BASIC SQL COMMANDS

SQL STATEMENTS

SQL statements are classified as follows:

Data Retrieval Statement:

SELECT is the data extracting statement which retrieves the data from the database.

Data Manipulation Language (DML):

This language constitutes the statements that are used to manipulate with the data. It has three commands, which are INSERT, UPDATE and DELETE.

Data Definition Language (DDL):

This is the language used to define the structure of the tables. It sets up, changes, and removes data structures from the tables. It uses 5 commands, which are CREATE, ALTER, DROP, RENAME and TRUNCATE.

Data Transaction Language (DTL):

This is the language used to do undo and redo the transaction performed in the database. The commands are Commit, Rollback, and Save Point

Data Control Language:

This language is used to sanction the rights to the users to use the other user's database objects. The commands are Grant, Revoke

BASE SCHEMA

EMPLOYEE

```
    Name                          Type
 --------------------------    ----------------------
 EMPLOYEE_ID                      NUMBER(3)
 FIRST_NAME                       VARCHAR2(10)
 LAST_NAME                        VARCHAR2(10)
 MGR                              NUMBER(4)
 HIRE_DATE                        DATE
 JOB_ID                           VARCHAR2(10)
 SALARY                           NUMBER(10)
 COMMISION                        NUMBER(8)
 DEPTNO                           NUMBER(2)
```

DEPARTMENT

```
Name                    Type
---------------      -----------------
DEPTNO                NUMBER(2)
DNAME                 VARCHAR2(14)
LOC                   VARCHAR2(13)
```

BONUS

```
Name                     Type
----------------       -------------------
ENAME                  VARCHAR2(10)
JOB                    VARCHAR2(9)
SAL                     NUMBER(10,2)
COMM                    NUMBER(10)
```

JOBGRADE

```
Name                      Type
------------------   ---------------------
JOB_ID               VARCHAR2(10)
GRADE                 NUMBER
LOSAL                 NUMBER
HISAL                 NUMBER
```

DATA TYPES IN ORACLE:

Data Type	**Description**
VARCHAR2(size)	Variable-length character data
CHAR(size)	Fixed-length character data
NUMBER(p,s)	Variable-length numeric data
DATE	Date and time values
LONG	Variable-length character data up to 2 gigabytes
CLOB	Character data up to 4 gigabytes
RAW and LONG RAW	Raw binary data
BLOB	Binary data up to 4 gigabytes
BFILE	Binary data stored in an external file; up to 4 gigabytes
ROWID	A 64 base number system representing the unique address of a row in its table

ORACLE 9I TABLE STRUCTURES

- Table can be created at any time
- No need to specify the size of table, the size is ultimately defined by the amount of space allocated to the database as a whole.
- Tables can have up to 1000 columns

NAMING RULES

Table names and Column names

- Must begin with a letter
- Must be 1-30 characters long
- Must contain only A-Z,a-z,0-9,_,$,#
- Must not duplicate the name of another object owned by the same user
- Must not be a reserved word

Data Definition Language (DDL)

1. Create 2. Alter 3. Drop 4. Truncate 5. Rename

1. a. Creating a table

Syntax:

```
Create table <Table Name>
( <Field1> <Data Type> <(width) <constraints> ,
  <Field2> <Data Type> <(width)> <constraints>,
.................................);
```

Example:

```
SQL> create table employee

( employee_id number(3),

first_name varchar2(10),

last_name varchar2(10),

mgr number(4),
hire_date date,

job_id varchar2(10),

salary number(10),

commision number(8),

deptno number(2));
```

Output:

```
Table created.
```

Example:

```
SQL> create table department

(deptno number(2),

 dname varchar(14),

  loc varchar(13));
```

Output:

```
Table created.
```

Note:

Other tables can be created in the similar way.

b. To view the Structure of the table, desc command is used

```
SQL> desc employee;

 Name                                      Null?      Type
 ----------------------------------------- -------- ---------------
EMPLOYEE_ID                                          NUMBER(3)
 FIRST_NAME                                          VARCHAR2(10)
 LAST_NAME                                           VARCHAR2(10)
 MGR                                                 NUMBER(4)
 HIRE_DATE                                           DATE
 JOB_ID                                             VARCHAR2(10)
 SALARY                                             NUMBER(10)
```

```
COMMISION                              NUMBER(8)
DEPTNO                                 NUMBER(2)
```

2. Alter Table Statement:

Alter command is used to perform the following action on the table:

a. Adding column in the existing table
b. Increasing and decreasing the column size and changing data types
c. Dropping column
d. Renaming the column
e. Adding and dropping constraints to the table(discussed in constraints topics)
f. Enabling & disabling constraints in the table(discussed in constraints topics)

a. To Add a column to the table (structure)

Add option is used to add a new column

Syntax:

Alter Table <Table-Name> Add <Field Name> <Type> (width);

Example:

```
SQL> alter table employee add address varchar2 (20);
```

Output:

```
Table altered.
```

b. To Modify a field of the table

- Increase the width or precision of numeric column
- Increase the width of numeric or character columns
- Decrease the width of the column only if the column contains only null values or if the table has no rows
- Change the data type only if the column contains null values

Syntax:

Alter Table <tablename> MODIFY (<column name > < newdatatype>);

Example:

```
SQL> alter table employee modify address varchar2 (10);
```

Output:

Table altered.

c. To Drop a field of the table

Drop option is used to delete a column or remove a constraint

Syntax:

Alter Table <tablename> DROP COLUMN < column name>;

Example:

```
 SQL> alter table employee drop column address;
```

Output:

Table altered.

d.To rename a column

Syntax:

ALTER TABLE <tablename> RENAME COLUMN <oldcolumnname> TO

<newcolumn name>

Example:

```
SQL> alter table employee rename column mgr to manager;
```

Output:

Table altered.

