What is SQL

- SQL stands for **Structured Query Language**.
- It is designed for managing data in a relational database management system (RDBMS).
- It is pronounced as S-Q-L or sometime **See-Qwell**.
- SQL is a database language, it is used for database creation, deletion, fetching rows, and modifying rows, etc.
- SQL is based on relational algebra and tuple relational calculus.

All DBMS like MySQL, Oracle, MS Access, Sybase, Informix, Postgres, and SQL Server use SQL as standard database language.

Why SQL is required

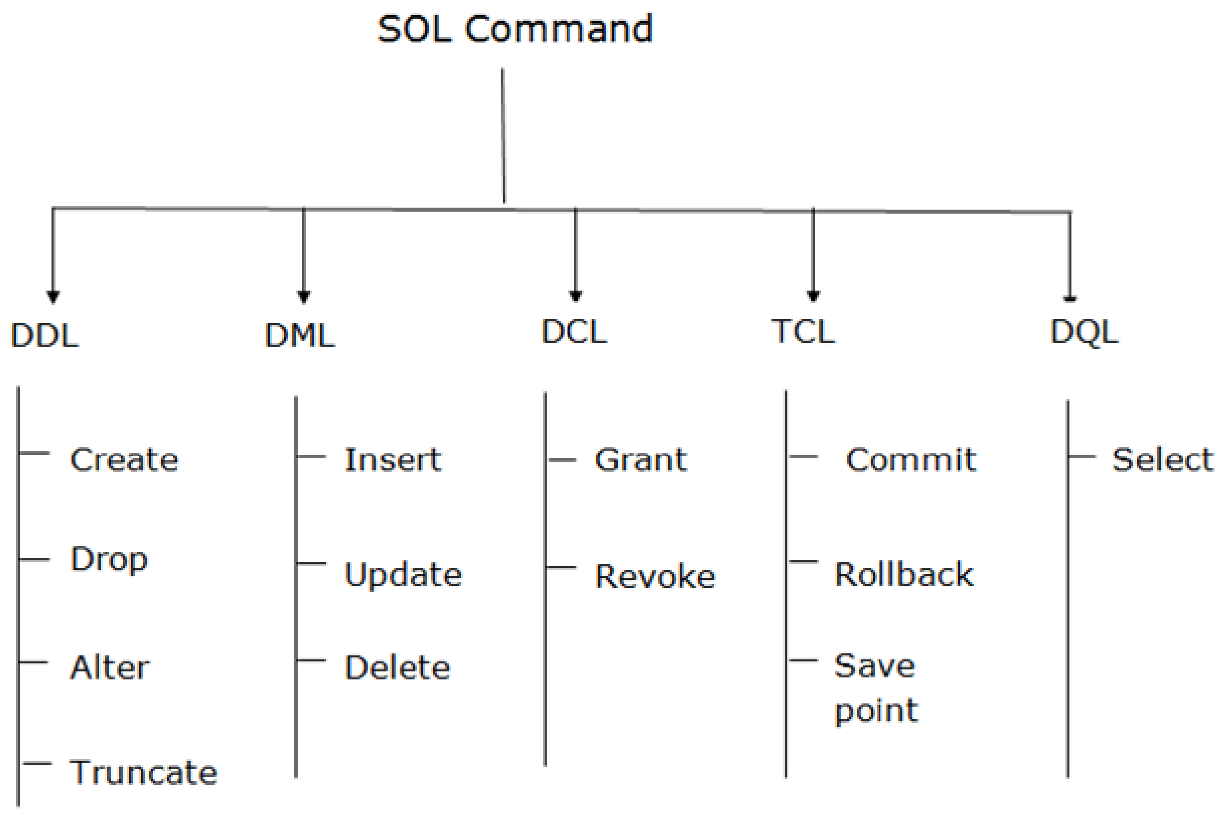
SQL is required:

- To create new databases, tables and views
- To insert records in a database
- To update records in a database
- To delete records from a database
- To retrieve data from a database

SQL command

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

Types of SQL Command:



Data types of SQL

1. NUMBER

Used to store a numeric value in a field/column. It may be decimal, integer or a real value. General syntax is:

Number(n,d)

Where **n** specifies the number of digits and

d specifies the number of digits to the right of the decimal point.

e.g marks number(3) declares marks to be of type number with maximum value 999.

pct number(5,2) declares pct to be of type number of 5 digits with two digits to the right of decimal point.

2. CHAR

Used to store character type data in a column. General syntax is

Char (size)

where size represents the maximum number of characters in a column. The CHAR type data can hold at most 255 characters.

e.g   name char(25) declares a data item name of type character of upto 25 size long.

3. VARCHAR/VARCHAR2

This data type is used to store variable length alphanumeric data. General syntax is,

varchar(size) / varchar2(size)

where size represents the maximum number of characters in a column. The maximum allowed size in this data type is 2000 characters.

e.g   address varchar(50);

address is of type varchar of upto 50 characters long.

