OPERATORS:-

Arithmetic Operators:-

Oracle allows arithmetic operators to be used while viewing records from a table or while performing Data Manipulation operations such as select and update.

Arithmetic Operators like

(1) + Addition (4) / Division
(2) – Subtraction (5) ( ) Enclosed Operations
(3) * Multiplication

- Multiplication and Division take priority over Addition and Subtraction.
- Operators of the same priority are evaluated from left to right.
- Parenthesis can be used to force prioritized evaluation

Example:-

Select sal, **sal*.10** "DA",**sal*.50** "hra",**sal+(sal*.10)+(sal*.50)** "gs" from emp;

Relational Operator:

Oracle allows relational operators to be used while comparison of values or expression.

Relational Operators like

1. Equal (=) Operator
2. Not Equal (!= or <>) Operator
3. Greater Than (>) Operator
4. Less Than (<) Operator
5. Greater Than or Equal To (>=) Operator
6. Less Than or Equal To (<=) Operator

Example:-

Select * from emp where sal > 50000;

Logical Operators:-

Logical operators that can be used in SQL sentences are:

The AND Operator:-

- The AND operator allows creating an SQL statement based on two or more conditions being met.
- The AND operator requires that each condition must be met for the record to be included in the result set.

- The oracle engine will process all rows in a table and display the result only when **all** of the conditions specified using the **AND** operator is satisfied.

Example1:	**Example2:**
Select * from emp where sal>=500 **and** sal<=5000;	Select * from emp where sal>=500 **and** sal<=5000 **and** gender='m';

The OR Operator:-

- The **OR** condition allows creating an SQL statement where records are returned when any one one of the conditions are met.
- The **OR** condition requires that any of the condition must be met for the record to be included in the result set.
- The oracle engine will process all rows in a table and display the result only when **any of** the conditions specified using the **OR** operator is satisfied.

Example1:	**Example2:**
Select * from emp where job='manager' **or** manager='clerk'; i.e. Display all the information of employee whose job either 'manager' or 'clerk'	Select * from emp where city='delhi' **or** city='bombay' **or** city='baroda'; i.e. Display all the information of employee whose city like 'delhi', 'bombay' or 'baroda'.

The NOT Operator:-

- The oracle engine will process all rows in a table and display only those records that **do not** satisfied the condition specified.

Example1:	**Example2:**
Select * from emp where **NOT**(desig='maanager' or design='clerk') i.e. Display all the information of employee whose designation **neither** 'manager' **nor** 'clerk'.	Select *from emp where comm is **not** null;

The IN and NOT IN Predicate:-

IN Predicate:-

- The arithmetic operator (=) compares a single value to another single value.
- In case a value needs to be compared to a list of values then the **IN** predicate is used.
- The **IN** predicate helps **reduce** the need to use multiple **OR** condition.
- It is also known as membership condition.
- The IN condition can be used with any data type. If character or date are used in the list that must be enclosed within **single quotation mark.**

Syntax:-	**Examples:-**
Select * from <table_name> where <column_name> **IN (value1,value2,…);**	Sql> Select * from emp where **deptno in (10,20,30)**; Sql> Select * from emp where **job in ('clerk','peon')**;

NOT IN Predicate:-

- The **NOT IN** predicate is the opposite of the **IN** predicate.
- This will select all the rows where values do not match the values in the list.

Syntax:-	**Examples:-**
Select * from <table_name> where <column_name> **NOT IN (value1,value2,…);**	-> Select * from emp where **deptno not in (10,20,30)**; -> Select * from emp where **job not in ('clerk','peon')**;

Range Searching and Pattern Matching:-

➢ Range Searching:- (BETWEEN and NOT BETWEEN)

- In order to select data that is within a range of values, the **BETWEEN** operator is used.
- The **BETWEEN** operator allows the selection of rows that contain values within a specified lower and upper limit.
- The range coded after the word **BETWEEN is inclusive.**
- The lower value must be coded first. The two values between the range must be linked with the keyword **AND.**
- The **BETWEEN** operator can be used with both character and numeric data types. The data types cannot be mixed. i.e. the lower value of range of values from a character column and the other from a numeric column.

