DBMS COMMANDS

09 4Sql commands are instructions, coded into sql statements, which are used to communicate with the database to perform specific tasks, work, functions and queries with data. It is of two types : DML and DDL commands.

DDL Commands:

DDL is a short name of Data Definition Language, which deals with database schemas and descriptions, of how the data should reside in the database. It has 5 commands namely CREATE, ALTER, DROP, TRUNCATE, & RENAME.

- **CREATE**: This command is used to create database (MYSQL) and tables.
- **ALTER**: This command is used to alter the structure of the existing table.
- **DROP**: This command is used to delete or drop the table along with its structure.
- **TRUNCATE**: This command is used to delete all the records from the table including the space allotted to the records.
- **RENAME**: This command is used to rename the table.

DML Commands:

DML is a short name of Data Manipulation Language which deals with data manipulation, and includes most common sql statements such as SELECT, DELETE, UPDATE, INSERT etc and it is used store, modify, retrieve, delete, and update data in tables.

- **SELECT**: This command is used to retrieve the data from table.
- **INSERT**: This command is used to insert the data in table.
- **UPDATE**: This command is used to update the existing the data in the table.
- **DELETE**: This command is used to delete all the records from a table including the spaces allotted for the records but this command can also be used to delete a single record using where condition which is not possible using TRUNCATE command.

DESCRIBE Command:

This command is used to describe the detailed structure of a table which includes or shows the fields/columns/attributes, data type of each column and the constraint used in every column(if used).

DBMS DATATYPES

Every data in database or table is of a particular category. For example the data in 'name' field is of VARCHAR2 type whereas 'phone number' field can be of NUMBER type. The data type available in oracle is: VARCHAR2, NUMBER, & DATE.

- **VARCHAR2**: This data type is used to insert string data in the table. The data while inserting should be represented in single quotes.
- **NUMBER**: This data type is used to insert number in the table. The data is written directly without single quotes.
- **DATE**: This data type is used to insert date in the table in the format DD-MM-YY. Note that MM should be written as JAN instead of 01 and YY should be written as 97 instead of 1997 (In case of 2000 write 00). The date is also written in single quotes.

DISTINCT STATEMENT:

In a table, a column may contain many duplicate values; and sometimes you only want to list the different (distinct) values. The

DISTINCT keyword can be used to return only distinct (different) values.

ORDER BY STATEMENT:

The ORDER BY keyword is used to sort the result-set by one or more columns. The ORDER BY keyword sorts the records in ascending order by default. To sort the records in a descending order, you can use the DESC keyword.

DBMS CONSTRAINTS

SQL constraints are used to specify rules for the data in a table. If there is any violation between the constraint and the data action, the action is aborted by the constraint. Constraints can be specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).

In SQL, we have the following constraints:

- **NOT NULL**: Indicates that a column cannot store NULL value.
- **UNIQUE**: Ensures that each row for a column must have a unique value.
- **PRIMARY KEY**: A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or

more columns) have a unique identity which helps to find a particular record in a table more easily and quickly.

- **FOREIGN KEY**: Ensure the referential integrity of the data in one table to match values in another table.
- **CHECK**: Ensures that the value in a column meets a specific condition.
- **DEFAULT**: Specifies a default value for a column.

DBMS JOINS

SQL joins are used to combine rows from two or more tables.

The types of the different SQL JOINs you can use:

- **INNER JOIN**: Returns all rows when there is at least one match in BOTH tables.
- **LEFT JOIN**: Return all rows from the left table, and the matched rows from the right table.
- **RIGHT JOIN**: Return all rows from the right table, and the matched rows from the left table.
- **FULL JOIN**: Return all rows when there is a match in ONE of the tables.

CREATE COMMAND

1) Syntax for creating a table:

```
CREATE TABLE table_name
(
column_name1 data_type(size),
column_name2 data_type(size),
column_name3 data_type(size),
....
);
```

2) Creating the table:

```
CREATE TABLE People
(
PersonID number(3),
LastName varchar2(25),
FirstName varchar2(25),
Address varchar2(25),
City varchar2(25)
);
```

3) Meaning of this command:

NOTE: People is the table name
Number, varchar2 is the data type
Numbers in braces() is the size of a column
PersonID, LastName,.. is the column name.