4. DATE

Date data type is used to store dates in columns. SQL supports the various date formats other that the standard DD-MON-YY.

e.g   dob   date; declares dob to be of type date.

5. LONG

This data type is used to store variable length strings of upto 2 GB size.

e.g   description long;

1. Data definition language (DDL)

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE

a. CREATE It is used to create a new table in the database.

Syntax:

1. CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);

Example:

1. CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);

b. DROP: It is used to delete both the structure and record stored in the table.

Syntax

```
DROP object object_name
```

Example

1. DROP TABLE EMPLOYEE;

c. ALTER: It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

Syntax:

To add a new column in the table

1. ALTER TABLE table_name ADD column_name COLUMN-definition;

To modify existing column in the table:

1. ALTER TABLE table_name   MODIFY(COLUMN DEFINITION....);

EXAMPLE

1. ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));
2. ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));

d. TRUNCATE: It is used to delete all the rows from the table and free the space containing the table.

Syntax:

1. TRUNCATE TABLE table_name;

Example:

1. TRUNCATE TABLE EMPLOYEE;

DROP vs TRUNCATE

- Truncate is normally ultra-fast and its ideal for deleting data from a temporary table.
- Truncate preserves the structure of the table for future use, unlike drop table where the table is deleted with its full structure.

2. Data Manipulation Language

- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

a. INSERT: The INSERT statement is a SQL query. It is used to insert data into the row of a table.

Syntax:

1. INSERT INTO TABLE_NAME
2. (col1, col2, col3,.... col N)
3. VALUES (value1, value2, value3, ... valueN);

Or

1. INSERT INTO TABLE_NAME
2. VALUES (value1, value2, value3, ... valueN);

For example:

1. INSERT INTO Trident (Author, Subject) VALUES (‘Sumati’, ‘DBMS’);

b. UPDATE: This command is used to update or modify the value of a column in the table.

Syntax:

1. UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CON DITION]

For example:

1. UPDATE Trident
2. SET User_Name = 'TAT'
3. WHERE Student_Id = '3'

c. DELETE: It is used to remove one or more row from a table.

Syntax:

1. DELETE FROM table_name [WHERE condition];

For example:

1. DELETE FROM Trident
2. WHERE Author= ' Sumati';

3. Data Control Language

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

- Grant
- Revoke

a. Grant: It is used to give user access privileges to a database.

Example

1. GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

b. Revoke: It is used to take back permissions from the user.

Example

1. REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

4. Transaction Control Language

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

- COMMIT
- ROLLBACK
- SAVEPOINT

a. Commit: Commit command is used to save all the transactions to the database.

Syntax:

```
1. COMMIT;
```

Example:

```
1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. COMMIT;
```

b. Rollback: Rollback command is used to undo transactions that have not already been saved to the database.

Syntax:

```
1. ROLLBACK;
```

Example:

```
1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. ROLLBACK;
```

c. SAVEPOINT: It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax:

```
1. SAVEPOINT SAVEPOINT_NAME;
```

5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

- SELECT

a. SELECT: This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

Syntax:

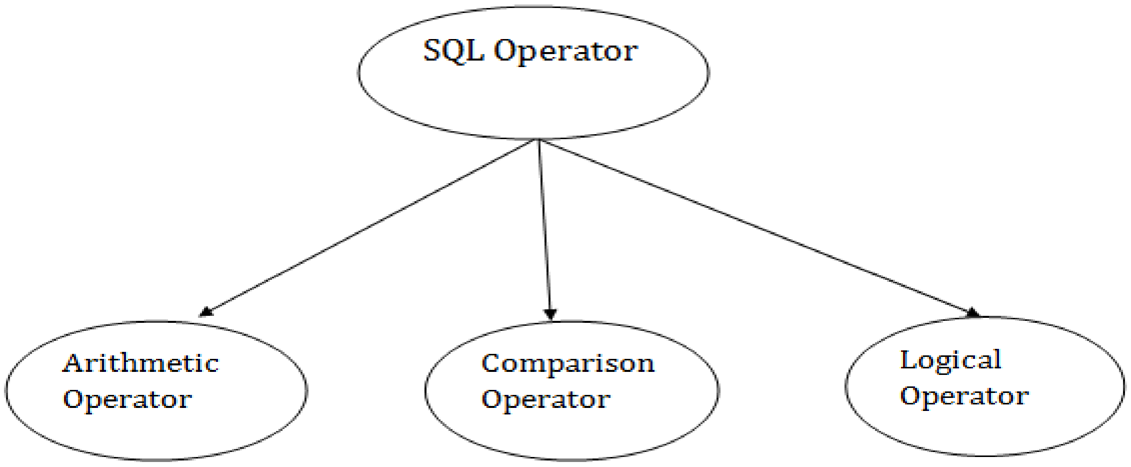
1. SELECT expressions
2. FROM TABLES
3. WHERE conditions;

For example:

1. SELECT emp_name
2. FROM employee
3. WHERE age > 20;

SQL Operator

There are various types of SQL operator:



SQL Arithmetic Operators

Operator	Description	Example a=20, b=10
+	It adds the value of both operands.	a+b will give 30
-	It is used to subtract the right-hand operand from the left-hand operand.	a-b will give 10
*	It is used to multiply the value of both operands.	a*b will give 200

/	It is used to divide the left-hand operand by the right-hand operand.	a/b will give 2
%	It is used to divide the left-hand operand by the right-hand operand and returns reminder.	a%b will give 0

SQL Comparison Operators:

Let's assume 'variable a' and 'variable b'. Here, 'a' contains 20 and 'b' contains 10.

