executable Section
 - section is between the BEGIN and END keywords
 - It is where you place the executable statements
 - have at least one RETURN statement in the executable statement.
 - Exception-Handling Section
 - Contains exception handler code.

A Sample Code for User-defined Function:

```

CREATE OR REPLACE FUNCTION get_total_sales(in_year PLS_INTEGER)
RETURN NUMBER
IS
    l_total_sales NUMBER := 0;
BEGIN
    -- get total sales
    SELECT SUM(unit_price * quantity)
    INTO l_total_sales
    FROM order_items
    INNER JOIN orders USING (order_id)
    WHERE status = 'Shipped'
    GROUP BY EXTRACT(YEAR FROM order_date)
    HAVING EXTRACT(YEAR FROM order_date) = in_year;
    -- return the total sales
    RETURN l_total_sales;
END;
```

The user-defined function calling is done by:

1. In an assignment statement


```

      DECLARE
          l_sales_2017 NUMBER := 0;
      BEGIN
          l_sales_2017 := get_total_sales (2017);
          DBMS_OUTPUT.PUT_LINE('Sales 2017: ' || l_sales_2017);
      END;
```
2. In a Boolean Expression


```

      BEGIN
          IF get_total_sales (2017) > 10000000 THEN
              DBMS_OUTPUT.PUT_LINE('Sales 2017 is above target');
          END IF;
      END;
```
3. In an SQL Statement


```

      SELECT  get_total_sales(2017)
      FROM    dual;
```

To remove a User-defined function,

```
DROP FUNCTION function_name;
```

Eg:

```
DROP FUNCTION get_total_sales;
```

PL/SQL STORED PROCEDURE:

- A PL/SQL procedure is a reusable unit that encapsulates specific business logic of the application
- It is a prepared SQL code that you can save, so the code can be reused over and over again.
- It is a named block stored as a schema object in the Oracle Database.
- Its basic syntax includes,

```
CREATE [OR REPLACE ] PROCEDURE procedure_name (parameter_list)
```

```
IS
```

```
    [declaration statements]
```

```
BEGIN
```

```
    [execution statements]
```

```
EXCEPTION
```

```
    [exception handler]
```

```
END [procedure_name ];
```

The most important part is parameters. Parameters are used to pass values to the Procedure. There are 3 different types of parameters, they are as follows:

1. IN: This is the Default Parameter for the procedure. It always receives the values from calling program.
2. OUT: This parameter always sends the values to the calling program.
3. IN OUT: This parameter performs both the operations. It Receives value from as well as sends the values to the calling program.

The sample procedure includes,

```
CREATE OR REPLACE PROCEDURE print_contact(in_customer_id NUMBER )
```

```
IS
```

```
    r_contact contacts%ROWTYPE;
```

```
BEGIN
```

```
    -- get contact based on customer id
```

```
    SELECT *
```

```
    INTO r_contact
```

```
    FROM contacts
```

```
    WHERE customer_id = p_customer_id;
```

```
    -- print out contact's information
```

```
    dbms_output.put_line( r_contact.first_name || ' ' || r_contact.last_name || '<' || r_contact.email || '>' );
```

```
EXCEPTION
```

```
    WHEN OTHERS THEN
```

```
        dbms_output.put_line( SQLERRM );
```

END;

To execute a PL/SQL Procedure,

EXECUTE procedure_name(arguments); OR
EXEC procedure_name(arguments);

Eg:

EXEC print_contact(100);

To remove a PL/SQL Procedure,

DROP PROCEDURE procedure_name;

Eg:

DROP PROCEDURE print_contact;

STORED PROCEDURES(SP) Vs FUNCTIONS (USER DEFINED FUNCTION / UDF):