Syntax 1:-	**Example:-**
Select <ColumnName1>,<columnName2>,…. from <table_name> where <column_name> **BETWEEN <lower_limit> AND <upper_limit>;** **Syntax 2:** Select <ColumnName1>, <ColumnName2> … from <table_name> where <column_name> **NOT BETWEEN <lower_limit> AND <upper_limit>;**	Select *from emp where **sal between 1000 and 2000;** This Query is same as the Select * from emp where sal>=1000 and sal<=2000; **Example:-** Select *from emp where **sal not between 1000 and 2000;**

➢ PATTERN MATCHING:- (LIKE)

- The **LIKE** predicate allows comparison of one string value with another string value, which is not identical.
- This is achieved by using **wildcard** characters. Two wildcard characters that are available are:

  For character data types:-
  - **%** allows to match any string of any length.
  - **_ (underscore)** allows to match on a single character.

Example1:-	**Example2:-**
Select * from emp where ename **like 'e%';** i.e. display all the information of employee whose employee name start with 'e'.	Select * from emp where ename **like '_e%';** i.e. display all the information of employee whose second character of empname is 'e'.

Null Values Concept:

Often there may be record in a table that do not have values for every field.

- **Principles of NULL Values:-**
  - Setting a NULL value is appropriate when the actual value is unknown, or when a value would not be meaningful.
  - A NULL value is **not equivalent** to a value of **zero** if the data type is **number** and is not equivalent to **spaces** if the data type is **character**.
  - A NULL value will evaluate to NULL in any expression. (e.g. NULL multiplied by 5 is NULL).

NULL value can be inserted into columns of **any data type**.

Tab Table

Finding out the tables Created by a User:-

The command shown below is used to determine the tables to which user has access.

Syntax:-	**Example:-**
Select * from **TAB:**	Select * from **TAB**; Output: Tab Name Tab Type emp Table dept Table

DUAL TABLE:-

- The Dual is owned by SysUser and can be access by all user.
- It contain one column **Dummy** and one row with the value **X**.
- The dual table is useful when you want to return a value once only, for instance , the value of constant or expression. i.e not derived from a table with used data.
- The dual table is generally used in select statement.

DESC DUAL;

Column name	Null?	Data Type
Dummy		varcahr2(1)

Constraints:-

The oracle server uses constraints to prevent invalid data entry into a table.

- **Use of Constraints:-**

  - Enforce Rules on the data in table whenever a row is inserted, updated or deleted from that table. The constraints must be satisfied for the operation to succeed.
  - Prevent the deletion of a table if there are dependencies from other tables.

- **Oracle allows to define Constraints at:**

  - Column Level
  - Table Level

Columns Level Constraints:-

If data constraints are defined as an attribute of a column definition when creating or altering a table structure, they are column level constraints.

Table Level Constraints:-

- If data constraints are defined after defining all table columns attribute when creating or altering a table structure, it is a table level constraint.
- Constraints are stored as a part of the global table definition by the oracle engine in its system tables.

- **Types of Data Constraints:-**

There are two types of data constraints:

1. **I/O constraints:-**
   The input/output data constraints are further divided into different constraints.
   (i) Primary key (ii) Foreign key (iii)Unique Key
2. **Business Rule Constraints:-**
   The Business Rule data constraints are further divided into different constraints.
   (i) Check Constraints. (ii) Not Null

Input/Output Constraints:-

➢ Primary Key Constraint:-

- A Primary key is one or more columns in a table used to uniquely identify **each row** in the table.
- None of the fields that are part of the primary key can contain a null value.
- A single column primary key is called a **Simple key**.
- A multicolumn primary key is called a **Composite primary key**.

A **Primary Key column** in a table has a **special attributes**.

- It defines the column, as a mandatory column. (i.e. the column can not be left blank.)
- The data held across the column must be UNIQUE.