Operator	Description
=	It checks if two operands values are equal or not, if the values are equal then condition becomes true.
!=	It checks if two operands values are equal or not, if values are not equal, then condition becomes true.
<>	It checks if two operands values are equal or not, if values are not equal then cor becomes true.
>	It checks if the left operand value is greater than right operand value, if yes then becomes true.
<	It checks if the left operand value is less than right operand value, if yes then condition becomes true.
>=	It checks if the left operand value is greater than or equal to the right operand value, if yes then condition becomes true.
<=	It checks if the left operand value is less than or equal to the right operand value then condition becomes true.

SQL Logical Operators

There is the list of logical operator used in SQL:

Operator	Description
AND	It allows the existence of multiple conditions in an SQL statement.
BETWEEN	It is used to search for values that are within a set of values.
IN	It compares a value to that specified list value.
NOT	It reverses the meaning of any logical operator.
OR	It combines multiple conditions in SQL statements.
EXISTS	It is used to search for the presence of a row in a specified table.
LIKE	It compares a value to similar values using wildcard operator.

The SQL LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

- % - The percent sign represents zero, one, or multiple characters
- _ - The underscore represents a single character

Selects all customers with a CustomerName starting with "a":

Example

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

selects all customers with a CustomerName that have "or" in any position:

Example

-
  ```
  SELECT * FROM Customers
  WHERE CustomerName LIKE '%or%';
  ```

  selects all customers with a CustomerName that starts with "a" and are at least 3 characters in length:

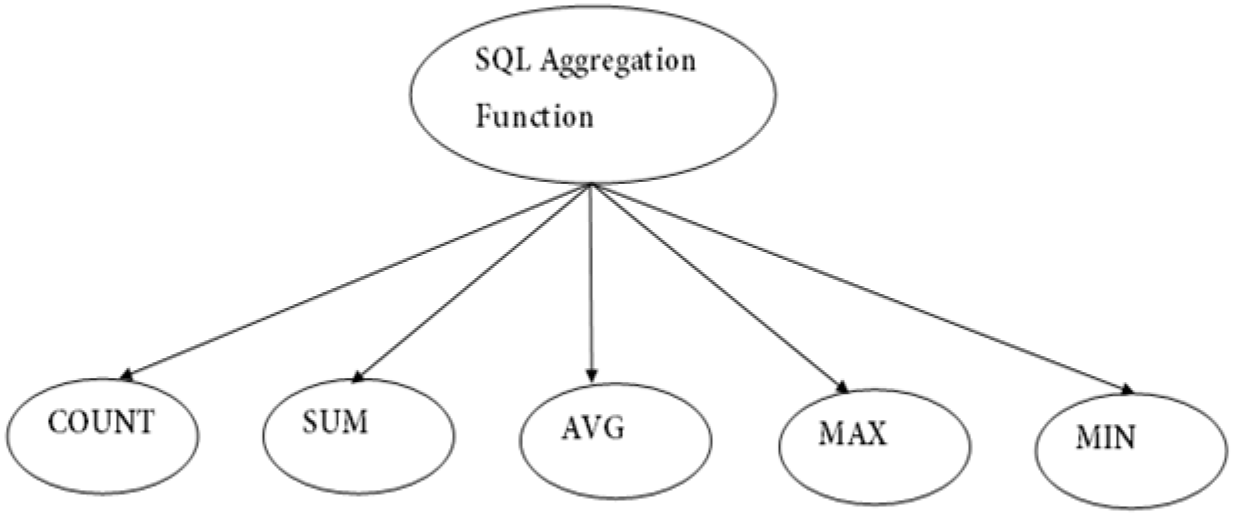
Example

-
  ```
  SELECT * FROM Customers
  WHERE CustomerName LIKE 'a__%';
  ```

SQL Aggregate Functions

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

Types of SQL Aggregation Function



COUNT FUNCTION

- COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.
- COUNT function uses the COUNT(*) that returns the count of all the rows in a specified table. COUNT(*) considers duplicate and Null.

