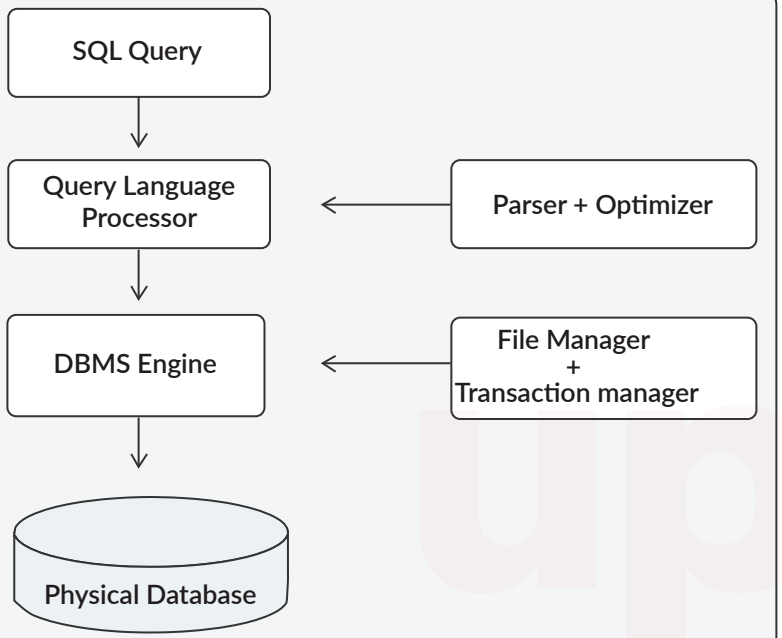
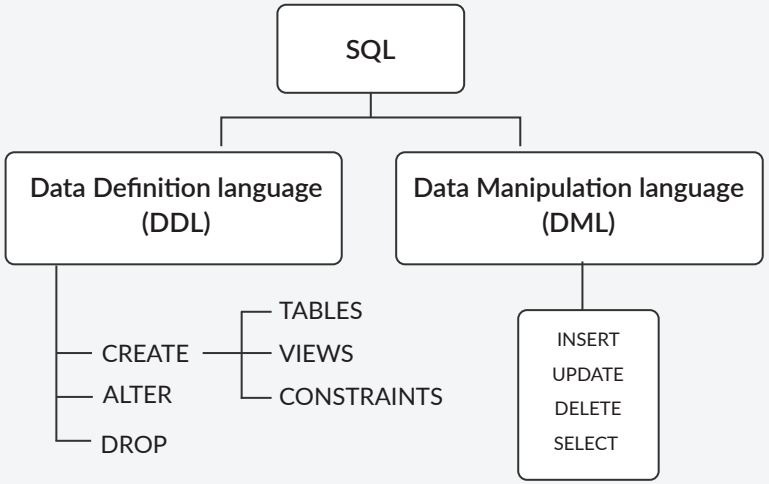
# SQL

Before you are introduced to SQL, you need to know about databases. A database is an organized collection of stored data, and software known as database management systems (DBMSes) is widely used to store, retrieve, and modify data in databases. DBMSes maintain a central data repository of an entire enterprise, which is known as a data warehouse. SQL is a data retrieval language used in RDBMSes (relational database management systems) for storing, manipulating, and retrieving data.




### NOT NULL
Ensures that a column does not have a NULL value

### UNIQUE
Ensures that all values in a column are different

### PRIMARY KEY
A combination of a NOT NULL and UNIQUE value. Uniquely identifies each row in a table

### FOREIGN KEY
Prevents actions that would destroy links between tables

### CREATE INDEX
Used to create and retrieve data from the database very quickly

### DEFAULT
Sets a default value for a column if no value is specified

### CHECK
Ensures that the values in a column satisfy a specific condition

**Strings datatype**
CHAR (size) VARCHAR (size)

**Numeric datatype**
INT(size) FLOAT(precision) BOOL

**Date/time datatype**
DATE YEAR

## INTERVIEW QUESTIONS

1. What is a Database/DBMS/SQL?
2. What is a Primary/Foreign Key?
3. What is a Join? List its types.
4. What is a self-join?
5. What is a View? How can you create a View?
6. How to test for NULL values in a database?
7. What is the purpose of using Indexes in SQL?
8. What is the difference between group by and having clause in SQL?
9. What is the difference between DROP and TRUNCATE statements?
10. What are aggregate and scalar functions?
11. How to fetch alternate records from a table?
12. How would you find the second highest salary from a table?
13. When should I consider using a view instead of writing queries against base tables?
14. Is it possible to write complex joins and other operations on a view?
15. What makes a union clause different from a join clause?