- **To Drop a table - Deletes a Table along with all contents**

Syntax:

Drop Table <Table-Name>;

Example:

```
 Drop Table Student_table;
```

Output:

Table Dropped

- **To Truncate a table - Deletes all rows from a table ,retaining its structure**

Syntax: Truncate Table <tablename>

Example:

```
SQL> truncate table employee;
```

Output:

Table truncated.

g. To rename a table- Renames a table with new name

Syntax:

Rename <oldtablename> To <newtablename>

Example:

```
 SQL> rename employee to emp;
```

Output:

```
Table renamed
```

<u>Data manipulation Language (DML)</u>

1. Insert 2. Delete 3. Update 4.Select

1 Insert command is used to load data into the table.

a. Inserting values from user

Syntax:

Insert into <tablename> values (val1,val2 …);

Example:

```
SQL> insert into department values(10,'accounts','chennai');
```

Output:

```
1 row created.
```

b. Inserting values for the specific columns in the table

Syntax:

Insert Into <Table-Name> (Fieldname1, Fieldname2, Fieldname3,..) Values (value1, value2, value3,..);

Example:

```
 SQL> insert into department (deptno,dname)values(20,'finance');
```

Output:

```
1 row created.
```

c. ***Inserting interactively(Inserting ,ultiple rows by using single insert command)***

Syntax:

Insert Into <tablename> Values(&<column name1> , &<column name2> …);

Example:

```
SQL> insert into employee values(&empid,'&fn','&ln',&mgr,'&hdate','&job',&sal,
    &comm,&dept);
Enter value for empid: 111
Enter value for fn: Smith
Enter value for ln: Ford
Enter value for mgr: 222
Enter value for hdate: 21-jul-2010
Enter value for job: J1
Enter value for sal: 30000
Enter value for comm: 0.1
Enter value for dept: 10
old   2: &comm,&dept)
new   2: 0.1,10)
```

Output:

```
1 row created.
```

Note: Column names of character and date type should be included with in single quotation.

- **Inserting null values**

Syntax:

Insert Into <tablename> Values (val1,’ ‘,’ ‘,val4);

Example:

```
insert into department  values( ‘101’,’’,chennai);
```

Output:

```
1 row created.
```

2. To Delete rows from a table

Syntax:

Delete from <table name> [where <condition>];

Example:

a) TO delete all rows:

```
SQL> delete from department;
```

Output:

```
89 rows deleted.
```

b) conditional deletion:

```
SQL> delete from department where loc='chennai';
```

Output:

```
1 row deleted.
```

3. Modifying (Updating) Records:

a. Updating single column

Syntax:

UPDATE <table name> Set <Field Name> = <Value> Where <Condition>;

Example:

```
SQL> update department set loc='Hyderabad' where deptno=20;
```

Output:

```
1 row updated.
```

Note: Without where clause all the rows will get updated.

b. Updating multiple column [while updating more than one column, the column must be separated by comma operator]

Example: SQL> update department set loc='Hyderabad', dname= ‘cse’ where deptno=20;

Output:

```
1 row updated.
```

4. Selection of Records [Retrieving (Displaying) Data:]

Syntax:

Select <field1, field2 …fieldn> from <table name> where <condition>;

Example:

```
a) SQL> select * from department;
```

Output:

```
   DEPTNO     DNAME          LOC
---------- -------------- -------------
      10           accounts        chennai
      20           finance         Hyderabad
      30           IT              Bangalore
      40           marketing       chennai
```

Example:

```
b) SQL> select dname, loc from department;
```

Output:

```
DNAME          LOC
-------------- -------------
accounts       chennai
finance        Hyderabad
IT             Bangalore
marketing      Chennai
```

- **Using Alias name for a field**

Syntax:

Select <col1> <alias name 1> , <col2> < alias name 2> from < tab1>;

Example:

```
SQL> select dname, loc as location from department;
```

Output:

```
DNAME          LOCATION
-------------- -------------
accounts       chennai
finance        Hyderabad
IT             Bangalore
marketing      chennai
```

- **With distinct clause [Used to retrieve unique value from the column]**

Syntax:

Select distinct <col2> from < tab1>;

Example:

```
SQL> select distinct loc from department;
```

Output:

```
LOC
-------------
chennai
Bangalore
Hyderabad
```

- **Creating Table using subquery**

Syntax:

Create table <new _table_name> as Select <column names> from <old_table_name>;

Example:

```
 SQL> create table copyOfEmp as select * from employee;
```

Output:

```
Table created.
```

- **To view the contents of new Table**

```
SQL> select * from copyofemp;
```

Output:

```
EMPLOYEE_ID  FIRST_NAME  LAST_NAME          MANAGER  HIRE_DATE  JOB_ID

SALARY  COMMISION    DEPTNO

 111            Smith           Ford              222           21-JUL-10    J1          30000

0.1             1 0
```

- **To create a table with same structure as an existing table**

Syntax:

Create table <new _table_name> as Select <column names> from<old_table_name>

where 1=2;

Example:

```
create table copyOfEmp2 as select * from employee where 1=2;
```

Output:

```
Table created.
```

```
SQL> select * from copyofemp2;
```

Output:

```
no rows selected
```

```
SQL> desc copyofemp2;
```

Output:

```
 Name                                      Null?    Type
 ----------------------------------------- -------- -----------------
 EMPLOYEE_ID                                        NUMBER(3)
 FIRST_NAME                                         VARCHAR2(10)
 LAST_NAME                                          VARCHAR2(10)
 MANAGER                                            NUMBER(4)
 HIRE_DATE                                          DATE
 JOB_ID                                             VARCHAR2(10)
 SALARY                                             NUMBER(10)
 COMMISION                                          NUMBER(8)
 DEPTNO                                             NUMBER(2)
```

Note: Only structure of table alone is copied and not the contents.