Example: COUNT()

1. SELECT COUNT(*)
2. FROM PRODUCT_MAST;

2. SUM Function

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

Example: SUM()

1. SELECT SUM(COST)
2. FROM PRODUCT_MAST;

AVG function

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

```
SELECT AVG(COST)
FROM PRODUCT_MAST;
```

4. MAX Function

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

```
SELECT MAX(RATE)
FROM PRODUCT_MAST;
```

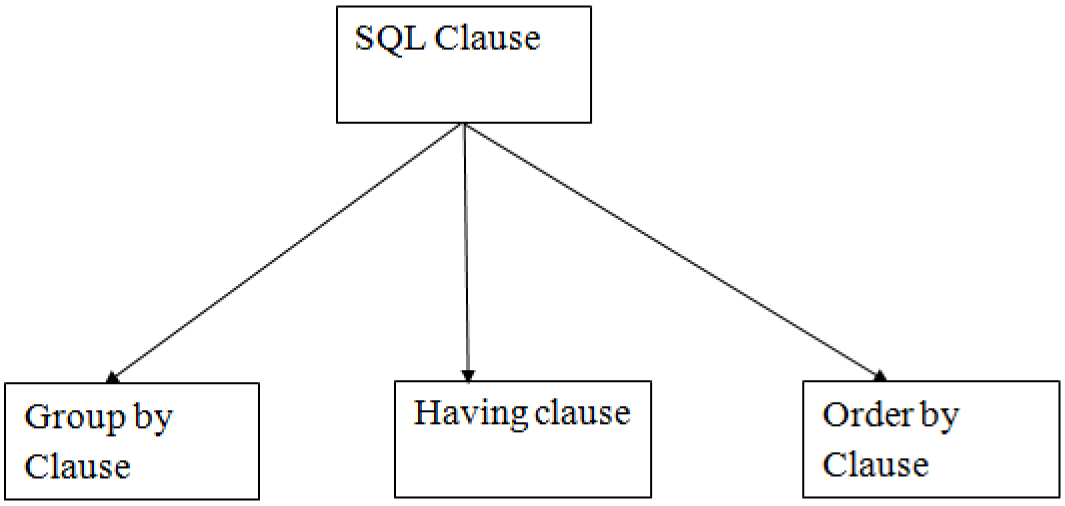
5. MIN Function

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

```
SELECT MIN(RATE)
FROM PRODUCT_MAST;
```

SQL Clauses

The following are the various SQL clauses:



1. GROUP BY

- SQL GROUP BY statement is used to arrange identical data into groups. The GROUP BY statement is used with the SQL SELECT statement.
- The GROUP BY statement follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.
- The GROUP BY statement is used with aggregation function.

Syntax

1. SELECT column
2. FROM table_name
3. WHERE conditions
4. GROUP BY column
5. ORDER BY column

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Example:

```
SELECT COMPANY, COUNT(*)
FROM PRODUCT_MAST
GROUP BY COMPANY;
```

Output:

```
Com1   5
Com2   3
Com3   2
```

2. HAVING

- HAVING clause is used to specify a search condition for a group or an aggregate.
- Having is used in a GROUP BY clause. If you are not using GROUP BY clause then you can use HAVING function like a WHERE clause.

Syntax:

```
SELECT column1, column2
FROM table_name
WHERE conditions
GROUP BY column1, column2
HAVING conditions
ORDER BY column1, column2;
```

Example:

```
SELECT COMPANY, COUNT(*)
FROM PRODUCT_MAST
GROUP BY COMPANY
HAVING COUNT(*)>2;
```

Output:

```
Com1   5
Com2   3
```

3. ORDER BY

- The ORDER BY clause sorts the result-set in ascending or descending order.
- It sorts the records in ascending order by default. DESC keyword is used to sort the records in descending order.

Syntax:

```
SELECT column1, column2
FROM table_name
WHERE condition
ORDER BY column1, column2... ASC|DESC;
```

SQL Sub Query

A Subquery is a query within another SQL query and embedded within the WHERE clause

Subqueries with the Select Statement

Syntax

```
SELECT column_name
FROM table_name
WHERE column_name expression operator
```

```
( SELECT column_name  from table_name WHERE ... );
```

```
SELECT *
    FROM EMPLOYEE
    WHERE ID IN (SELECT ID
    FROM EMPLOYEE
    WHERE SALARY > 4500);
```

Subqueries with the UPDATE Statement

```
UPDATE EMPLOYEE
   SET SALARY = SALARY * 0.25
   WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
      WHERE AGE >= 29);
```

Subqueries with the INSERT Statement

```
INSERT INTO EMPLOYEE_BKP
   SELECT * FROM EMPLOYEE
   WHERE ID IN (SELECT ID
   FROM EMPLOYEE);
```

Subqueries with the DELETE Statement

```
DELETE FROM EMPLOYEE
   WHERE AGE IN (SELECT AGE FROM EMPLOYEE_BKP
      WHERE AGE >= 29 );
```