➢ Features of Primary Key:-

- Primary key is a column or set of columns that uniquely identifies a row. Its main purpose is the **Record Uniqueness.**
- Primary key will not allow duplicate values.
- Primary key will also not allow null values.
- Primary key is not compulsory but it is recommended.
- Primary key cannot be LONG or LONG RAW data type.
- Only one Primary key is allowed per table.
- One table can combine upto 16 columns in a composite primary key.
- Primary key helps to identify one record from another record and also helps in relating tables with one another.
- Unique Index is created automatically if there is a primary key.

PRIMARY KEY Constraint Defined at Column Level:-

Syntax:-	**Example:-**
<ColumnName> <Datatype> (<size>) **PRIMARY KEY**	Create table emp (Empno number(5) **PRIMARY KEY,** EName char(10), Sal Number(6,2));

PRIMARY KEY Constraint Defined at Table Level:-

Syntax:-	**Example:-**
PRIMARY KEY (<ColumnName>,<ColumnName>)	Create table emp (Empno number(5), Deptno number(3), EName char(10), **PRIMARY KEY(Empno,Deptno);**);

❖ **PRIMARY KEY Constraint using user-defined Constraint-:-**

Syntax:-	**Example:-**
constraint <user_defined_con_name> **PRIMARY KEY (<ColumnName>)**	Create table emp (Empno number(5), Deptno number(3), EName char(10), **constraint pk_eno** **PRIMARY KEY(Empno);**);

➢ Foreign Key Constraint:-

- Foreign key represent relationships between tables.
- A foreign key is a column or group of columns whose values are derived from the **primary key** or **unique key** of some other table.
- The table in which the foreign key is defined is called a foreign table or Detail table. The table that defines the **primary or unique key** and is referenced by the **foreign key** is called **Primary table or Master table**.
- The master table can be referenced in the foreign key definition by using the clause **REFERENCES**.

➢ **Features of Foreign Key:-**

- Foreign key is column(s) that references a column(s) of a table and it can be the same table also.
- Parent table is being referenced has to be unique or Primary key.
- Child may have duplicates and nulls but unless it is specified.
- Foreign key constraint can be specified on child but not on parent.
- Parent record can be delete provided no child record exists.
- Master table cannot be updated if child record exists.

This constraint establishes a relationship between records across a Master and a Detail table.

- Records cannot be inserted into a detail table if corresponding records in the master table do not exist.
- Records of the master table cannot be deleted if corresponding records in the detail table actually exist.

❖ **FOREIGN KEY Constraint Defined at Column Level:-**

Syntax:-	**Example:-**
<ColumnName> <Datatype> (<size>) **REFERENCES <tablename> [<column name>]** **[ON DELETE CASCADE]** **[ON DELETE SET NULL]**	Create table dept (Empno number(5) **references emp(Empno),** Dno number(3), Dname char(10));

❖ **FOREIGN KEY Constraint Defined at Table Level:-**

Syntax:-	**Example:-**
FOREIGN KEY (<ColumnName> [,<ColumnName>] **REFERENCES <TableName> [(<ColumnName>)]**	Create table dept (Empno number(5), Dno number(3), Dname char(10), **foreign key(Empno) references** **emp(Empno),**);

❖ **FOREIGN KEY Constraint using user-defined Constraint-:-**

Syntax:-	**Example:-**
constraint <user_defined_con_name> **FOREIGN KEY(column_name)** **REFERENCES <TableName> [(<ColumnName>)]**	Create table dept (Empno number(5), Dno number(3), Dname char(10), **constraint fk_empno** **foreign key(Empno)** **references emp(Empno),**);

The foreign key defines in the child table and the table containing the referenced column is the parent table. The foreign key is defined using combination of following keywords.

(1) Foreign Key:-
It is used to define the column in the child table at table level constraint.

(2) References:-
It identifies the and column(s) in the parent table.