4) Result:

After you press execute button in Oracle ISQL plus server the following line should be printed:

```
Table created.
```

DESCRIBE COMMAND

Now you have created the table so it's time to check that how the table looks like or what is the structure of table. To view the structure of table use DESCRIBE or DESC command.

1) Syntax for using DESCRIBING or DESC command:

```
desc table_name;
```

2) Writing the command:

```
desc people;
```

3) Result:

NAME	NOT NULL?	TYPE
PERSONID		NUMBER(03)
LASTNAME		VARCHAR2(25)
FIRSTNAME		VARCHAR2(25)
ADDRESS		VARCHAR2(25)
CITY		VARCHAR2(25)

**Structure of the above created table.

INSERT COMMAND

1) Syntax for INSERT command

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

2) Inserting the records in table:

```
insert into people values(
100,'Rai','Sahil','LPU','Jalandhar'
);
```

3) Meaning of this command:

NOTE: i) People is the table name
ii) 100, Rai, Sahil...are the values for respective columns
iii) Values in string are put in single quotes as they are of varchar2 datatype.

4) Result of the INSERT command:

After you press the execute button in ORACLE isql plus this message will be displayed if there is no error in the code.

```
1 row created.
```

SELECT COMMAND

1) Syntax for SELECT command:

```
SELECT column_name,column_name
FROM table_name;
```

And

```
SELECT * FROM table_name;
```

2) Selecting or retrieving the records from table:

```
select * from people;
```

3) Meaning of the SELECT command:

NOTE: People is the table name
*** means selecting all the columns at once.**
If multiple columns needs to be selected,
but not all then column names can be seprated
using comma as given in syntax.

4) Result of the SELECT command:

PERSONI	LASTNAM	FIRSTNAM	ADDRES	CITY

D	E	E	S	
100	RAI	SAHIL	LPU	JALANDHAR

ALTER COMMAND

1) Syntax of ALTER command:

a) **To ADD a column:** To add a new column in a table.

```
ALTER TABLE table_name
ADD column_name datatype
```

b) **To DROP a column:** To drop any column from a table.

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

c) **To MODIFY a column**: To modify the size of column i.e. the number of characters which a column accepts. Remember the size can only be increased.

```
ALTER TABLE table_name
MODIFY column_name datatype
```

2) Using ALTER command:

a) **Adding a column:**

```
alter table people add DOB DATE;
```

b) **Drop a column:**

```
alter table people drop column DOB;
```

c) **Modifying a column:**

```
alter table people modify gender varchar2(6);
```

*Previous size of gender column was less than six. Using DESC command the change in size can be observed.

3) Meaning of ALTER command:

a) **ADD:**

NOTE: People is the table name
DOB is the name of the column
DATE is the data type.

b) **DROP:**

NOTE: People is the table name
DOB is the name of the column

c) **MODIFY:**

NOTE: People is the table name
GENDER is the name of the column
VARCHAR2 is the data type.
SIX is the given size during modify,
The size was less than six before MODIFY.

4) Result:

a) **ADD:**

Table altered.

b) **DROP:**

Table altered.

c) **MODIFY:**

Table altered.

UPDATE COMMAND

1) Syntax of UPDATE command:

```
UPDATE table_name
SET column1=value1,column2=value2,...
WHERE some_column=some_value;
```

2) Using UPDATE command:

```
update people
set city = 'Phagwara'
where personid = 100;
```

3) Meaning of the UPDATE command:

NOTE: People is the table name
city is the column name
'Phagwara' is the value which will replace 'Jalandhar' in city column
where is used for giving condition.
personid is the column name.
where 100 is the value for personid.

4) Result:

1 row updated.

DELETE COMMAND

1) Syntax of DELETE command:

```
DELETE FROM table_name
WHERE some_column=some_value;
```

2) Using DELETE command:

```
delete from people
where personid = 101;
```

3) Meaning of DELETE command:

NOTE: People is the table name
personid is the name of the column
101 is the one of the value in personid column,
which is a unique value.
where clause is used for giving condition.

4) Result:

1 row deleted.