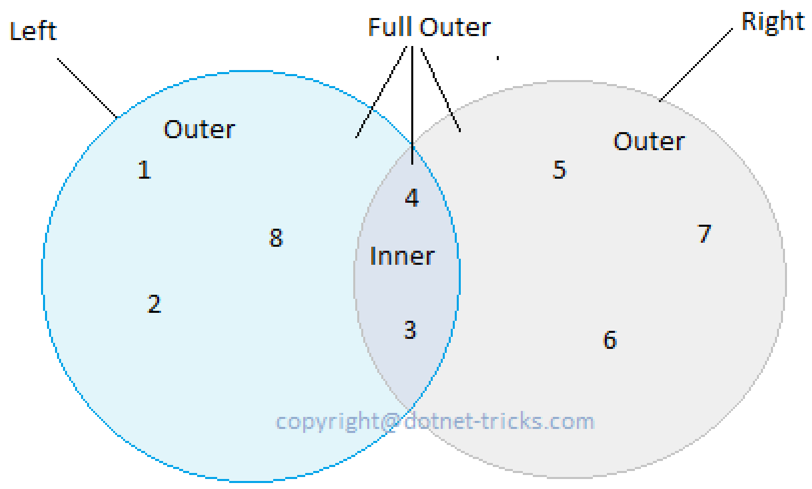
SQL JOIN

As the name shows, JOIN means to combine something. In case of SQL, JOIN means "to combine two or more tables".

In SQL, JOIN clause is used to combine the records from two or more tables in a database.

Types of SQL JOIN

1. INNER JOIN
2. LEFT JOIN
3. RIGHT JOIN
4. FULL JOIN
5. Self Join



Inner Join Result : (4,3)
Left Join Result : (1,2,8,4,3)
Right Join Result : (5,6,7,4,3)
Full Outer Join Result : (1,2,8,4,3,5,6,7)

Sample Table

EMPLOYEE

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36

6	Marry	Canada	600000	48

PROJECT

PROJECT_NO	EMP_ID	DEPARTMENT
101	1	Testing
102	2	Development
103	3	Designing
104	4	Development

1. INNER JOIN

In SQL, INNER JOIN selects records that have matching values in both tables as long as the condition is satisfied. It returns the combination of all rows from both the tables where the condition satisfies.

Syntax

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
INNER JOIN table2
ON table1.matching_column = table2.matching_column;
```

Query

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
INNER JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development

2. LEFT JOIN

The SQL left join returns all the values from left table and the matching values from the right t
value, it will return NULL.

Syntax

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
```

Query

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
LEFT JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output

EMP_NAME	DEPARTMENT
Angelina	Testing

Robert	Development
Christian	Designing
Kristen	Development
Russell	NULL
Marry	NULL

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
```

3. RIGHT JOIN

In SQL, RIGHT JOIN returns all the values from the values from the rows of right table and the

If there is no matching in both tables, it will return NULL.

Query

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
RIGHT JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

4. FULL JOIN

In SQL, FULL JOIN is the result of a combination of both left and right outer join. Join tables have all the records from both tables. It puts NULL on the place of matches not found.

Syntax

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
FULL JOIN table2
ON table1.matching_column = table2.matching_column;
```

Query

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
FULL JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

5. SELF JOIN

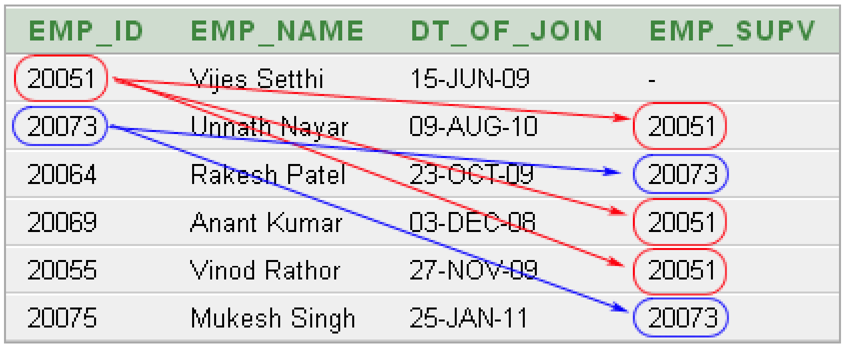
A self join is a join in which a table is joined with itself (which is also called Unary relationships), especially when the table has a FOREIGN KEY which references its own PRIMARY KEY. To join a table itself means that each row of the table is combined with itself and with every other row of the table.

syntax :

SELECT a.column_name, b.column_name...

FROM table1 a, table1 b

WHERE a.common_filed = b.common_field;



EMP_ID	EMP_NAME	DT_OF_JOIN	EMP_SUPV
20051	Vijes Setthi	15-JUN-09	-
20073	Unnath Nayar	09-AUG-10	20051
20064	Rakesh Patel	23-OCT-09	20073
20069	Anant Kumar	03-DEC-08	20051
20055	Vinod Rathor	27-NOV-09	20051
20075	Mukesh Singh	25-JAN-11	20073

How the employees are related to themselves:

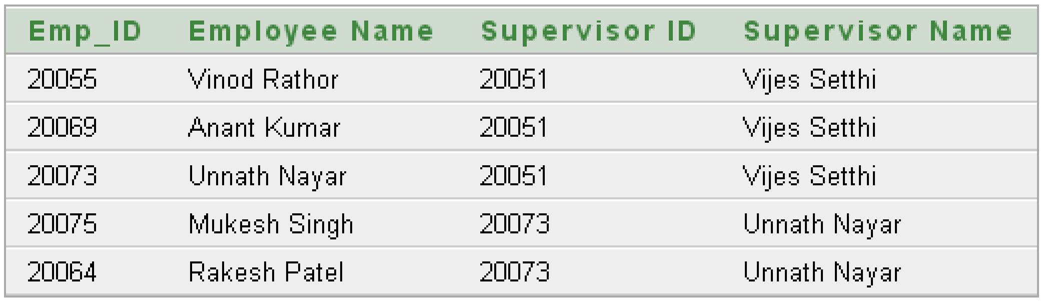
- An employee may report to another employee (supervisor).