(3) ON DELETE CASCADE:-
This is specified in the foreign key definition, if a record is deleted in the master table, all corresponding records in the detail table along with the record in the master table will be deleted. This option used in at the column level as well as table level with foreign key.

(4) ON DELETE SET NULL:-
A foreign key with a SET NULL ON DELETE means that if a record in the parent table is deleted, then the corresponding records in the child table will have the foreign key fields set to be null. The records in the child table will not be deleted. This option used in the column level as well as table level with foreign key.

➢ Unique Key Constraint:-

The unique column constraint **permits multiple entries** of **NULL** to the column. This is the essential difference between the primary key and the unique constraints when applied to the column(s).

- **Features of Unique Key:-**

  - Unique key will not allow duplicate values.
  - Unique index is created automatically.
  - A table can have more than one Unique key which is not possible in Primary key.
  - Primary key cannot be LONG or LONG RAW data type.
  - One table can combine upto 16 columns in a composite primary key.

❖ **UNIQUE KEY Constraint Defined at Column Level:-**

Syntax:-	**Example:-**
<ColumnName> <Datatype> (<size>) **UNIQUE**	Create table emp (Empno number(5) **UNIQUE,** EName char(10), Sal Number(6,2));

❖ **UNIQUE KEY Constraint Defined at Table Level:-**

Syntax:-	**Example:-**
Create table <tablename> **(<columnname1> <Datatype> (<size>),** **<columnname2> <Datatype> (<size>),** **…….,** **<columnnameN> <Datatype> (<size>),** **UNIQUE <ColumnName1>,<ColumnName2>)** **);**	Create table emp (Empno number(5), Deptno number(3), EName char(10), **UNIQUE (Empno,Deptno);**);

❖ **UNIQUE KEY Constraint using user-defined Constraint-:-**

Syntax:-	**Example:-**
constraint <user_defined_con_name> **UNIQUE (<ColumnName1>,<ColumnName2>)**	Create table emp (Empno number(5), Deptno number(3), EName char(10), **constraint uk_eno** **UNIQUE (Empno);**);

Business Rule Constraints:-

➢ Check Constraint:-

Business Rule Validation can be applied to a column by using CHECK constraint. Check constraint must be specified as a logical expression that evaluates either TRUE or FALSE.

If the expression in a check constraint does not return a true/false, the value is **Indeterminate** or **unknown**. Unknown values do not violate a check constraint condition.

LIMITATIONS of Check Constraints:

(1) The condition must be a Boolean expression.
(2) The condition cannot contain sub queries or sequences.
(3)The condition cannot include sysdate, uid and user functions.

❖ Check Constraint Defined at Column Level:-

Syntax:-	**Example:-**
<ColumnName> <Datatype> (<size>) **check (<logical Expression>)**	Create table emp (Empno number(5) **check(empno like 'e%'),** EName char(10), Sal Number(6,2));

❖ Check Constraint Defined at Table Level:-

Syntax:-	**Example:-**
Create table <tablename> **(<columnname1> <Datatype> (<size>),** **<columnname2> <Datatype> (<size>),** **…….,** **<columnnameN> <Datatype> (<size>),** **Check (<Logical Expression>)** **);**	Create table emp (Empno number(5), Deptno number(3), Sal number(6,2), **Check (sal>5000)**);

❖ Check Constraint using user-defined Constraint-:-

Syntax:-	**Example:-**
constraint <user_defined_con_name> **Check (<Logical Expression>)**	Create table emp (Empno number(5), Deptno number(3), EName char(10), **constraint chk_eno check(empno like 'e%')**);

➢ Not Null Constraint:-

- The NOT NULL column constraint ensures that a table column cannot be left empty.
- The NOT NULL constraints ensure that the column contains no null values.
- Column without the not null constraints can contain null values by default.
- The NOT NULL constraint can be specified only at the column level not at the table level.