DROP COMMAND

1) Syntax of DROP command:

```
DROP TABLE table_name
```

2) Using DROP command:

```
drop table person;
```

3) Meaning of DROP command:

NOTE: Person is the name of the table.

4) Result:

Table dropped.

TRUNCATE COMMAND

1) Syntax of TRUNCATE command:

```
TRUNCATE TABLE table_name
```

2) Using TRUNCATE command:

```
truncate table people;
```

3) Meaning of TRUNCATE command:

NOTE: People is the name of the table.

4) Result:

Table truncated.

RENAME COMMAND

1) Syntax of RENAME command:

```
RENAME old_table_name TO new_table_name
```

2) Using RENAME command:

```
rename people to people_01;
```

3) Meaning of RENAME command:

NOTE: People is the older table name
People_01 is the new table name

4) Result:

Table renamed.

NOT NULL CONSTRAINT

1) Using NOT NULL:

```
create table employ(
emp_id number(3) not null,
emp_name varchar2(20) not null,
emp_dob date);
```

2) Checking of NOT NULL:

I) Using the DESCRIBE command:

NAME	NOT NULL?	TYPE
EMP_ID	**NOT NULL**	NUMBER(03)
EMP_NAME	**NOT NULL**	VARCHAR2(20)
EMP_DOB		DATE

II) Trying to insert a NULL value:

```
 insert into employ values(100,'','16-JAN-97')
*

ERROR at line 1:
ORA-01400: cannot insert NULL into ("SCOTT"."EMPLOY"."EMP_NAME")
```

UNIQUE KEY CONSTRAINT

1) Using UNIQUE KEY:

```
create table employ(
emp_id number(3) not null unique,
emp_name varchar2(20) not null,
emp_dob date);
```

** For defining a UNIQUE constraint on multiple columns:

```
create table employ(
emp_id number(3) not null,
emp_name varchar2(20) not null,
emp_dob date,
CONSTRAINT uc_Employ UNIQUE (emp_id,emp_dob)
);
```

2) UNIQUE constraint on ALTER table:

```
alter table employ add unique (emp_id);
```

** For defining a UNIQUE constraint on multiple columns:

```
ALTER TABLE employ ADD CONSTRAINT uc_employ UNIQUE (emp_id,emp_dob);
```

NOTE: Employ is the table name.
'uc_employ' is a user defined name,
it can be anything depending upon the user.
emp_id, emp_dob is the column name.
CONSTRAINT is the keyoword in sql.

3) DROP a UNIQUE constraint:

```
ALTER TABLE employ
DROP unique(emp_id);
```

**To DROP UNIQUE constraint on multiple columns:

```
ALTER TABLE employ DROP CONSTRAINT uc_Employ;
```

NOTE: Employ is the table name.
'uc_employ' is a user defined name,
it can be anything depending upon the user.
'uc_employ' is the name given to the
group of columns having unique key.
CONSTRAINT is the keyoword in sql.

4) Checking of UNIQUE KEY constraint:

**Inserting a duplicate record in a column having UNIQUE key:

```
insert into employ values(100,'Ram','18-NOV-99')
*

ERROR at line 1:
ORA-00001: unique constraint (SCOTT.SYS_C007879) violated
```

PRIMARY KEY CONSTRAINT

1) Using PRIMARY KEY constraint:

```
create table employ(
emp_id number(3) primary key,
emp_name varchar2(20) not null,
emp_dob date not null);
```

** For defining a PRIMARY KEY constraint on multiple columns:

```
create table employ(
emp_id number(3) not null,
emp_name varchar2(20) not null,
emp_dob date,
CONSTRAINT pk_Employ primary key (emp_id,emp_dob)
);
```

NOTE: Employ is the table name.
'pk_employ' is a user defined name,
it can be anything depending upon the user.
'uc_employ' is the name given to the
group of columns having primary key.
CONSTRAINT is the keyoword in sql.
Primary key when applied to more than one column is known as Candidate key.