- **Inserting into table using a subquery**

Syntax :

Insert into <new_table_name> (Select <columnnames> from <old_table_name>);

Example:

```
SQL> insert into copyofemp2 (select * from employee where employee_id > 100);
```

Output:

```
50 rows created.
```

Constraints

- Constraints enforce rules on the table whenever rows is inserted, updated and deleted from the table.
- Prevents the deletion of a table if there are dependencies from other tables.
- Name a constraints or the oracle server generate name by using SYS_cn format.
- Define the constraints at column or table level. constraints can be applied while creation of table or after the table creation by using alter command.
- View the created constraints from User_Constraints data dictionary.

Constraints Types

CONSTRAINT	**DESCRIPTION**
NOT NULL	Specifies that a column must have some value.
UNIQUE	Specifies that columns must have unique values.
PRIMARY KEY	Specifies a column or a set of columns that uniquely identifies as row. It does not allow null values.
FOREIGN KEY	Foreign key is a column(s) that references a column(s) of a table.
CHECK	Specifies a condition that must be satisfied by all the rows in a table.

1. Creating Constraints without constraint name

Syntax:

```
CREATE TABLE  < tablename>   (
<column name 1>  < datatype>,
<column name 2>  < datatype> UNIQUE ,
<column name 3>  < datatype> ,
 PRIMARY KEY ( <column name2>)
);
```

Example:

```
  CREATE TABLE emp_demo2
   ( employee_id    NUMBER(6) PRIMARY KEY,
    first_name     VARCHAR2(20) NOT NULL,
    last_name      VARCHAR2(25) NOT NULL,
    email          VARCHAR2(25) UNIQUE,
    phone_number   VARCHAR2(20) UNIQUE,
    job_id         VARCHAR2(10),
    salary         NUMBER(8,2) CHECK(SALARY>0),
    deptid  NUMBER(4)
   ) ;
```

2. Creating constraints with constraint name

Example:

```
     CREATE TABLE  < tablename1>   (
     <column name 1> < datatype> CONSTRAINT <constraint name1> UNIQUE,
     <column name 2> < datatype> CONSTRAINT <constraint name2> NOT NULL,
     constraint  < constraint name3 >  PRIMARY KEY ( <column name1>),
     constraint  <constraint name4>  FOREIGN KEY (<column name2>)
     REFERENCES  <tablename2>  (<column name1>)
     );
```

Example:

```
  CREATE TABLE emp_demo3
   ( employee_id    NUMBER(6) CONSTRAINT emp_eid PRIMARY KEY,
    first_name     VARCHAR2(20),
    last_name      VARCHAR2(25) CONSTRAINT emp_last_name_nn  NOT NULL,
    email          VARCHAR2(25) CONSTRAINT emp_email_nn  NOT NULL,
    phone_number   VARCHAR2(20),
    job_id         VARCHAR2(10) CONSTRAINT emp_job_nn  NOT NULL,
    salary         NUMBER(8,2) CONSTRAINT  emp_salary_nn  NOT NULL,
    deptid  NUMBER(4), CONSTRAINT emp_dept FOREIGN KEY(deptid)
     REFERENCES department(deptid)  ,
    CONSTRAINT     emp_salary_min CHECK (salary > 0) ,
    CONSTRAINT     emp_email_uk   UNIQUE (email)
      ) ;
```

3. With check constraint

Syntax:

```
CREATE TABLE  < tablename>   (
<column name1 >          < datatype> ,
 <column name 2>         < datatype>,
  CHECK  ( < column name 1  > in  ( values) )
CHECK  ( < column name 2  >  between <val1> and  <val2> ) );
```

Example:

```
CREATE TABLE emp_demo4
  ( emp_id     NUMBER(6),
   emp_name   VARCHAR2(15),
   salary     NUMBER(10)CHECK (salary between 1000 and 10000)
   );
```

Adding Constriants

Constraints can be added after the table creation by using alter command

Syntax: Add constraints

```
ALTER TABLE <tablename> ADD CONSTRAINT <constraint_name>  constriant_type
(<column name>);
```

Examples:

```
ALTER TABLE emp_demo4 ADD CONSTRAINT con_pk1  PRIMARY KEY(emp_id);

ALTER TABLE emp_demo4 ADD CONSTRAINT con_emp_uk  UNIQUE(phoneno);

ALTER TABLE emp_demo4 ADD CONSTRAINT con_empfk  FOREIGN KEY(DNO)
REFERENCES department(dno);

ALTER TABLE emp_demo4 ADD CONSTRAINT con_emp_ck CHECK  ( salary >0 );

ALTER TABLE emp_demo4 MODIFY (<Column name> <datatype> CONSTRAINT
constraint_name NOT NULL);
```

Drop Constraints

Syntax

ALTER TABLE <tablename> DROP CONSTRAINT < constraint name >;

Drop the unique key on the email column of the employees table:

e.g **ALTER TABLE** employees **DROP UNIQUE** (email);

CASCADE Constraints

The CASCADE Constraints clause is used along with the Drop Column Clause.