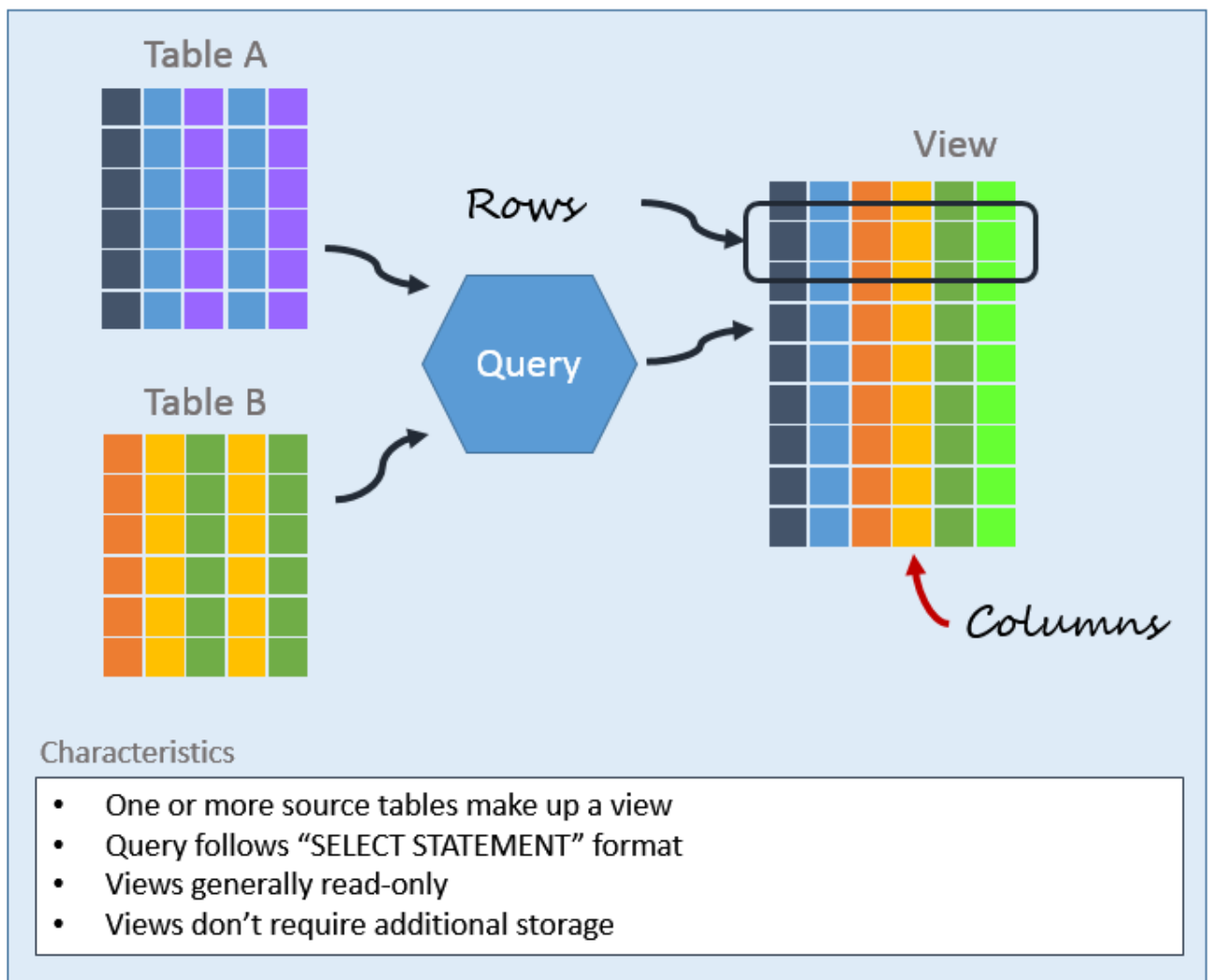
S.NO	SP	UDF
1	It may or may not return a value	It must return a value
2	It can have input/output parameters	It has only input parameters
3	Can call UDF from SP	Cannot call SP from UDF
4	SP cannot be used in SQL statements like SELECT, INSERT, UPDATE, DELETE	UDF can be used in SQL statements like SELECT, INSERT, UPDATE, DELETE
5	SP cannot be used anywhere in the WHERE/HAVING/SELECT part of SQL queries	UDF can be used anywhere in the WHERE/HAVING/SELECT part of SQL queries
6	Execution plan can be re-used in SP	Execution plan can be compiled every time in UDF
7	Can use transactions in SP	Cannot use transactions in UDF
8	SP can have both table variables and temporary tables	UDF can have only table variables as it does not permit the use of temporary tables
9	<ul style="list-style-type: none"> Improved security measures Client/server traffic is reduced. Programming abstraction and efficient code reuse Pre-compiled Execution 	<ul style="list-style-type: none"> Enables faster execution Modular Programming Can reduce network traffic (data)