2) PRIMARY KEY constraint on ALTER table:

```
ALTER TABLE employ ADD PRIMARY KEY (emp_id);
```

** For defining a PRIMARY KEY constraint on multiple columns:

```
ALTER TABLE Employ ADD CONSTRAINT pk_Employ PRIMARY KEY (emp_id,emp_dob);
```

3) DROP PRIMARY constraint:

```
ALTER TABLE employ DROP PRIMARY KEY
```

**To DROP PRIMARY constraint on multiple columns:

```
ALTER TABLE employ DROP CONSTRAINT pk_Employ
```

4) Checking of PRIMARY KEY constraint:

**Inserting a PRIMARY KEY in a table already having primary key

```
alter table employ add primary key(emp_dob)
                       *
ERROR at line 1:
ORA-02260: table can have only one primary key
```

FOREIGN KEY CONSTRAINT

1) Using FOREIGN KEY constraint:

```
create table persons(p_id number(3) primary key,
p_name varchar2(20),
address varchar2(20)
);
```

```
create table orders(order_id number(3) primary key,
person_id number(3) references persons(p_id),
order_no number(5),
order_date DATE
);
```

Create table course (name varchar20, roll int references student(roll)

** For defining a FOREIGN KEY constraint on multiple columns:

```
create table persons(p_id number(3) primary key,
p_name varchar2(20),
address varchar2(20)
);
```

```
create table orders(order_id number(3) primary key,
person_id number(3),
order_no number(5),
order_date DATE,
CONSTRAINT fk_PerOrders  foreign key(person_id)  REFERENCES persons(p_id)
);
```

NOTE: Persons & Orders are the table names.
'fk_PerOrders' is a user defined name,
it can be anything depending upon the user.
'fk_PerOrders' is the name given to the
column having foreign key.
CONSTRAINT is the keyoword in sql.
Person_id is the column name in orders table.
p_id is the column name in persons table.
Foreign key in one table is the primary key in other table.

2) Inserting records in table having FOREIGN KEY constraint:

```
insert into persons values(100,'Sahil','LPU');

insert into orders values(273,(select p_id from persons where p_id=100),10000,'16-NOV-16');
```

NOTE: Person and Orders are the table names.
p_id is the column name in persons table.
person_id is the column under orders table,
which is refered to p_id in person table.

3) FOREIGN KEY constraint using ALTER table:

```
ALTER TABLE Orders
ADD FOREIGN KEY (person_id)
REFERENCES Persons(p_id);
```

** For defining a PRIMARY KEY constraint on multiple columns:

```
ALTER TABLE Orders
ADD CONSTRAINT fk_PerOrders
FOREIGN KEY (person_id)
REFERENCES Persons(p_id);
```

3) DROP FOREIGN KEY constraint:

```
ALTER TABLE Orders DROP CONSTRAINT fk_PerOrders
```

NOTE: orders is the table name.
fk_PerOrders is a user defined name,
it can be anything depending upon the user.
'fk_PerOrders' is the name given to the
column having foreign key.
This will only work for FOREIGN KEY with a constriant name,
which is 'fk_PerOrders'.
For dropping a foreign key without constriant name,
You may need to drop the complete column having foreign key.

4) Checking the FOREIGN KEY constraint:

** Dropping the table having PRIMARY KEY which is referred as FOREIGN KEY to the other table:

```
drop table persons
           *

ERROR at line 1:
ORA-02449: unique/primary keys in table referenced by foreign keys
```

CHECK CONSTRAINT

1) Using CHECK constraint:

```
create table person(p_id number(3) primary key check (p_id>100),
 p_name varchar2(20),
 address varchar2(20)
 );
```

**For defining a CHECK constraint on multiple columns:

```
create table person(p_id number(3) primary key,
 p_name varchar2(20),
 address varchar2(20),
 CONSTRAINT chk_person CHECK(p_id>0 and address='LPU')
 );
```

NOTE: Person is the table name.
CHECK will check if p_id is greater than 100 or not.
Address will also be checked if it is lpu or not.
If both the conditions are correct then record is entered else not.