❖ NOT NULL Constraint Defined at Column Level:-

Syntax:-	**Example:-**
<ColumnName> <Datatype> (<size>) **NOT NULL**	Create table emp (Empno number(5) **not null,** EName char(10), Sal Number(6,2));

❖ NOT NULL constraint using user-defined Constraint-:-

Syntax:-	**Example:-**
<ColumnName> <Datatype> (<size>) **constraint <user_defined_con_name> NOT NULL**	Create table emp (Empno number(5) **constraint nt_null not null**, Deptno number(3), EName char(10));

Default Value Concept:-

- At the time of table creation a default value can be assigned to a column.
- When a record is loaded into the table, and the column is left empty, the oracle engine will automatically load this column with the default value specified.
- The data type of the default value should match the data type of the column.
- The DEFAULT clause can be used to specify a default value for a column.
- Character and date values will be specified in single quotes.

Syntax:-	**Example:-**
<ColumnName> <datatype>(<size>) DEFAULT <value>;	create table test (Empno number(2), Gender char(1) **default 'M'**);

USER_CONSTRAINTS:-

- A table can be created with multiple constraints attached to its column. If a user wishes to see the the table structure along with its constraints, oracle provides the DESCRIBE<tablename> command. This command displays only the column names, data type, size and the not null constraint.
- The information about the other constraints that may be attached to the table columns such as the PRIMARY KEY, FOREIGN KEY, CHECK and so on. Oracle stores such information in a table called USER_CONSTRAINTS.
- USER_CONSTRAINTS provides information bound to the names of all the constraints on the table.
- USER_CONSTARINT contain of multiple columns.

❖ USER_CONSTRAINT table:

Column Name	Description
OWNER	The owner of the constraints.
CONSTRIANT_NAME	The names of the constraint.
TABLE_NAME	The name of the table associated with the constraint.
CONSTRAINT_TYPE	The types of constraint: P:-Primary key Constraint R:-Foreign key Constraint U:-Unique Constraint C:-Check Constraint
SEARCH_CONDITION	The search condition used (for CHECK constraint)
R_OWNER	The owner of the table referenced by the FOREIGN KEY constraint.
R_CONSTRAINT_NAME	The name of the constraint referenced by a FOREIGN KEY constraint.

Example: Select owner, constraint_name,constraint_type,search_condition from user_constraints where table_name='EMP'

OUTPUT:-

owner	constraint_name	constraint_type	search_condition
SB01	PK_ENO	P	-
SB01	FK_DNO	R	-
SB01	CHK_ENO	C	eno like 'E%'

Defining Integrity Constraint via the ALTER TABLE COMMAND:-

(ADD, MODIFY and DROP Constraint with Alter Table Command)

- Integrity constraint can be defined using the **constraint** clause, in the **ALTER TABLE** command.
- Oracle will not allow constraints defined using the ALTER table, to be applied to the table if data previously placed in the table violates such constraints.
- If a primary key constraint was being applied to a table and the column has duplicates values in it, the Primary key constraint will not be set to that column.
- We can only **ADD or DROP** an existing constraint but we can not modify an existing constraint.
- Only **NOT NULL** constraint can be modified.

❖ **To ADD constraint in the existing table:-**

Syntax:- **Alter table <table_name>** **ADD (constraint <constraint_nm>** **<constraint_type> (<ColumnName>);**	**Example 1:-** **Alter table** dept **ADD** primary key(eno); **Example 2:-** **Alter table** emp **ADD** (constraint fk_eno **foreign key**(eno) **references** dept(eno));

❖ **To MODIFY constraint in the existing table:-**

Syntax:- **Alter table <table_name>** **MODIFY (<ColumnName> NOT NULL);**	**Example 1:-** **Alter table** emp **MODIFY (**mgr **NOT NULL)**;

❖ **To DROP constraint in the existing table:-**

Syntax:- **Alter table <table_name>** **DROP (PRIMARY KEY<ColumnName> |** **UNIQUE <ColumnName> |** **constraint <Constraint_name>);**	**Example 1:-** **Alter table** emp **DROP constraint fk_eno;** **Example 2:-** **Alter table** dept **DROP primary key(dno);**

ORACLE FUNCTION:-

- Oracle functions serve the purpose of manipulating data items and returning a result.
- Functions are also capable of accepting user-supplied variables or constant and operating on them. Such variables or constants are called arguments.