VIEWS

- view is a virtual table that does not physically exist.
- It is stored in Oracle data dictionary and do not store any data. It can be executed when called.
- A view is created by a query joining one or more tables.
- Views are used in the following cases:
 - Simplifying data retrieval
 - Build a complex query, test it carefully, and encapsulate the query in a view

- Then, you can access the data of the underlying tables through the view instead of rewriting the whole query again and again.
- Maintaining logical data independence
 - expose the data from underlying tables to the external applications via views.
 - Whenever the structures of the base tables change, you just need to update the view
 - The interface between the database and the external applications remains intact
- Implementing data security
 - Views allow you to implement an additional security layer
 - They help you hide certain columns and rows from the underlying tables and expose only needed data to the appropriate users.
 - Oracle provides you the with GRANT and REVOKE commands on views so that you can specify which actions a user can perform against the view
-

Anatomy of a View



To create a new view in a database,

```
CREATE [OR REPLACE] VIEW view_name [(column_aliases)] AS
```

defining-query

[WITH READ ONLY]

[WITH CHECK OPTION]

A) Creating View

Eg:

```
CREATE VIEW employee_yos AS
```

```
SELECT employee_id, first_name || ' ' || last_name full_name,
```

```
FLOOR( months_between( CURRENT_DATE, hire_date )/ 12 ) yos FROM employees;
```

B) Creating read-only view

Eg:

```
CREATE OR REPLACE VIEW customer_credits(customer_id,name,credit) AS
```

```
SELECT customer_id, name, credit_limit FROM
```

```
customers WITH READ ONLY;
```

C) Creating join view

Eg:

```
SELECT product_name, EXTRACT(YEAR FROM order_date) YEAR, SUM( quantity * unit_price )  
amount
```

```
FROM orders INNER JOIN order_items USING(order_id)
```

```
INNER JOIN products USING(product_id)
```

```
WHERE status = 'Pending'
```

```
GROUP BY EXTRACT(YEAR FROM order_date), product_name;
```

To remove a view,

```
DROP VIEW view_name;
```

Eg:

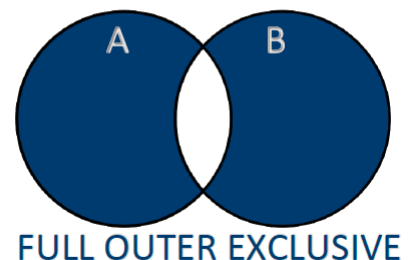
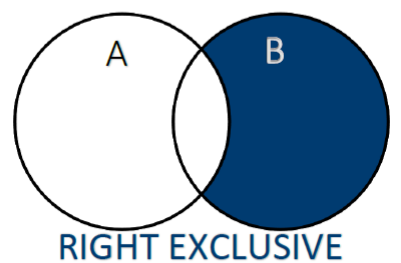
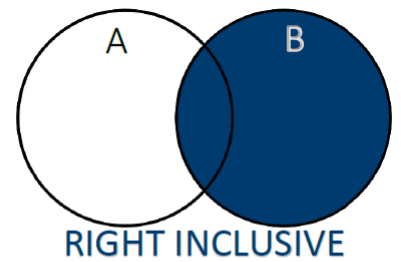
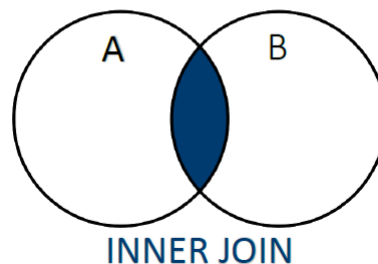
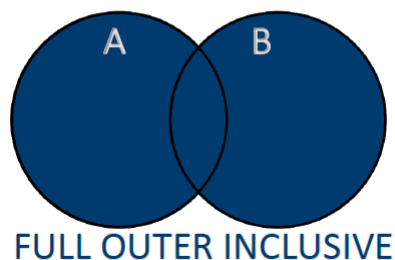
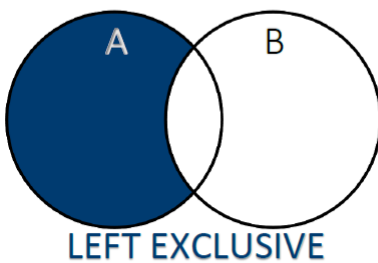
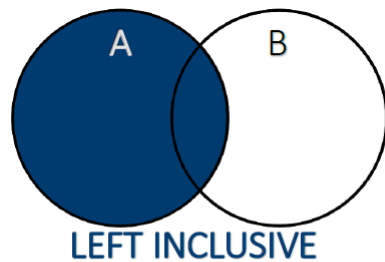
```
DROP VIEW employee_yos;
```

