Order By:

Order by clause allows sorting of query results by one or more columns.

Syntax : Select <colname>… from <tablename> where condition order by <colname>.

select \* from emp where e\_sal>23000 order by e\_name;

select \* from emp order by e\_desig , e\_name;

Group by:

Group by clause combines all those records that have identical values in a particular field. Grouping can be done with aggregate functions.

Syntax: Select <colname> , agg\_func from <tablename> group by <colname>

select e\_desig, count(\*) from emp group by e\_desig;

Having clause:

Having clause places condition on groups in constrast to where clause that places conditions on individual rows.

Syntax: Select <colname> , agg\_func from <tablename> group by <colname> having <condition>

select e\_desig, count(\*) from emp group by e\_desig having count(\*)>=2;

Subquery:

select \* from emp where e\_id in (select e\_id from emp where e\_sal > 25000);

Joins:

A join is a query that combines rows from two or more tables.

**Cartesian join/cross join**: joins two tables

Syntax: select \* from <table1><table2>

select \* from emp, dept;   
products: rows1\*rows2= result.

**Equi join** : performs joins against matching columns values of the associated table.

**Syntax**1: select <col\_list> from table 1, table 2 where table1.column\_name = table2.column\_name;

select \* from emp, dept where emp.e\_id = dept.e\_id;

**Syntax**2: select <col\_list> from table 1 join table 2 on <join condition>;

select \* from emp join dept on emp.e\_id= dept.e\_id;

**Natural join**: is a type of join and is structured in such a way that columns with the same name of associated tables will appear only once.

Syntax: select \* from table1 natural join table2;

**Inner join**: Matches rows of one table with the other table

Syntax: select \* from table1 inner join table2 on <join condition>

select \* from emp inner join dept on emp.e\_id = dept.e\_id;

**Left join**: results in all matching rows from the two tables and all the rows from the left table.

Syntax: Select \* from left table left join right table on <join condition>

select \* from emp left join dept on emp.e\_id = dept.e\_id;

**Right join**: results in all matching rows from the two tables and all the rows from the right table.

Syntax: Select \* from left table right join right table on <join condition>

select \* from emp right join dept on emp.e\_id = dept.e\_id;

Self Join: Joining a table with itself.

Stored Procedures:

A procedure (often called a stored procedure) is a **collection of pre-compiled SQL statements** stored inside the database. So that we can call and reuse it anytime we want to.

Advantages:

* Stored Procedure increases the performance of the applications. Once stored procedures are created, they are compiled and stored in the database.
* Stored procedure reduces the traffic between application and database server. Because the application has to send only the stored procedure's name and parameters instead of sending multiple SQL statements.
* Stored procedures are reusable and transparent to any applications.
* A procedure is always secure. The database administrator can grant permissions to applications that access stored procedures in the database without giving any permissions on the database tables.

How to create a stored procedures.

delimiter &&  
create procedure getEmp()

begin

select \* from emp;

end &&

call getEmp();

**with parameter.**

delimiter &&  
create procedure getEmpDet(emp\_id int)

begin

select \* from emp where e\_id = emp\_id;

end &&

call getEmpDet(100);

With in and out parameters.

delimiter &&

mysql> create procedure get(in id int, out designation varchar(20))

-> begin

-> select e\_desig into designation from emp where e\_id = id;

-> end &&

TO CALL:

set @id=100;

call get(@id , @designation);

select @designation;

There is inout, go through that.

Triggers:

Triggers are sql codes that are automatically executed in response to certain events on a particular table .

Insert trigger before:

create trigger insert

-> before insert on emp1

-> for each row

-> set new.ename = upper(new.ename);

Update trigger before:

create trigger update

-> before update on emp1

-> for each row

-> set new.ename = lower(new.ename);