2) CHECK constraint on ALTER table:

```
ALTER TABLE Persons ADD CHECK (P_Id>0)
```

**For defining a CHECK constraint on multiple columns:

```
ALTER TABLE Persons ADD CONSTRAINT chk_Person CHECK (P_Id>0 AND Address='Lpu');
```

3) DROP CHECK constraint:

```
ALTER TABLE Persons DROP CONSTRAINT chk_Person
```

NOTE: persons is the table name.
'chk_person' is a user defined name,
it can be anything depending upon the user.
'chk_person' is the name given to the
columns having CHECK constraint.
This will only work for CHECK with a constriant name,
which is 'chk_person'.
For dropping a CHECK without constriant name,
You may need to drop the complete column having CHECK.

4) Checking the CHECK constraint:

```
insert into person values(100,'Sahil','LPU')
*

ERROR at line 1:
ORA-02290: check constraint (SCOTT.SYS_C007905) violated
```

DEFAULT CONSTRAINT

1) Using DEFAULT constraint:

```
create table person(p_id number(3) primary key,
 p_name varchar2(20),
 address varchar2(20) default 'LPU'
 );
```

2) DEFAULT constraint using ALTER table:

```
ALTER TABLE Person MODIFY ADDRESS DEFAULT 'LPU'
```

3) DROP DEFAULT constraint:

```
ALTER TABLE Persons ALTER COLUMN address DROP DEFAULT
```

4) Checking DEFAULT constraint:

**Type in the INSERT command like this to access DEFAULT value:

```
insert into person values(100,'Sahil',default);
```

RENAMING A COLUMN

1) Syntax of RENAMING a column:

```
ALTER TABLE table_name
  RENAME COLUMN old_name TO new_name;
```

2) Renaming the column:

```
ALTER TABLE Persons RENAME COLUMN P_ID TO PERSON_ID;
```

3) Meaning of the command:

NOTE: Person and Orders are the table names.
p_id is the older column name in persons table.
person_id is the new column name.

4) Result:

Table altered.

CREATING NEW TABLE FROM AN EXSISTING TABLE

1) Syntax:

```
CREATE TABLE new_table_name (column1, column2, column3,.....column_n)
AS SELECT col1, col2, col3,.....col_n FROM old_table_name;
```

2) Using the command:

```
CREATE TABLE customer(cust_id, cust_name, cust_address, cust_dob)
AS SELECT ord_id, ord_name, ord_address, ord_date FROM old_table_name;
```

NOTE: Before creating a new table from an old table it is important, that the old table exists.
cust_id, cust_name, cust_address, cust_dob are the columns in customer table (new table).
ord_id, ord_name, ord_address, ord_date are the columns in order table (old table).
The data type of the columns in the new table will be the same as in the columns of old table.
The NOT NULL and all the other constraints are in the old table are automatically inherited into new table.
If there are any values or records in the old table they will be automatically added in the new table also.

** create a SQL table from another table without copying any values from the old table:

Syntax:

```
CREATE TABLE new_table_name (column1, column2, column3,.....column_n)
AS SELECT col1, col2, col3,.....col_n FROM old_table_name WHERE 1=2;
```

Using the command:

```
CREATE TABLE customer(cust_id, cust_name, cust_address, cust_dob)
AS SELECT ord_id, ord_name, ord_address, ord_date FROM old_table_name WHERE 1=2;
```

3) Result:

```
Table created.
```

ORDER BY CLAUSE

1) Syntax:

```
SELECT * FROM table_name
ORDER BY column_name;
```

2) Using ORDER BY:

```
SELECT * FROM customer
ORDER BY cust_name;
```

NOTE: customer is the table name
By default the records will arranged in the ASCENDING order.
To arrange the records in DESCENDING order use the keyword DESC,
after writting the table name.

3) Result:

CUST_ID	CUST_NAME	CUST_ADDRESS	CUST_DOB
105	AAKASH	LPU	16-JAN-97
103	BINDESH	LPU	23-NOV-96
101	CINTHIYA	LPU	24-FEB-97
104	SAHIL	LPU	16-MAR-97
102	ZAKARYA	LPU	22-SEP-96

LIKE CLAUSE

1) Syntax:

```
SELECT * FROM table_name
where column_name like 'character%' ;
```

Or

```
SELECT * FROM table_name
where column_name like '_character%' or/and
column_name like '_character%' ;
```

2) Using LIKE Clause:

```
SELECT * FROM customer
where cust_name like 'ch%' ;
```

NOTE: customer is the table name
cust_name is the column name.
'LIKE' is a clause.
'ch%' represents that select details of only those names which starts from 'ch'.