- A foreign key with a cascade delete means that if a record in the parent table is deleted, then the corresponding records in the child table will automatically be deleted. This is called a cascade delete.
- A foreign key with a cascade delete can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.

Syntax:

```
CREATE TABLE table_name
(column1 datatype null/not null,
column2 datatype null/not null,
...
CONSTRAINT fk_column
FOREIGN KEY (column1, column2, ... column_n)
REFERENCES parent_table (column1, column2, ... column_n)
ON DELETE CASCADE
);
```

Example:

```
CREATE TABLE supplier
(supplier_id number(10)not null,
supplier_namevarchar2(50)not null,
contact_namevarchar2(50),
CONSTRAINT supplier_pk PRIMARY KEY (supplier_id));
```

```
CREATE TABLE products
(product_id number(10)not null,
suppl_id number(10) not null,
CONSTRAINT fk_supplier FOREIGN KEY (suppl_id) REFERENCES
supplier(supplier_id) ON DELETE CASCADE);
```

Because of the cascade delete, when a record with a particular supplier_ id is deleted from supplier table ,then all the records of the same supplier_id will be deleted from products table also.

Operators in SQL*PLUS

Type	Symbol / Keyword	Where to use
Arithmetic	+ , - , * , /	To manipulate numerical column values, WHERE clause
Comparison	=, !=, <, <=, >, >=, between, not between, in, not in, like, not like	WHERE clause
Logical	and, or, not	WHERE clause, Combining two queries

- **Between**

Example:

```
SQL> select first_name, deptno from employee where salary between 20000 and 35000;
```

Output:

```
FIRST_NAME     DEPTNO
---------- ----------
Smith              10
```

- **IN**

Example:

```
SQL> select first_name, deptno from employee where job_id in ('J1','J2');
```

Output:

```
FIRST_NAME     DEPTNO
---------- ----------
Smith                  10
Arun                   30
Nithya                 10
```

- **NOT IN**

Example:

```
SQL> select dname,loc from department where loc not in ('chennai','Bangalore');
```

Output:

```
DNAME          LOC
-------------- -------------
finance        Hyderabad
```

- **Like**

Use the LIKE condition to perform wild card searches of valid search string values.

Search conditions can contain either characters or numbers

```
 %      -    denotes zero or many characters.

 _      -    denotes one character.
```

Example:

```
SQL> select dname,loc from department where loc like 'c%';
```

Output:

```
DNAME          LOC
-------------- -------------
accounts       chennai
marketing      Chennai
```

Example:

```
SQL> select dname,loc from department where loc like 'chen_ _ _';
```

Output:

```
DNAME          LOC
-------------- -------------
accounts       chennai
marketing      Chennai
```

Example:

```
SQL> select dname,loc from department where loc not like 'c%';
```

Output:

```
DNAME          LOC
-------------- -------------
finance        Hyderabad
IT             Bangalore
```

- **Between**

Example:

```
SQL> select first_name, deptno, salary from employee where salary not between 20000 and 35000;
```

Output:

```
FIRST_NAME     DEPTNO     SALARY
---------- ---------- ----------
Arun               30      40000
Nithya             10      45000
```

<u>**Note:**</u> Inserting null value into location column of department table

Example:

```
SQL> insert into department(deptno,dname) values(40,'Sales');
```

Output:

```
1 row created.
```

- **Is Null**

Example:

```
SQL> select * from department where loc is null;
```

Output:

```
   DEPTNO DNAME          LOC
---------- -------------- -------------
       40 Sales
```

Example:

```
SQL> select * from department where loc is not null;
```

Output:

```
   DEPTNO DNAME          LOC
---------- -------------- -------------
        10 accounts        chennai
        20 finance         Hyderabad
        30 IT              Bangalore
        40 marketing      chennai
```

LOGICAL OPERATORS: Used to combine the results of two or more conditions to produce a single result. The logical operators are: OR, AND, NOT.

Operator Precedence

- Arithmetic operators-Highest precedence
- Comparison operators
- NOT operator
- AND operator
- OR operator----Lowest precedence

The order of precedence can be altered using parenthesis.

Example:

```
SQL> select first_name, deptno, salary from employee where salary > 20000 ;
```

Output:

```
FIRST_NAME     DEPTNO     SALARY
---------- ---------- ----------
Smith              10      30000
Arun               30      40000
Nithya             10      45000
```

Example:

```
SQL> select first_name, deptno, salary from employee
where salary > 20000 and salary < 35000;
```

Output:

```
FIRST_NAME     DEPTNO     SALARY
---------- ---------- ----------
Smith              10      30000
```

Example:

SQL> select first_name, deptno, salary+100 from employee where salary > 35000;

Output:

```
FIRST_NAME     DEPTNO SALARY+100
---------- ---------- ----------
Arun               30      40100
```

Example:

SQL> update employee set salary = salary+salary*0.1 where employee_id = 111;

Output:

1 row updated.