JOINS

- A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied. Three types of joins are available namely inner join, outer join, semi

join.

- Oracle join is used to combine columns from two or more tables based on values of the related columns. The related columns are typically the primary key column(s) of the first table and foreign key column(s) of the second table.
- Oracle supports inner join, left join, right join, full outer join and cross join.
- Note that you can join a table to itself to query hierarchical data using an inner join, left join, or right join. This kind of join is known as self-join
- The VIEW continues to exist even after one of the tables (that the Oracle VIEW is based on) is dropped from the database. However, if you try to query the Oracle VIEW after the table has been dropped, you will receive a message indicating that the Oracle VIEW has errors.



SQL JOINS	
LEFT INCLUSIVE SELECT [Select List] FROM TableA A LEFT OUTER JOIN TableB B ON A.Key = B.Key	RIGHT INCLUSIVE SELECT [Select List] FROM TableA A RIGHT OUTER JOIN TableB B ON A.Key = B.Key
LEFT EXCLUSIVE SELECT [Select List] FROM TableA A LEFT OUTER JOIN TableB B ON A.Key = B.Key WHERE B.Key IS NULL	RIGHT EXCLUSIVE SELECT [Select List] FROM TableA A LEFT OUTER JOIN TableB B ON A.Key = B.Key WHERE A.Key IS NULL
FULL OUTER INCLUSIVE SELECT [Select List] FROM TableA A FULL OUTER JOIN TableB B ON A.Key = B.Key	FULL OUTER EXCLUSIVE SELECT [Select List] FROM TableA A FULL OUTER JOIN TableB B ON A.Key = B.Key WHERE A.Key IS NULL OR B.Key IS NULL
INNER JOIN SELECT [Select List] FROM TableA A INNER JOIN TableB B ON A.Key = B.Key	

Inner Join :

An inner join includes only those tuples with matching attributes and the rest are discarded in the resulting relation. Theta Join, Equijoin, and Natural Join are called inner joins.

Theta Join (\bowtie)

Theta join combines tuples from different relations provided they satisfy the theta condition. The join condition is denoted by the symbol \bowtie . Theta join can use all kinds of comparison operators. The general form is

$R1 \bowtie_{\theta} R2$

Example

Relations: Employee, Project

Query:

Employee $\bowtie_{\text{Employee.Eid=Project.Empid}}$ Project

Result

<u>Eid</u>	Name	Designation	Salary	Phno	Empid	Pid	Plocation
125	Kumar	Developer	25000	9489562312	125	P123	Coimbatore
568	Sachin	HR	33000	8561234568	568	P225	Bangalore
222	Dhoni	Accountant	39650	8956235689	222	P556	Delhi

Equi Join(\bowtie_c)

Equijoin is a special case of join where only equality condition holds between a pair of attributes. As values of two attributes will be equal in result of equijoin, only one attribute will appear in result. The general form is

$R1 \bowtie_c R2$

Example

Relations: Employee, Project

Query:

Employee $\bowtie_{\text{Employee.Eid=Project.Empid}}$ Project

Result

<u>Eid</u>	Name	Designation	Salary	Phno	Empid	Pid	Plocation
125	Kumar	Developer	25000	9489562312	125	P123	Coimbatore
568	Sachin	HR	33000	8561234568	568	P225	Bangalore
222	Dhoni	Accountant	39650	8956235689	222	P556	Delhi