The above data shows:

- Unnath Nayar's supervisor is Vijes Setthi
- Anant Kumar and Vinod Rathor can also report to Vijes Setthi.
- Rakesh Patel and Mukesh Singh are under supervison of Unnith Nayar.

To get the list of employees and their supervisor the following SQL statement has used:

```
SELECT a.emp_id AS "Emp_ID",a.emp_name AS "Employee Name",
b.emp_id AS "Supervisor ID",b.emp_name AS "Supervisor Name"
FROM employee a, employee b
  WHERE a.emp_supv = b.emp_id;
```



Emp_ID	Employee Name	Supervisor ID	Supervisor Name
20055	Vinod Rathor	20051	Vijes Setthi
20069	Anant Kumar	20051	Vijes Setthi
20073	Unnath Nayar	20051	Vijes Setthi
20075	Mukesh Singh	20073	Unnath Nayar
20064	Rakesh Patel	20073	Unnath Nayar

Views in SQL

- Views in SQL are considered as a virtual table. A view also contains rows and columns.
- To create the view, we can select the fields from one or more tables present in the database.
- A view can either have specific rows based on certain condition or all the rows of a table.

Creating view

A view can be created using the **CREATE VIEW** statement. We can create a view from a single table or multiple tables.

Syntax:

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

2. Creating View from a single table

In this example, we create a View named DetailsView from the table Student_Detail.

Query:

```
CREATE VIEW DetailsView AS
SELECT NAME, ADDRESS
FROM Student_Details
WHERE STU_ID < 4;
```

3. Creating View from multiple tables

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

In the given example, a view is created named MarksView from two tables Student_Detail and Student_Marks.

Query:

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

4. Deleting View

A view can be deleted using the Drop View statement.

Syntax

```
DROP VIEW view_name;
```

Definition of Materialized View

Materialized View is the **Physical copy** of the original base tables. The Materialized View is like a **snapshot** or **picture** of the original base tables. Like View, it also contains the data retrieved from the **query expression** of **Create Materialized View** command.

BASIS FOR COMPARISON	VIEW	MATERIALIZED VIEW
Basic	A View is never stored it	A Materialized View is stored

BASIS FOR COMPARISON	VIEW	MATERIALIZED VIEW
	is only displayed.	on the disk.
Define	View is the virtual table formed from one or more base tables or views.	Materialized view is a physical copy of the base table.
Update	View is updated each time the virtual table (View) is used.	Materialized View has to be updated manually or using triggers.
Speed	Slow processing.	Fast processing.
Memory usage	View do not require memory space.	Materialized View utilizes memory space.
Syntax	Create View V As	Create Materialized View V Build [clause] Refresh

BASIS FOR COMPARISON	VIEW	MATERIALIZED VIEW
		[clause] On [Trigger] As

Example:

```
CREATE MATERIALIZED VIEW MV_Employee BUILD immediate
REFRESH complete
on commit SELECT * FROM Employee;
```

SQL Index

- Indexes are special lookup tables. It is used to retrieve data from the database very fast.
- An Index is used to speed up select queries and where clauses. But it shows down the data input with insert and update statements. Indexes can be created or dropped without affecting the data.
- An index in a database is just like an index in the back of a book.
- **For example:** When you reference all pages in a book that discusses a certain topic, you first have to refer to the index, which alphabetically lists all the topics and then referred to one or more specific page numbers.

1. Create Index statement

It is used to create an index on a table. It allows duplicate value.

Syntax

1. CREATE INDEX index_name
2. ON table_name (column1, column2, ...);

2. Unique Index statement

It is used to create a unique index on a table. It does not allow duplicate value.

Syntax

1. CREATE UNIQUE INDEX index_name
2. ON table_name (column1, column2, ...);

3. Drop Index Statement

It is used to delete an index in a table.

Syntax

1. DROP INDEX index_name;

Procedures & Functions

"A **procedures** or **function** is a group or set of SQL and PL/SQL statements that perform a specific task."
A function and procedure is a named PL/SQL Block which is similar . The major difference between a procedure and a function is, a function must always return a value, but a procedure may or may not return a value.