Or

```
SELECT * FROM customer
where cust_name like '_a%' or cust_name like '_s%' ;
```

NOTE: customer is the table name
cust_name is the column name.
'LIKE' is a clause.
'_' underscore represents a single character before 'a%' and 's%'.
Only details of those name will be selected where a or s are second characters.
In place of 'or', 'and' can be used.

3) Result:

CUST_ID	CUST_NAME	CUST_ADDRESS	CUST_DOB
108	CHARLIE	LPU	16-JAN-97
111	CHANDRA.V	LPU	23-NOV-96
116	CHATUR	LPU	24-FEB-97

Or

CUST_ID	CUST_NAME	CUST_ADDRESS	CUST_DOB
105	AAKASH	LPU	16-JAN-97
104	SAHIL	LPU	16-MAR-97
107	ASHISH	LPU	22-SEP-96

IN CLAUSE

1) Syntax:

```
SELECT * FROM table_name
where column_name in ('value1','value2','value3'...) ;
```

2) Using IN CLAUSE:

```
SELECT * FROM customer where
cust_name in ('Sameer','Sameira','Rehman') ;

NOTE: customer is the table name
      cust_name is the column name.
      'IN' is a clause.
      Only details of Sameer, Sameira & Rehman
      will be displayed.
```

3) Result:

CUST_ID	CUST_NAME	CUST_ADDRESS	CUST_DOB
120	SAMEER	LPU	16-JAN-97
131	SAMEIRA	LPU	16-MAR-97
115	REHMAN	LPU	22-SEP-96

*there is one more clause called 'NOT IN' clause which is just reverse of 'IN' clause where details of all the records will be displayed except those records/names/values provided in 'NOT IN' clause.(Syntax and usage is same as 'IN' clause).

GROUP BY CLAUSE

1) Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

2) Use of GROUP BY Clause:

SELECT emp_no, count(emp_name)
from customer group by emp_no;

NOTE: customer is the table name.
emp_no and emp_name is column name.
count is a function to count no. of records in a specified column.
group by groups the records on the basis of count.

3) Result:

EMP_NO	COUNT(ENAME)
101	1
102	1
103	1
104	1
105	1

HAVING CLAUSE

1) Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

2) Use of HAVING Clause:

```
select empname,count(deptno) from employ
where empname like 'A%' or empname like 'P%'
group by empname having count(deptno)>=1
```

NOTE: employ is the name of the table.
empname and deptno are the column names.
'like' and 'having' is a keyword.
count will count the number of records in deptno column by matching the where condition.

3) Result:

EMPNAME	COUNT(DEPTNO)
ABC	1
PQR	1

SQL FUNCTIONS

**for implementing every sql functions we will use dummy table named as dual.

1) abs(number)

This function Returns modulus of negative number.

Syntax:

SELECT abs(-126) “ABSOLUTE” FROM dual;

OUTPUT:

ABSOLUTE
126

2) power (m,n)

This function returns the power of a given number where m is number and n is power.

Syntax:

SELECT power(3,3) “POWER” FROM dual;

OUTPUT:

POWER
27

3) Round(number)

This function Returns round off value for a decimal number. If decimal point is greater than or equal to 0.5 then the next higher value is returned else the same number is returned.

Syntax:

SELECT round(178.499) “ROUND” FROM dual;

OUTPUT:

ROUND
178

4) Truncate(n,m)

This function Returns the number after mentioning the number of decimal parts to be shown where n is the number and m is the number of decimal parts to be included.

Syntax:

SELECT trunc(178.599,2) “TRUNCATE” FROM dual;

OUTPUT:

TRUNCATE
178.59

5) sqrt(number)

This function Returns square root of any given number.

Syntax:

SELECT sqrt(25) “SQUARE ROOT” FROM dual;

OUTPUT:

SQUARE ROOT
5

6) Greatest (number1, number2, number3….)

This function Returns the greatest number amongst the group of numbers.

Syntax:

SELECT greatest (14,22,7) “GREATEST” FROM dual;

OUTPUT:

GREATEST
22

7) Least(number1, number2, number3…..)