## Operator

arithmetic + - * / %
Bitwise & | ^
Comparison > < = >= <= <>
Compound += -= *= /= %= &= |= ^=

## Keywords

FROM, WHERE, LIMIT, VALUES, DISTINCT, LIMIT, ORDER BY, ASC, DESC, GROUP BY, HAVING, SET, DEFAULT, SET, CASE

## Functions

String: CONCAT, ASCII, LOWER, UPPER, SUBSTR, TRIM
Number: ABS, AVG, COUNT, MAX, MIN, MOD, POWER, SUM, SORT
Date: ADDDATE, ADDTIME, CURRENT_DATE, DATE, DAY, YEAR, DATEDIFF

## Comments

Single line - -
Multi line /* */

## UNION: no duplicate rows

## UNION ALL: duplicate rows allowed

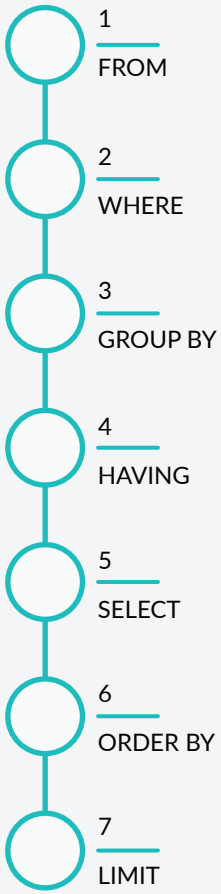
## Set operrations

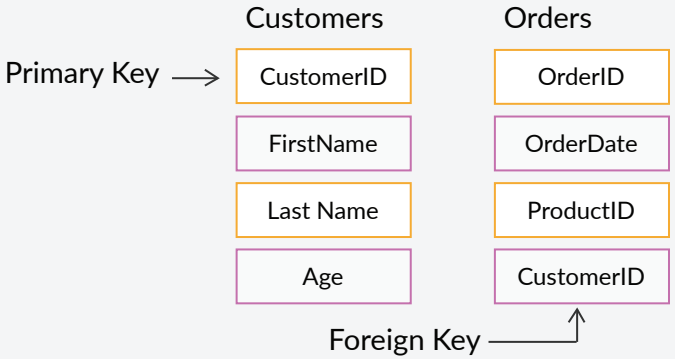
UNION, INTERSECT, DIFFERENCE

## Wildcards

%, -, []

## ORDER OF EXECUTION




## DDL Examples

**create a table**
create table Customers (CustomeriD int primary key, FirstName char (20), LastName varchar (20), Age int);

**Add an extra column to the table**
alter table Customers add PhoneNo varchar(12);

**Remove/drop a column from the table**
alter table Customers drop PhoneNo varchar 2);

Change the datatype of en existing column
alter table Customers alter FirstName varchar(20);

Drop a table
drob table Customers;

## DML Examples:

**Insert data into table**
insert into Customers values(502,"Jane", "Dsouza", 28);

**Modify existing data of a table**
update Customers set FirstName = "Jack", Age=30 where CustomeriD=502;

**Fetch the data from the table using wildcard**
select * from Customers where LastName LIKE '%s;

**Filter the data from the table - lists the number of orders of, each customer only if the customer has ordered more than 5 different products, sorted high to low**
select COUNT(ProductiD), CustomeriD from Orders group by CustomerID having COUNT(ProductID) > 5 order by COUNT(ProductiD) DESC;

**List all the product details of product bat, mat and hat**
select * from Products where ProductName LIKE "_at":

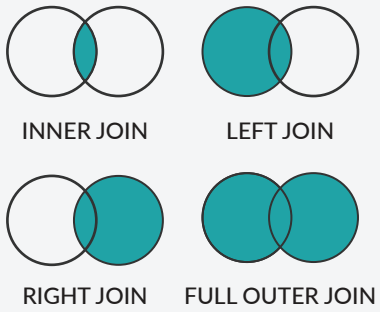
List all the customers whose age is between 25 and 50
select * from Customers where Age BETWEEN 25 and 50;

## SAMPLE DATABASE

This is the structure of the database tables. We will use this structure for the following SQL queries throughout.