Procedures:

A procudure is a named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages. A procedure has a header and a body.
The **header** consists of the name of the procedure and the parameters or variables passed to the procedure.
The **body** consists or declaration section, execution section and exception section similar to a general PL/SQL Block. A procedure is similar to an anonymous PL/SQL Block but it is named for repeated usage.
We can pass parameters to procedures in three ways :

Parameters	Description
IN type	These types of parameters are used to send values to stored procedures.
OUT type	These types of parameters are used to get values from stored procedures. This is

	similar to a return type in functions.
IN OUT type	These types of parameters are used to send values and get values from stored procedures.

A procedure may or may not return any value.

Syntax:

```
CREATE [OR REPLACE] PROCEDURE procedure_name (<Argument> {IN, OUT, IN
OUT}   <Datatype>,…)
IS
  Declaration section<variable, constant> ;
BEGIN
  Execution section
EXCEPTION
  Exception section
END
```

IS - marks the beginning of the body of the procedure and is similar to DECLARE in anonymous PL/SQL Blocks. The code between IS and BEGIN forms the Declaration section. The syntax within the brackets [ ] indicate they are optional. By using CREATE OR REPLACE together the procedure is created if no other procedure with the same name exists or the existing procedure is replaced with the current code.

How to execute a Procedure?

There are two ways to execute a procedure :

- From the SQL prompt : **EXECUTE** [or EXEC] procedure_name;
- Within another procedure – simply use the procedure name : procedure_name;

Example:

create table named emp have two column id and salary with number datatype.

```
CREATE OR REPLACE PROCEDURE p1(id IN NUMBER, sal IN NUMBER)
AS
BEGIN
  INSERT INTO emp VALUES(id, sal);
```

```
 DBMD_OUTPUT.PUT_LINE('VALUE INSERTED.');
END;
/
```

Output:

Functions:

A function is a named PL/SQL Block which is similar to a procedure. The major difference between a procedure and a function is, a function must always return a value, but a procedure may or may not return a value.

Syntax:

```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
 RETURN return_datatype;  {IS, AS}
 Declaration_section <variable,constant> ;
 BEGIN
    Execution_section
    Return return_variable;
 EXCEPTION
   exception section
    Return return_variable;
 END;
```

RETURN TYPE: The header section defines the return type of the function. The return datatype can be any of the oracle datatype like varchar, number etc.
The execution and exception section both should return a value which is of the datatype defined in the header section.

How to execute a Function?
A function can be executed in the following ways.

- As a part of a SELECT statement : SELECT emp_details_func FROM dual;
- In a PL/SQL Statements like, : dbms_output.put_line(emp_details_func);

This line displays the value returned by the function .

Example:

```
create or replace function getsal (no IN number) return number
is
 sal number(5);
begin
 select salary into sal from emp where id=no;
 return sal;
end;
/
```

In the example we are retrieving the 'salary' of employee with id 2 to variable 'sal'.
The return type of the function is number.

Destroying procedure and function :

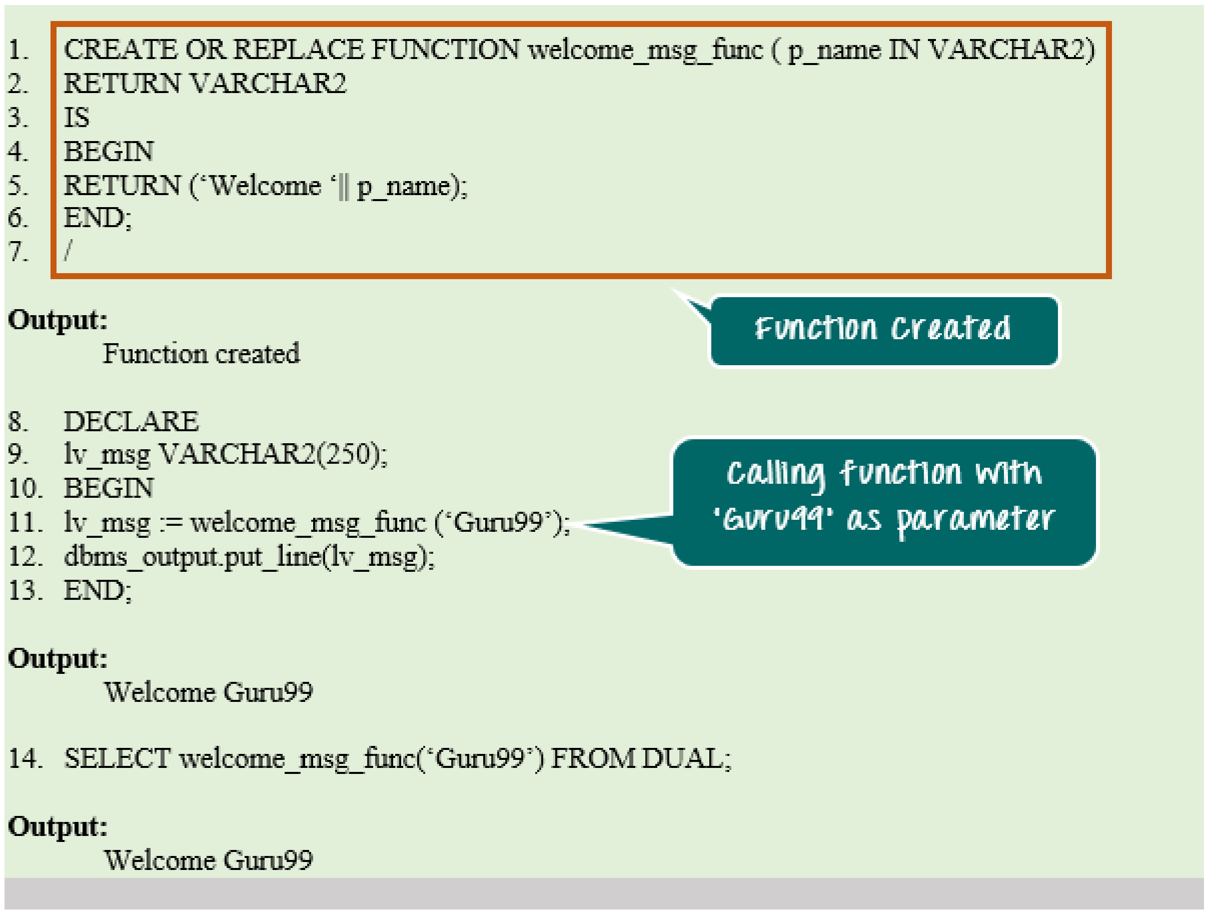
Syntax:

```
DROP PROCEDURE/FUNCTION PROCEDURE/FUNCTION_NAME;
```