There are two types of functions.

1) Group functions 2) Scalar Function

1) Group Function:- (Aggregate Functions)

Functions that act on a **set of values** are called **group function**. For example, SUM, is a function, which calculates the total set of numbers.

A group function returns a single result row for a group of queries rows.

2) Scalar Function:- (Single Row Functions)

Functions that act on **only one value** at a time are called **scalar function**. For example, LENGTH is a function, which calculates the length of one particular string value.

A scalar function returns a one result for every row of queried table.

➢ AGGREGATE FUNCTIONS:-

1) AVG:

Description: - Returns an average value of 'n', ignoring null values in a column.
avg function can be used only with numeric data type.

Syntax:-

AVG([<DISTINCT>|<ALL>] <n>)

Example:-

Select avg(sal) "average sal" from emp;

Output:-

Average sal
1200

2) MIN:

Description: - Returns a minimum value of **expr**.
min function can be used any kind of data type.

Syntax:-

MIN([<DISTINCT>|<ALL>] <expr>)

Example:-

Select min(sal) "Minimum Sal" from emp;

Output:-

Minimum Sal
800

3) MAX:

Description: - Returns a maximum value of **expr**.
max function can be used any kind of data type.

Syntax:-

MAX([<DISTINCT>|<ALL>] <expr>)

Example:-

Select max(sal) "Maximum Sal" from emp;

Output:-

Maximum Sal
8000

4) SUM:

Description: - Returns the sum of the value of 'n'.
sum function can be used only with numeric data type.

Syntax:-

SUM([<DISTINCT>|<ALL>] <n>)

Example:-

Select SUM(sal) "TOTAL Sal" from emp;

Output:-

TOTAL Sal
12000

5) COUNT(EXPR):

Description: - Returns the numbers of rows where **expr** is not null.
sum function can be used only with numeric data type.

Syntax:-

COUNT([<DISTINCT>|<ALL>] <expr>)

Example:-

Select COUNT(emp_no) "No.of employee" from emp;

Output:-

No. of employee
10

6) COUNT(*):

Description: - Returns the numbers of rows in the table, including duplicates and those with nulls.

Syntax:-

COUNT(*)

Example:-

Select COUNT(emp_no) "No.of employee" from emp;

Output:-

No. of employee
15

➢ SCALAR FUNCTIONS:-

- **Numeric Functions:-**

1) ABS:

Description: - Returns the absolute value of n.

Syntax:-

ABS(n)

Example:-

Select abs(-15) from dual;

Output:-

Abs(-15)
15

2) POWER:

Description: - Returns **m** raised to the **n**th power. **n** must be an integer, else an error is returned.

.

Syntax:-

POWER(m,n)

Example:-

Select power(3,2) from dual;

Output:-

power(3,2)
9

3) ROUND:

Description: - Returns a n, rounded to m places to the right of a decimal point. If m is omitted, n is rounded to 0 places. m can be negative to round off digits to the left of the decimal point. M must be an integer.

Syntax:-

ROUND(n[,m])

Example:-

1. Select round(15.19,1) "round" from dual;

Output:-

Round
15.2

2. Select round(124.19,-2) "round" from dual;

Output:-

Round
100

4) SQRT:

Description: - Returns square root of n. if n<0, null. SQRT returns a real result.

Syntax:-

SQRT(n)

Example:-

Select SQRT(25) "SQR ROOT" from dual;

Output:-

SQR ROOT
5

5) TRUNC:

Description: - Returns a number truncated to a certain number of decimal places. The decimal place value must be an integer. If this parameter is omitted, the TRUNC function will truncate the number to 0 decimal places.

Syntax:-

TRUNC(number,[decimal places]))

Example:-

Select TRUNC(125.147,1) "trunc1" TRUNC(125.147,-2) "trunc2" from dual;

Output:-

trunc1	trunc2
125.1	100

6) FLOOR:

Description: - Returns the largest integer value that is equal to or less than a number.