Example:

SQL> select * from department where loc = 'chennai' or dname='IT';

Output:

```
    DEPTNO    DNAME          LOC
----------    -------------- -------------
        10    accounts       chennai
        30     IT            Bangalore
        40    marketing      chennai
```

<u>FUNCTIONS</u>

- Single Row Functions
- Group functions

<u>Single Row Functions</u>

Returns only one value for every row can be used in SELECT command and included in WHERE clause

Types

- Character functions
- Numeric functions
- Date functions

CHARACTER FUNCTIONS:

Character functions accept a character input and return either character or number values. Some of them supported by Oracle are listed below

Syntax	**Description**
initcap (char)	Changes first letter to capital
lower (char)	Changes to lower case
upper (char)	Changes to upper case
ltrim (char, set)	Removes the set from left of char
rtrim (char, set)	Removes the set from right of char
translate(char, from, to)	Translate ‘from’ anywhere in char to ‘to’
replace(char, search string, replace string)	Replaces the search string to new
substring(char, m , n)	Returns chars from m to n length
lpad(char, length, special char)	Pads special char to left of char to Max of length
rpad(char, length, special char)	Pads special char to right of char to Max of length
chr(number)	Returns char equivalent
length(char)	Length of string

Examples:

Function	**Input**	**Output**
Initcap(char)	SQL>select initcap(‘hello’) from dual;	Hello
Lower(char)	SQL>select lower(‘FUN’) from dual;	fun
Upper(char)	SQL>select upper(‘sun’) from dual;	SUN
Ltrim(char, set)	SQL>select ltrim(‘xyzhello’,’xyz’) from dual;	hello
Rtrim(char, set)	SQL>select rtrim(‘xyzhello’,’llo’) from dual;	xyzhe
translate(char,from,to)	SQL>select translate(‘jack’,’j’,’b’) from dual;	back
Replace(char,from,to)	SQL>select replace(‘jack and jue’,’ j’, ’bl’) from dual;	black and blue

Example:

SQL> select initcap(dname) from department;

Output:

```
INITCAP(DNAME)
-------------
Accounts
Finance
It
Marketing
Sales
```

Lpad is a function that takes three arguments. The first argument is the character string which has to be displayed with the left padding. The second is the number which indicates the total length of the return value, the third is the string with which the left padding has to be done when required.

Example:

SQL> select lpad(dname,15,'*') lpd from department;

Output:

```
LPD
---------------
*******accounts
********finance
*************IT
******marketing
**********Sales
```

Example:

```
SQL> select rpad(dname,15,'*') rpd from department;
```

Output:

```
RPD
---------------
accounts*******
finance********
IT*************
marketing******
Sales**********
```

Length: returns the length of a string

Example:

```
 SQL> select dname, length(dname) from department;
```

Output:

```
DNAME          LENGTH(DNAME)
-------------- -------------
accounts             8
finance              7
IT                   2
marketing            9
Sales                5
```

Concatenation || operator: is used to merge or more strings.

Example:

```
SQL> select dname || ' is located in ' || loc from department;
```

Output:

```
DNAME||'ISLOCATEDIN'||LOC
-----------------------------------------
accounts is located in chennai
finance is located in Hyderabad
IT is located in Bangalore
marketing is located in chennai
Sales is located in
```

NUMERIC FUNCTIONS:

Numeric functions accept numeric input and returns numeric values as output.

Syntax	**Description**
abs ( )	Returns the absolute value
ceil ( )	Rounds the argument
cos ( )	Cosine value of argument
exp ( )	Exponent value
floor( )	Truncated value
power (m,n)	N raised to m
mod (m,n)	Remainder of m / n
round (m,n)	Rounds m's decimal places to n
trunc (m,n)	Truncates m's decimal places to n
sqrt (m)	Square root value

Function	**Input**	**Output**
Abs(n)	SQL>select abs(-15) from dual	15
Ceil(n)	SQL>select ceil(48.778) from dual;	49
Cos(n)	SQL>select cos(180) from dual;	-0.59884601
Cosh(n):	SQL>select cosh(0) from dual;	1
Exp(n)	SQL>select exp(4) from dual;	54.59815
Floor(n)	SQL>select floor(4.678) from dual;	4
Power(m ,n)	SQL>select power(5,2) from dual;	25
Mod(m ,n)	SQL>select mod(11,2) from dual;	1
Round(m ,n)	SQL>select round(112.257,2) from dual;	112.26

Example:

SQL> select ln (2) from dual; (returns natural logarithm value of 2)

SQL>select sign (-35) from dual; (output is -1)

CONVERSION FUNCTIONS: Convert a value from one data type to another.

- **To__char ( )**

To_ char (d [,fmt]) where d is the date fmt is the format model which specifies the format of the date. This function converts date to a value of varchar2datatype in a form specified by date format fmt.if fmt is neglected then it converts date to varchar2 in the default date format.

Example:

SQL> select to_char (hire_date, 'ddth "of" fmmonth yyyy') from employee;

Output:

```
TO_CHAR(HIRE_DATE,'DDT
----------------------
21st of july 2010
05th of june 2008
12th of february 1999
```

- **To_ date ( )**

The format is to_date (char [, fmt]). This converts char or varchar data type to date data type. Format model, fmt specifies the form of character.

Example:

SQL>select to_date (‘December 18 2007’,’month-dd-yyyy’) from dual;

Output:

18-DEC-07 is the output.

Example:

SQL> select round(hire_date,'year') from employee;

Output:

```
ROUND(HIR
---------
01-JAN-11
01-JAN-08
01-JAN-99
```

- **To_ Number( )**

Allows the conversion of string containing numbers into the number data type on which arithmetic operations can be performed.

