1. select \* from Customers;
2. select CustomerName, Address from Customers;
3. select \* from  Customers where  CustomerName = 'pratik' and  Address = 'abc';
4. select \* from Customers where CustomerID = 8 and Country = 'Spain' and City = 'Madrid'; (DONT GIVE COMMA ALWAYS USE AND)

The AND operator displays a record if all the conditions separated by AND are TRUE.

The OR operator displays a record if any of the conditions separated by OR is TRUE.

1. select \* from Customers where CustomerID = 8 or Country = 'Spain' and City = 'Madrid';
2. select PostalCode, City from Customers order by CustomerID asc;
3. select PostalCode, City from Customers order by PostalCode asc;
4. insert into Customers (customerID, contactName) values (1001,'Prt');
5. select **max**(CustomerID) from Customers where contactName = 'Prt';
6. select **min**(CustomerID) from Customers ;
7. select **count**(CustomerID) from customers;
8. select **avg**(customerID) from customers;
9. 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

1. select city from customers where city like 'B%';
2. select ContactName from customers where ContactName like 'a%';
3. select \* from Customers where city **like** 'b%';
4. select \* from Customers where city **not** **in** ('Berlin', 'Bern');
5. select \* from Customers where city **in** ('Berlin', 'Bern');
6. select \* from products where price **between** 10 and 20; -- here 10 and 20 both inclusive
7. **INNER JOIN** :  
   This type of join returns those records which have matching values in both tables. So, if you perform an INNER join operation between the Employee table and the Projects table, all the tuples which have matching values in both the tables will be given as output.

select Customers.CustomerName, Customers.Country, Orders.OrderDate, Orders.OrderID from customers **INNER JOIN** orders on Customers.CustomerID = Orders.CustomerID;  ---  customers INNER JOIN orders OR  orders INNER JOIN customers  > gives the same result.

1. **Left Join** :   
   The LEFT JOIN keyword returns all records from the left table (table1)(i.e. all column values of the column/s mentioned in the query), and the matched records from the right table (table2). The result is **NULL** from the right side if there is no match.

ex. : select Customers.CustomerName, Orders.OrderID, Orders.OrderDate from Customers **left join** orders on Customers.CustomerID = orders.CustomerID; (here all column values of Customers.CustomerName will be fetched and if there is no result from the right side then NULL will be printed in front of those values.)

1. **Right Join** :   
   The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.
2. Ex. : select Customers.CustomerName, Customers.ContactName, Orders.OrderDate from Customers **right join** orders on Customers.CustomerID = orders.CustomerID;
3. **FULL OUTER JOIN / FULL JOIN** :   
   The FULL OUTER JOIN keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.  The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.
4. Ex. : SELECT Customers.CustomerName, Orders.OrderID FROM Customers **FULL OUTER JOIN** Orders ON Customers.CustomerID=Orders.CustomerID
5. **GroupBy** :  The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".  here we can group-by country   
   SELECT CustomerID, Country  
   FROM Customers GROUP BY Country
6. **Having** :   
   SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country HAVING COUNT(CustomerID) > 5;
7. **Distinct** :   
   The SELECT DISTINCT statement is used to return only distinct (different) values.  
   Inside a table, a column often contains many duplicate values then   SELECT DISTINCT eliminates duplicate records from the results.    
   ex. :  SELECT DISTINCT Country FROM Customers;
8. **SUB** **Query**  :  
   Always sub-query executes first and that result is used to generate the result from the outer query  
   Ex. : select \* from OrderDetails where OrderId in (select OrderId where productID > 70);  
   select \* from Suppliers where SupplierID in (select SupplierID where Country like 'USA');  
   Select \* from OrderDetails where OrderDetailID in (select OrderDetailID from OrderDetails where ProductID > 50 and Quantity > 40);
9. **Limit / Top** :   
   Limit : used in Mysql : to fetch limited records in DB  
   Top :  used in Microsoft SQL  : to fetch top records in DB

Interview SQL :

1. **Second highest** sal using MAX :   
   a. select max(Quantity) from OrderDetails where Quantity not in (select max(Quantity) from OrderDetails)   
   b. select max(orderID) from OrderDetails where OrderID < (select max(orderID) from OrderDetails);  
   c. select max(Price) from products where price < (select max(Price) from products);
2. **second and third highest** salary : / **N**th HIGHEST SAL.  
   SELECT Fname, Lname  
   FROM Employee  
   ORDER BY Salary  
   Limit 3  
   OFFSET 10;  
   ( OFFSET 10 : Start from here i.e. it start showing from 11th row till : 13th row i.e. 11, 12, 13 )
3. **3rd largest salary** : EASY SOLUTION : select distinct (salary) from employee order by salary desc LIMIT 2,1 > in the query change DESC to ASC if want smallest sal.
4. **GENERIC query** :   select distinct (salary) from employee order by salary desc LIMIT (n-1,1) : nth largest salary. Ex. : SELECT distinct(Price) FROM Products order by Price desc limit 2,1;
5. Select **TOP** 1ST salary :   
   a. SELECT top 1 EmployeeID FROM employee order by salary **desc**;  
   b. Select max(Salary) from employee;  
   c. SELECT \* from customers limit 1 offset 1;
6. PRINT even / odd/ alternate rows of table :   
   a.  SELECT \* FROM Products where ProductID%1=0;   //prints odd rows   
   b.  SELECT \* FROM Products where ProductID%2=0;   //prints even rows
7. Print all **DUPLICATE**rows available in table :   
   a. select OrderID, CustomerID, EmployeeID, **count(ShipperID)** from Orders group by **ShipperID**having **count(ShipperID**) > 1; ///// here all duplicate ShipperID will be listed  
   b. select OrderID, CustomerID, EmployeeID, ShipperID, **count(\*)** from Orders group by **CustomerID**having **count(\*) > 1**;  ///// here all duplicate CustomerID will be listed
8. Select **1st and last N** rows :   
   a. 1st 5 : Select \* from orders order by orderId desc limit 5;   ///// here 1st top 5 records will be displayed  
   b. Last 5 : Select \* from orders order by orderId asc limit 5;   ///// here Last top 5 records will be displayed   
   c. 1st 5 without any order : Select \* from orders limit 8;   ///// here without sorting 1st 8 records will be displayed
9. Select **Nth row from table** :   
   a. select \* from Orders order by orderId limit 2 offset 10;    //// start displaying 11th and 12th  
   b. select \* from Orders order by orderId limit 1 offset 5;    //// only 6th Row will  be displayed

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  offset n : start displaying **from**the **n+1 th element**,.

1. INTERSECTION :   
   a.  The SQL INTERSECT operator is used to return the results of 2 or more SELECT statements. However, it only returns the rows selected by all queries or data sets. If a record exists in one query and not in the other, it will be omitted from the INTERSECT results.    
   b. Ex. :  SELECT CustomerID FROM Orders INTERSECT SELECT CustomerID FROM customers;