Syntax:-

FLOOR(n)

Example:-

Select FLOOR(24.8) "flr1" from dual;

Output:-

flr1
24

7) CEIL:

Description: - Returns the smallest integer value that is equal to or gretaer than a number.

Syntax:-

CEIL(n)

Example:-

Select CEIL(24.8) "ceil1" from dual;

Output:-

ceil1
25

- **String Functions:-**

1) LOWER:

Description: - Returns char, with all letters in lowercase.

Syntax:-

LOWER(char)

Example:-

Select lower('SPUVVN') "lower" from dual;

Output:-

lower
spuvvn

2) UPPER:

Description: - Returns char, with all letters in uppercase. .

Syntax:-

UPPER(char)

Example:-

Select UPPER('spuvvn') "upper" from dual;

Output:-

upper
SPUVVN

3) INITCAP:

Description: - Returns a string with the first letter of each word in upper case. .

Syntax:-

INITCAP(char)

Example:-

Select INITCAP('spu vvn') "INITCAP" from dual;

Output:-

INITCAP
Spu Vvn

4) LENGTH:

Description: - Returns the length of a word.

Syntax:-

LENGTH(word)

Example:-

Select LENGTH('SPUVVN') "length" from dual;

Output:-

length
6

5) SUBSTR

Description: - Returns a portion of characters, beginning at character m, and going upto character n. If n is omitted, the result returned is upto the last character in the string. The first position of char is 1.

Syntax:-

SUBSTR(<string>,<start position>,[<length>])

Example:-

Select SUBSTR('SPUVVN',3,5) "substr" from dual;

Output:-

substr
UVV

6) LTRIM:

Description: - Removes characters or blanks from the left of char with initial characters removed upto the first character not in set. **'set'** is optional, it default to spaces.

Syntax:-

LTRIM(char,[,set])

Example:-

Select **LTRIM('SPUVVN','S')** "LTRIM" from dual;

Output:-

LTRIM
PUVVN

7)RTRIM:

Description: - Returns char, with final characters removed after the last character not in the set. **'set'** is optional, it default to spaces.

Syntax:-

RTRIM(char,[,set])

Example:-

Select **RTRIM('SPUVVN','N')** "LTRIM" from dual;

Output:-

LTRIM
SPUVV

7)TRIM:

Description: - Removes all specified characters either from the beginning or the ending of a string.

Syntax:-

TRIM ([leading | trailing | both [<trim_character> from]] <string1>)

Where, **leading**- removes trim_string from the front of the string1.

trailing- removes trim_string from the end of the string1.

both - removes trim_string from the front and end of the string1.

→ If none of the above option is chosen, the trim function willl remove trim_string from both front and end of string1.

→ **trim_character** is omitted, the trim function will remove all leading and trailing space from string1.

Example:-

1. Select **TRIM(' SPUVVN ')** "TRIM" from dual;

Output:-

TRIM

SPUVVN

2. Select **TRIM(leading 'S' from 'SPUVVN')** "TRIM" from dual;

Output:-

TRIM

PUVVN

3. Select **TRIM(trailing 'N' from 'SPUVVN')** "TRIM" from dual;

Output:-

TRIM

SPUVV

4. Select **TRIM(both 'S' from 'SPUVVNS')** "TRIM" from dual;

Output:-

TRIM

PUVVN

8)LPAD:

Description: - Returns **char1**, left-padded to length **n** with the sequence of characters specified in **char2**. If char is not specified oracle uses blanks by default.

Syntax:-

LPAD(char1,n [,char2])

Example:-

Select LAPD**('SPUVVN','10','$')** "LPAD" from dual;

Output:-

LPAD

$$$$SPUVVN

9)RPAD:

Description: - Returns **char1**, right-padded to length **n** with the sequence of characters specified in **char2**. If char is not specified oracle uses blanks by default.