Example:

```
SQL> select to_number (‘100’) from dual;
```

DATE FUNCTIONS

Function Name	**Return Value**
ADD_MONTHS (date, n)	Returns a date value after adding *’n’*months to the date *’x’*.
MONTHS_BETWEEN (x1, x2)	Returns the number of months between dates x1 and x2.
ROUND (x, date_format)	Returns the date *‘x’*rounded off to the nearest century, year, month, date, hour, minute, or second as specified by the *‘date_format’*.
TRUNC (x, date_format)	Returns the date *‘x’* lesser than or equal to the nearest century, year, month, date, hour, minute, or second as specified by the ‘date_format’.
NEXT_DAY (x, week_day)	Returns the next date of the *‘week_day’*on or after the date *‘x’* occurs.
LAST_DAY (x)	It is used to determine the number of days remaining in a month from the date *'x'*specified.
SYSDATE	Returns the systems current date and time.

Example:

```
SQL> select sysdate from dual;
```

Output:

```
SYSDATE
---------
22-JUL-10
```

Example:

```
SQL> select hire_date from employee;
```

Output:

```
HIRE_DATE
---------
21-JUL-10
05-JUN-08
12-FEB-99
```

Example:

```
SQL> select add_months(hire_date,3) from employee;
```

Output:

```
ADD_MONTH
---------
21-OCT-10
05-SEP-08
12-MAY-99
```

Example:

```
SQL> select months_between(sysdate,hire_date) from employee;
```

Output:

```
MONTHS_BETWEEN(SYSDATE,HIRE_DATE)
---------------------------------
                       .047992085
                       25.5641211
                       137.338315
```

Example:

```
SQL> select next_day(hire_date,'wednesday') from employee;
```

Output:

```
NEXT_DAY(
---------
28-JUL-10
11-JUN-08
17-FEB-99
```

Example:

```
SQL> select last_day(hire_date) from employee;
```

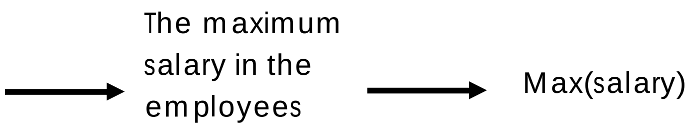
Output:

```
LAST_DAY(
---------
31-JUL-10
30-JUN-08
28-FEB-99
```

- **Group Functions:** -Result based on group of rows.

Group functions operate on sets of rows to give one result per group Employees

Dept_id	Salary
90	5000
90	10000
90	10000
60	5000
60	5000


The maximum salary in the employees

Max(salary)

Types of Group Functions

Syntax	**Description**
count (*), count (column name), count (distinct column name)	Returns number of rows
min (column name)	Min value in the column
max (column name)	Max value in the column
avg (column name)	Avg value in the column
sum (column name)	Sum of column values

Group Functions Syntax:

```
Select  [column,]  group_function(column),..
From     table
[where    condition]
[GROUP BY column];
```

Example:

Q.Display the average,highest, lowest and sum of salaries for all the sales representatives.

A. Select avg(salary), max(salary), min(salary), sum(salary) From employees where

job_id like ‘%rep%’;

Groups of Data :Divide rows in a table in to smaller groups by using the group by clause

Employee Table

Dept_id	Salary
10	4000
10	5000
10	6000
50	5000
50	3000

→ The average salary in employees table for each department →

D_id	Avg(Salary)
10	5000
50	4000

SET OPERATORS: UNION,UNION ALL,DIFFERENCE,MINUS

Example:

```
sql> select first_name from employees union select name from sample ;
```

Output:

```
FIRST_NAME
----------
DHANA
GUNA
JAI
JAISANKAR
KUMAR
RAJA
VENKAT
```

Example:

```
sql> select first_name from employees union all select name from sample ;
```

Output:

```
FIRST_NAME
----------
VENKAT
JAI
DHANA
GUNA
JAISANKAR
VENKAT
RAJA
KUMAR
```

Example:

```
sql>  select first_name from employees intersect select name from sample ;
```

Output:

```
FIRST_NAME
----------
VENKAT
```

Example:

```
sql>  select first_name from employees minus select name from sample ;
```

Output:

```
FIRST_NAME
----------
DHANA
GUNA
JAI
```

- **JOINS :**A join is the SQL way of combining the data from many tables. It is performed by WHERE Clause which combines the specified rows of the tables.

Type	**Sub type**	**Description**
Simple join	Equi join (=) Non – equi join (<, <=, >, >=, !=, < >)	Joins rows using equal value of the column Joins rows using other relational operators(except =)
Self join	-- (any relational operators)	Joins rows of same table
Outer join	Left outer join ((+) appended to left operand in join condition) Right outer join ((+) appended to right operand in join condition)	Rows common in both tables and uncommon rows have null value in left column Vice versa

Simple Join:

a. EQUI JOIN OR INNER JOIN : A column (or multiple columns) in two or more tables match.

Syntax:

```
SELECT <column_name(s)>
FROM <table_name1>
INNER JOIN <table_name2>
ON <table_name1.column_name>=<table_name2.column_name>;
```

Example 1 :

```
SELECT employee.first_name, department.dname
FROM employee INNER JOIN department
ON employee.deptno = department.deptno;
```

Output:1

```
DEPTNO FIRST_NAME
  ---------- ----------
     10        Smith
     30         Arun
     10         Nithya
```

Oracle automatically defaults the JOIN to INNER so that the INNER keyword is not required.