This function Returns the least number amongst all the numbers.

Syntax:

SELECT least(14,22,7) “LEAST” FROM dual;

OUTPUT:

LEAST
7

7) Modulus (n,m)

This function Returns remainder of the two provided numbers where n is the number divided by m.

Syntax:

SELECT mod(15,7) “MODULUS” FROM dual;

OUTPUT:

MODULUS
1

8) Floor(number)

This function Returns the same base value of any provided number.

Syntax:

SELECT floor(126.99) “FLOOR” FROM dual;

OUTPUT:

FLOOR
126

9) Ceil(number)

This function Returns the next higher value for any given number.

Syntax:

SELECT ceil(126.12) ”CEIL” FROM dual;

OUTPUT:

CEIL
127

STRING FUNCTIONS

1) Lower (character/word/sentence)

This function Returns the provided character in lower case.

Syntax:

SELECT lower('SAHIL') "LOWER" FROM dual;

OUTPUT:

LOWER
Sahil

2) Initcap(character/word/sentence)

This function Returns the word with its first character being capitalized.

Syntax:

SELECT initcap('sahil rai') "INITCAP" FROM dual;

OUTPUT:

INITCAP
Sahil Rai

3) Upper(character/word/sentence)

This function Returns the character or word with complete upper case.

Syntax:

SELECT upper(‘sahil rai’) “UPPER” FROM dual;

OUTPUT:

UPPER
SAHIL RAI

4) Length(character/word/sentence)

This function Returns the length of the word or sentence in integer or number.

Syntax:

SELECT length(‘sahil rai’) “LENGTH” FROM dual;

OUTPUT:

LENGTH
9(counts the space also)

5) Ltrim(word/sentence, character(to be removed))

This function Returns the string after omitting the character mentioned in the function from the left.

Syntax:

SELECT ltrim(‘SAHIL’,’S’) “LTRIM” FROM dual;

OUTPUT:

LTRIM
AHIL

6) Rtrim(word/sentence, character(to be removed))

This function Returns the string after omitting the character the mentioned in the function from the right.

Syntax:

SELECT rtrim('SAHIL','L') "RTRIM" FROM dual;

OUTPUT:

RTRIM
SAHI

7) Trim(TRAILING ‘character’ from word/sentence)

Or

Trim(LEADING ‘character’ from word/sentence)

Or

Trim(BOTH ‘character’ from word/sentence)

This function Returns the word or sentence after removing the specified character from left or front in case of LEADING, from right or last in case of TRAILING and from both the ends in case of BOTH.

Syntax:

SELECT trim(TRAILING ‘M’ from MADAM) “TRIM” FROM dual;

SELECT trim(LEADING ‘M’ from MADAM) “TRIM” FROM dual;

SELECT trim(BOTH ‘M’ from MADAM) “TRIM” FROM dual;

OUTPUT:

TRIM
MADA

TRIM
ADAM

TRIM
ADA

AVERAGE FUNCTIONS

1) Average(column name)

This function Returns the average of all the numeric values in a particular column.

Syntax:

SELECT avg(salary) “AVERAGE SALARY” FROM employ;

OUTPUT:

AVERAGE SALARY
47500

2) Minimum (column name)

This function Returns the minimum value amongst all the values in a column.

Syntax:

SELECT min(SALARY) “MINIMUM SALARY” FROM employ;

OUTPUT:

MINIMUM SALARY
23000

3) Maximum(column name)

This function Returns the maximum value multiple values in a column.

Syntax:

SELECT max(SALARY) “MAXIMUM SALARY” FROM employ;

OUTPUT:

MAXIMUM SALRY
90000

4) Sum(column name)

This function Returns the sum of the values in a column.

Syntax:

SELECT sum(SALARY) “SUM SALARY” FROM employ;

OUTPUT:

SUM SALARY
380000

5) Count(column name) & Count(*)

The difference between count(column name) and count(*) is that the count(column name) will count the values present in that column whereas the count(*) will count all the rows in a table.

Syntax:

SELECT count(SALARY) “COUNT SALARY” FROM employ;

OUTPUT:

COUNT SALARY
8