Syntax:-

RPAD(char1,n [,char2])

Example:-

Select RAPD**('SPUVVN','10','$')** "LPAD" from dual;

Output:-

LPAD

SPUVVN$$$$

- **Date Functions:-**

1) ADD_MONTHS:

Description: - Returns date after adding the number of months specified in the function.

Syntax:-

ADD_MONTHS(d,n)

Example:-

Select add_months(sysdate,4)) “add_months” from dual;

Output:-

Add_months
01-NOV-24

2) LAST_DAY:

Description: - Returns the last date of the month specified with the function.

Syntax:-

LAST_DAY(d)

Example:-

Select sysdate,last_day(sysdate) “last_day” from dual;

Output:-

Sysdate	last_day
01-JUL-24	31-JUL-24

3) MONTHS_BETWEEN

Description: - Returns number of months between d1 and d2.

Syntax:-

MONTHS_BETWEEN(d1,d2)

Example:-

Select months_between(’01-OCT-24’,’01-AUG-24’) “Months1”,
months_between(’01-AUG-24’,’01-OCT-24’) “Months2” from dual;

Output:-

Months1	Months2
2	-2

4) NEXT_DAY:

Description: - Returns the **date** of the first weekday named by **char** that is after the date named by **date**. **char** must be a day of the week. numeric data type.

Syntax:-

NEXT_DAY(date,char)

Example:-

Select next_day(’01-JUL-24’,’MONDAY’) “next day” from dual;

Output:-

next day
08-JUL-24

➢ Conversion Functions:

1) TO_NUMBER:-

Description: - Converts **char**, a **character** value expressing a number, to a NUMBER datatype.

Syntax:-

TO_NUMBER(char)

Example:-

Select to_char('123') "to_number" from dual;

Output:-

to_number

123

2) TO_DATE:-

Description: - Converts a character field to a date field.

Syntax:-

TO_DATE(char [, fmt])

Example:-

Select to_date('01/09/24','dd/mm/yy') "to_date" from dual;

Output:-

to date

01-SEP-09

3) TO_CHAR:-

(1) TO_CHAR(number conversion)

Description: - Converts a value of a NUMBER datatype to a CHARACTER datatype, using the optional format string. TO_CHAR() accepts a number(n) and a numeric format (fmt) in which the number has to appear.

Syntax:-

TO_CHAR(n [,fmt])

Different Number formats Model:-

Element	Description	Example	Result
9	Numeric position	99999	12345
0	Display leading zeros	09999	01234
$	Floating Dollars Sign	$9999	$1234
.	Decimal point in position specified.	9999.99	1234.00
,	Comma in position specified	99,999	12,345

(2) TO_CHAR(Date conversion)

Description: - Converts a value of a **DATE** datatype to a **CHAR** value, to_char() accept a date, as well as the format in which the date has to appear. Fmt must be a date format. If fmt is omitted, the **date** is converted to a character value using the default date format. i.e. 'dd-mon-yy'.

Different Date formats Model:-

Format	Description
D	Day in a week
DD	Integer Day of Month.
DDD	Integer Day in a Year
DAY	Name of Day the padded with blanks to length of Nine character.
MON	Name of Month three letter abbreviation.
MONTH	Name of Month padded with blank to length of nine character.
YY	Last two digit of year.
YYYY	Last four digit of year.
YEAR	Year in spelled.
WW	Week in a Year
W	Week in a Month
DDTH	Ordinal number (4^{th})
DDSP	Spelled out number (Four)
DDSPTH	Spelled out ordinal number (Fourth)
"of the"	Quoted string is represented in result.
HH	Hours of day
MI	Minutes(0-59)
SS	Seconds(0-59)

Addition and Subtraction of Dates:-

Operation	Result	Description
1) Date+Number	Date	Add number of days into date.
2) Date-Number	Date	Subtract number of days from date.
3) Date- Date	No. Of Days	Subtraction one date from another date.
4) Date+(number/24)	Date	Add a number of hours into a date.