Natural Join (⋈)

It is used to combine related tuples from two relations into single based on equality condition. There is noneed to specify equality condition explicitly

Example

Relation: Employee, Project

Query:

employee ⋈ project

Result:

<u>Eid</u>	Name	Designation	Salary	Phno	Pid	Plocation
125	Kumar	Developer	25000	9489562312	P123	Coimbatore
568	Sachin	HR	33000	8561234568	P225	Bangalore
222	Dhoni	Accountant	39650	8956235689	P556	Delhi

Semi Join (⋈)(a)

A join where the result only contains the columns from one of the joined relations. A semi join between two tables returns rows from the first table where one or more matches are found in the second table.

Example

Relation: Employee, Project

Query:

employee ⋈ project

Result:

<u>Eid</u>	Name	Designation	Salary	Phno
125	Kumar	Developer	25000	9489562312
568	Sachin	HR	33000	8561234568

222	Dhoni	Accountant	39650	8956235689
-----	-------	------------	-------	------------

Outer Join

Outer joins include all the tuples from the participating relations in the resulting relation. There are three kinds of outer joins – left outer join, right outer join, and full outer join.

Left Outer Join (⋈)

All the tuples from the Left relation, R, are included in the resulting relation. If there are tuples in R without any matching tuple in the Right relation S, then the S-attributes of the resulting relation are made NULL.

Example:

Customer Relation:

customer_id	first_name	last_name	email	address	city	state	zipcode
1	George	Washington	gWASHINGTON@usa.gov	3200 Mt Vernon Hwy	Mount Vernon	VA	22121

2	John	Adams	jadams@usa.gov	1250 Hancock St	Quincy	MA	02169
3	Thomas	Jefferson	tjefferson@usa.gov	931 Thomas Jefferson Pkwy	Charlottesville	VA	22902
4	James	Madison	jmadison@usa.gov	11350 Constitution Hwy	Orange	VA	22960
5	James	Monroe	jmonroe@usa.gov	2050 James Monroe Parkway	Charlottesville	VA	22902

Orders Relations

order_id	order_date	amount	customer_id
1	07/04/1776	\$234.56	1
2	03/14/1760	\$78.50	3
3	05/23/1784	\$124.00	2
4	09/03/1790	\$65.50	3
Result	07/21/1795	\$25.50	10
6	11/27/1787	\$14.40	9

Query:

Customer ⋈ **Orders**

first_name	last_name	order_date	order_amount
George	Washington	07/04/1776	\$234.56
John	Adams	05/23/1784	\$124.00
Thomas	Jefferson	03/14/1760	\$78.50
Thomas	Jefferson	09/03/1790	\$65.50
James	Madison	NULL	NULL
James	Monroe	NULL	NULL

Right Outer Join(⋈)

All the tuples from the Right relation, S, are included in the resulting relation. If there are tuples in S without any matching tuple in R, then the R-attributes of resulting relation are made NULL.

Example

Relation: Customer, Orders

Query:

Customer ⋈ Orders

Result:

first_name	last_name	order_date	order_amount
George	Washington	07/04/1776	\$234.56
Thomas	Jefferson	03/14/1760	\$78.50
John	Adams	05/23/1784	\$124.00
Thomas	Jefferson	09/03/1790	\$65.50
NULL	NULL	07/21/1795	\$25.50
NULL	NULL	11/27/1787	\$14.40

Full Outer Join (⋈)

All the tuples from both participating relations are included in the resulting relation. If there are nomatching tuples for both relations, their respective unmatched attributes are made NULL.

Relation:

Customer,

OrdersQuery:

Customer ⋈

Orders

Result

first_name	last_name	order_date	order_amount
George	Washington	07/04/1776	\$234.56
Thomas	Jefferson	03/14/1760	\$78.50
John	Adams	05/23/1784	\$124.00
Thomas	Jefferson	09/03/1790	\$65.50
NULL	NULL	07/21/1795	\$25.50
NULL	NULL	11/27/1787	\$14.40
James	Madison	NULL	NULL
James	Monroe	NULL	NULL