They are the same query, though. It is preferred not to type the INNER keyword.

Example 2 using where Condition:

```
SELECT employee.ename, department.dname
FROM employee JOIN department
ON employee.deptno = department.deptno
WHERE department.dname = 'SALES';
```

Output 2:

```
DEPTNO FIRST_NAME
   ---------- ----------
    10          Smith
    30          Arun
    10          Nithya
```

b. SELF JOIN : Is a join where a table is joined to itself.

Syntax:

```
SELECT <column_name(s)>
FROM <table_name1>
JOIN <table_name2>
ON <table_name1.column_name>=<table_name1.column_name>;
```

Example1:

```
SELECT e1.first_name, e2.first_name
```

```
FROM employee e1 join employee e2
on e1.mgr = e2.employee_id;
```

OR

```
SELECT e1.first_name, e2.first_name
FROM employee e1 join employee e2
where e1.mgr = e2.employee_id;
```

Output:

```
FIRST_NAME FIRST_NAME
----------------    --------------------
          john         john
```

An alias is just a way to refer to a column or table with a UNIQUE name. If we try to call both of the instances of the table EMP, Oracle wouldn't know which table instance I meant. Using an alias clears that right up.

c. OUTER JOIN

An **outer join** tells Oracle to return the rows on the **left or right** (of the JOIN clause) even if there are no rows.

The **LEFT OUTER** keyword to the JOIN clause says, return the rows to the left (in this case DEPARTMENT) even if there are no rows on the right (in this case employee).

Syntax:

```
SELECT <column_name(s)>
FROM <table_name1>
LEFT OUTER JOIN <table_name2>
ON <table_name1.column_name>=<table_name2.column_name>;
```

Example:

```
SELECT department.dname, employee.first_name
FROM department  LEFT OUTER JOIN employee
ON department.deptno = employee.deptno
WHERE department.dname = 'marketing';
```

Output:

```
DNAME         FIRST_NAME
--------------          ----------
Marketing
```

The **RIGHT OUTER** keyword to the JOIN clause says ,return the rows to the right (in this case DEPARTMENT) even if there are no rows on the left (in this case employee).

Syntax:

```
SELECT <column_name(s)>
FROM <table_name1>
RIGHT OUTER  JOIN <table_name2>
ON <table_name1.column_name>=<table_name2.column_name>;
```

Example:

```
SELECT employee.first_name, department.dname
FROM employee RIGHT OUTER JOIN department
ON employee.deptno = department.deptno
WHERE department.dname = 'marketing';
```

Output:

```
FIRST_NAME  DNAME
----------          --------------
                      marketing
```

d. FULL OUTER JOIN

Let's insert a new record into the employee table:

```
INSERT INTO EMPLOYEE (employee_id, first_name, last_name, mgr, hiredate, job-id,sal, comm, deptno) VALUES (9999, 'Joe ‘,’Blow', 7698, sysdate ,0008, 10500, 0, NULL );
```

Note:

We inserted an employee record that has no department. How can we get the records for all employees AND all departments? We would use **the FULL OUTER** join syntax:

Syntax:

```
SELECT <column_name(s)>
FROM <table_name1>
FULL OUTER  JOIN <table_name2>
ON <table_name1.column_name>=<table_name2.column_name>;
```

Example:

SELECT employee.first_name, department.dname

FROM employee FULL OUTER JOIN department

ON employee.deptno = department.deptno;

Output:

```
FIRST_NAME DNAME
----------         -------------
Nithya             accounts
Smith              accounts
                  finance
john              IT
Arun              IT
                  marketing
john
```

e.Cross Join

Displays all the rows and all the colums of both the tables.

Synatx:

SELECT <column_name(s)> FROM <table_name1> CROSS JOIN<table_name2>;

Example:

select employee.deptno from employee cross join department;
Or
select employee.deptno from employee,department;

Output:

```
   DEPTNO
----------
        10
        10
        10
        10
        30
        30
        30
        30
        10
        10
        10
    DEPTNO
----------
        10
        30
        30
        30
        30
```

f. Natural Join

If two tables have same column name the values of that column will be displayed only once.

Syntax:

SELECT <column_name(s)> FROM <table_name1> Natural JOIN<table_name2>;

Example:

select deptno,first_name from employee natural join department;

Output:

```
DEPTNO FIRST_NAME
------          ----------
    10          Smith
    30          Arun
    10          Nithya
    30          john
```

SUB QUERIES

- Nesting of queries
- A query containing a query in itself
- Inner most sub query will be executed first
- The result of the main query depends on the values return by sub query

- Sub query should be enclosed in parenthesis

1. Sub query returning only one value

a. Relational operator before sub query.

Syntax:

SELECT <column_name(s)> FROM <table_name> WHERE < column name >
< relational op.> < sub query>;

Example:

```
SELECT employee_id ,first_name FROM employee
WHERE deptno =
(SELECT deptno FROM department
WHERE dname = ‘IT’)
```

Output:

```
EMPLOYEE_ID FIRST_NAME
-----------          ----------
      112              Arun
      114               john
```

2. Sub query returning more than one value

a. ANY

For the clause any, the condition evaluates to true if there exists at least on row selected by the sub query for which the comparison holds. If the sub query yields an empty result set, the condition is not satisfied.