Procedures VS Functions:

- A function MUST return a value
- A procedure cannot return a value
- The return statement of a procedure returns control to the calling program and cannot return a value
- Functions can be called from SQL, procedure cannot

What is Function?



```
CREATE OR REPLACE FUNCTION welcome_msg_func ( p_name IN VARCHAR2)
RETURN VARCHAR2
IS
BEGIN
RETURN (‘Welcome ‘|| p_name);
END;
/
```

Output:

```
        Function created
```

```
DECLARE
lv_msg VARCHAR2(250);
BEGIN
lv_msg := welcome_msg_func (‘Guru99’);
dbms_output.put_line(lv_msg);
END;
```

Output:

```
        Welcome Guru99
```

```
SELECT welcome_msg_func(‘Guru99’) FROM DUAL;
```

Output:

```
        Welcome Guru99
```

Trigger:

A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

Syntax:

```
create trigger [trigger_name]

[before | after]

{insert | update | delete}

on [table_name]

[for each row]

[trigger_body]
```

1. create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.
2. [before | after]: This specifies when the trigger will be executed.
3. {insert | update | delete}: This specifies the DML operation.
4. on [table_name]: This specifies the name of the table associated with the trigger.
5. [for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.

6. [trigger_body]: This provides the operation to be performed as trigger is fired

```
create trigger stud_marks

before INSERT

on

Student

for each row

set Student.total = Student.subj1 + Student.subj2 + Student.subj3,
Student.per = Student.total * 60 / 100;
```

Above SQL statement will create a trigger in the student database in which whenever subjects marks are entered, before inserting this data into the database, trigger will compute those two values and insert with the entered values. i.e.,

```
insert into Student values(0, "ABCDE", 20, 20, 20, 0, 0);

Query OK, 1 row affected (0.09 sec)


sql> select * from Student;

+-----+-------+-------+-------+-------+-------+------+

| tid | name  | subj1 | subj2 | subj3 | total | per  |

+-----+-------+-------+-------+-------+-------+------+

| 100 | ABCDE |    20 |    20 |    20 |    60 |   36 |

+-----+-------+-------+-------+-------+-------+------+
```

EXCEPTION

An exception is an error condition during a program execution. PL/SQL supports programmers to catch such conditions using **EXCEPTION** block in the program and an appropriate action is taken against the error condition. There are two types of exceptions −

- System-defined exceptions
- User-defined exceptions

Syntax for Exception Handling

The general syntax for exception handling is as follows. Here you can list down as many exceptions as you can handle. The default exception will be handled using ***WHEN others THEN*** –

```
DECLARE
   <declarations section>
BEGIN
   <executable command(s)>
EXCEPTION
   <exception handling goes here >
   WHEN exception1 THEN
      exception1-handling-statements
   WHEN exception2  THEN
      exception2-handling-statements
   WHEN exception3 THEN
      exception3-handling-statements
   ........
   WHEN others THEN
      exception3-handling-statements
END;
```

Example

Let us write a code to illustrate the concept. We will be using the CUSTOMERS table we had created and used in the previous chapters –

```
DECLARE
   c_id customers.id%type := 8;
   c_name customerS.Name%type;
   c_addr customers.address%type;
BEGIN
   SELECT  name, address INTO  c_name, c_addr
   FROM customers
   WHERE id = c_id;
   DBMS_OUTPUT.PUT_LINE ('Name: '||  c_name);
   DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);

EXCEPTION
   WHEN no_data_found THEN
      dbms_output.put_line('No such customer!');
   WHEN others THEN
      dbms_output.put_line('Error!');
END;
```