| Orders | Products | Customers | Suppliers |
|---|---|---|---|
| OrderID | ProductID | CustomersID | SuppliersID |
| OrderDate | ProductName | FirstName | FirstName |
| ProductID | ProductPrice | Last Name | LastName |
| CustomerID | ProductQuantity | Age | Age |

| Query | Description |
|---|---|
| select LastName from Customers UNION ALL select LastName from Suppliers ORDER BY LastName; | It lists the lastNames( duplicates included) from both the tables. |
| select LastName from Customers UNION select LastName from Suppliers ORDER BY LastName; | it lists the distinct lastNames from both of the tables |
| Select A.FirstName as CustomerName1, B.FirstName AS CustomerName2, A.LastName from Customers A, Customers B where A.CustomerID <> B.CustomerID AND A.LastName = B.LastName ORDER BY A.LastName; | it lists the customers with the same last name ( SELF JOIN ) |
| select Customers.FirstName, Orders.OrderID from Customers FULL OUTER JOIN Orders ON Customers. CustomerID=Orders.CustomerID ORDER BY Customers.FirstName; | it lists all the customers and orders even if there is no matching record |
| select Orders.OrderID, Product.ProductID from Orders RIGHT JOIN Products ON Orders.ProductID = Products. ProductID ORDER BY Orders.OrderID; | It lists all the Product Id's and order details if any orders are placed |
| select Customers.FirstName, Orders.OrderID from Customers LEFT JOIN Orders ON Customers. CustomerID = Orders.CustomerID ORDER BY Customers.FirstName; | It lists all the customers and any orders they have |
| select Orders.OrderID, Customers.FirstName, Orders.OrderDate from Orders INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID; | It lists the customer's order details only of those customers with matching records |
| select * from Customers where LastName LIKE '%s"; | It lists all the customers whose LastName ends with letter 's' |
| select * from Customers where Age BETWEEN 25 and 50; | It lists all the customers whose age is between 25 and 50 |
| select * from Products where ProductPrice IN ( 2000 3000 4000); | It lists the products whose price is .2000 or 3000 or 4000 |
| select DISTINCT Age from Customers; | It lists the distinct ages of customers from the Customers table |
| select CustomerID, FirstName from Customers where Age>25; | It selects CustomerID, FirstName from Customers where Age>25; |
| insert into Customers values (CustomerID, FirstName) values (501, "John"); | Insert data into the Customers table |
| create table Customers (CustomerID int primary key, FirstName char(20), LastName varchar(20), Age int) ; | FirstName stores fixed length characters and LastName stores variable length characters |

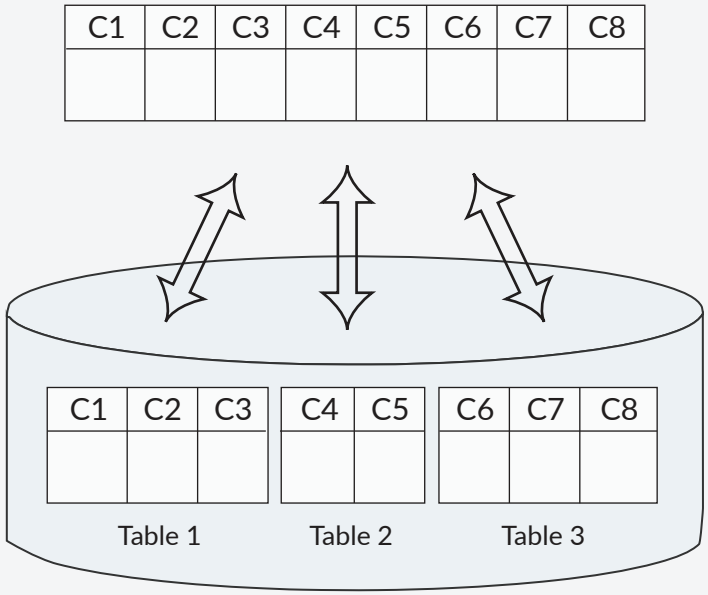



```
SELECT Customers.customer_id, Customers.first_name,
Orders.amount FROM Customers
INNER JOIN Orders ON Customers.customer_id = Orders.customer;

SELECT Customers.customer_id, Customers.first_name,
Orders.amount FROM Customers
LEFT JOIN Orders ON Customers.customer_id = Orders.customer;
```

```
SELECT Customers.customer_id, Customers.first_name,
Orders.amount FROM
Customers
RIGHT JOIN Orders ON Customers.customer_id = Orders.customer;
SELECT Customers.customer_id, Customers.first_name,
Orders.amount FROM Customers
FULL OUTER JOIN Orders ON Customers.customer_id = Orders.customer;
```

## SQL VIEWS




## SQL Views

```
CREATE VIEW [Products above average price] AS
SELECT ProductName, Price
FROM Products
WHERE Price > (SELECT AVG(Price) FROM Products);

SELECT * FROM [Products Above Average Price];
```

## Common Table Expressions

-used to create a temporary table

-cannot be individually queried, has to be used as part of the main query

-Syntax: WITH cte_name (column_list) AS ( query )

```
CREATE VIEW order_details AS
SELECT Customers.CustomerID, Customers.FirstName, Orders.OrderID
FROM Customers
JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;

SELECT * FROM order_details;
```