Syntax:

```
SELECT <column_name(s)>
FROM <table_name>
 WHERE < column name >
 < relational op.> ANY (<sub query>);
```

Example:

```
SELECT employee_id ,first_name FROM employee
WHERE salary>= ANY
(SELECT salary FROM employee
WHERE deptno = 30)
AND deptno = 10;
```

Output:

```
EMPLOYEE_ID FIRST_NAME
-----------          ----------
      113               Nithya
      112               Arun
      111               Smith
      114                john
      114                john
```

b. ALL

For the clause all, in contrast, the condition evaluates to true if for all rows selected by the sub query the comparison holds. In this case the condition evaluates to true if the Sub query does not yield any row or value.

Syntax:

```
SELECT <column_name(s)>
FROM <table_name>
WHERE < column name > < relational op.> ALL (<sub query>);
```

Example:

```
SELECT employee_id ,first_name FROM employee
WHERE salary > ALL
(SELECT salary FROM employee
WHERE deptno = 30);
```

Output:

```
EMPLOYEE_ID FIRST_NAME
-----------                ----------
       113                   Nithya
```

c. IN : Main query displays the values that match with any of the values returned by sub query.

Syntax:

SELECT <column_name(s)>
FROM <table_name>
WHERE < column name > IN (<sub query>);

Example:

```
SELECT employee_id ,first_name FROM employee
WHERE deptno IN
(SELECT deptno FROM department
WHERE loc = 'Bangalore');
```

Output:

```
EMPLOYEE_ID FIRST_NAME
-----------                  ----------
       114                   john
       112                   Arun
```

d. NOT IN

Main query displays the values that match with any of the values returned by sub query.

Syntax:

SELECT <column_name(s)>
FROM <table_name>
WHERE < column name > NOT IN (<sub query>);

Example:

```
SELECT employee_id ,first_name FROM employee
WHERE deptno NOT IN
(SELECT deptno FROM department
WHERE loc = ’Bangalore’);
```

Output:

```
EMPLOYEE_ID FIRST_NAME
-----------          ----------
      113            Nithya
      111            Smith
```

e. EXISTS

Main query displays the values that match with any of the values returned by sub query.

Syntax:

```
SELECT <column_name(s)>
FROM <table_name>
WHERE EXISTS (<sub query>);
```

Example:

```
SELECT *  FROM department

WHERE EXISTS

(SELECT  *  FROM employee

WHERE deptno = department.deptno);
```

Output:

```
   DEPTNO DNAME         LOC
   ---------- ------------- -------------
      10       accounts     chennai
      30        IT          Bangalore
```

f. NOT EXISTS

Main query displays the values that match with any of the values returned by sub query.

Syntax:

```
SELECT <column_name(s)>
FROM <table_name>
WHERE NOT EXISTS (<sub query>);
```

Example:

```
SELECT *  FROM department
WHERE NOT EXISTS
(SELECT  *  FROM employee
WHERE deptno = department.deptno);
```

Output:

```
   DEPTNO DNAME          LOC
    ---------- ------------- -------------
      20         finance       Hyderabad
      40       marketing      chennai
```

g. GROUP BY CLAUSE

Often applications require grouping rows that have certain properties and then applying an aggregate function on one column for each group separately. For this, SQL provides the clause group by <group column(s)>. This clause appears after the where clause and must refer to columns of tables listed in the from clause.

Rule:

Select attributes and group by clause attributes should be same.

Syntax:

```
        SELECT <column_name(s)>
        FROM <table_name>
        Where <conditions>
        GROUP BY <column2>, <column1>;
```

Example:

```
SELECT deptno, min(salary), max(salary)
FROM employee
GROUP BY deptno;
```

Output:

```
 DEPTNO MIN(SALARY) MAX(SALARY)
     ---------        -----------          -----------
       30            30000                  40000
                     30000                  30000
       10            33000                  45000
```

h. HAVING CLAUSE: used to apply a condition to group by clause

Syntax:

```
SELECT <column(s)>
FROM <table(s)>
WHERE <condition>
[GROUP BY <group column(s)>]
[HAVING <group condition(s)>];
```

Example:

```
SELECT deptno, min(salary), max(salary)
FROM employee
WHERE job_id = ’J2’
GROUP BY deptno
HAVING count(*) > 1;
```

Output:

```
   DEPTNO MIN(SALARY) MAX(SALARY)
----------      -----------               -----------
       30         13000                    40000
```

A query containing a group by clause is processed in the following way:

1. Select all rows that satisfy the condition specified in the where clause.

2. From these rows form groups according to the group by clause.

3. Discard all groups that do not satisfy the condition in the having clause.

4. Apply aggregate functions to each group.

5. Retrieve values for the columns and aggregations listed in the select clause.

i. ORDER BY

Used along with where clause to display the specified column in ascending

order or descending order .Default is ascending order

Syntax:

```
SELECT [distinct] <column(s)>
FROM <table>
[ WHERE <condition> ]
[ ORDER BY <column(s) [asc|desc]> ]
```

Example:

```
SELECT first_name, deptno, hire_date
FROM employee
ORDER BY deptno  ASC, hire_date desc;
```

Output:

```
FIRST_NAME     DEPTNO  HIRE_DATE
   ----------           ----------       ---------
     Smith                  10              21-JUL-10
     Nithya                 10              12-FEB-99
      john                  30              20-JAN-10
     Arun                   30              05-JUN-08
     john                                   20-JAN-10
```