Nowadays, every organization needs data to run its business. Databases are the best option to store organized collection of data. And, SQL (Structured Query Language) is the most widely used programming language for organizing and retrieving the data in a database. It allows us to perform all the CRUD (create, read, update and delete) operations in the database.

Purpose of SQL

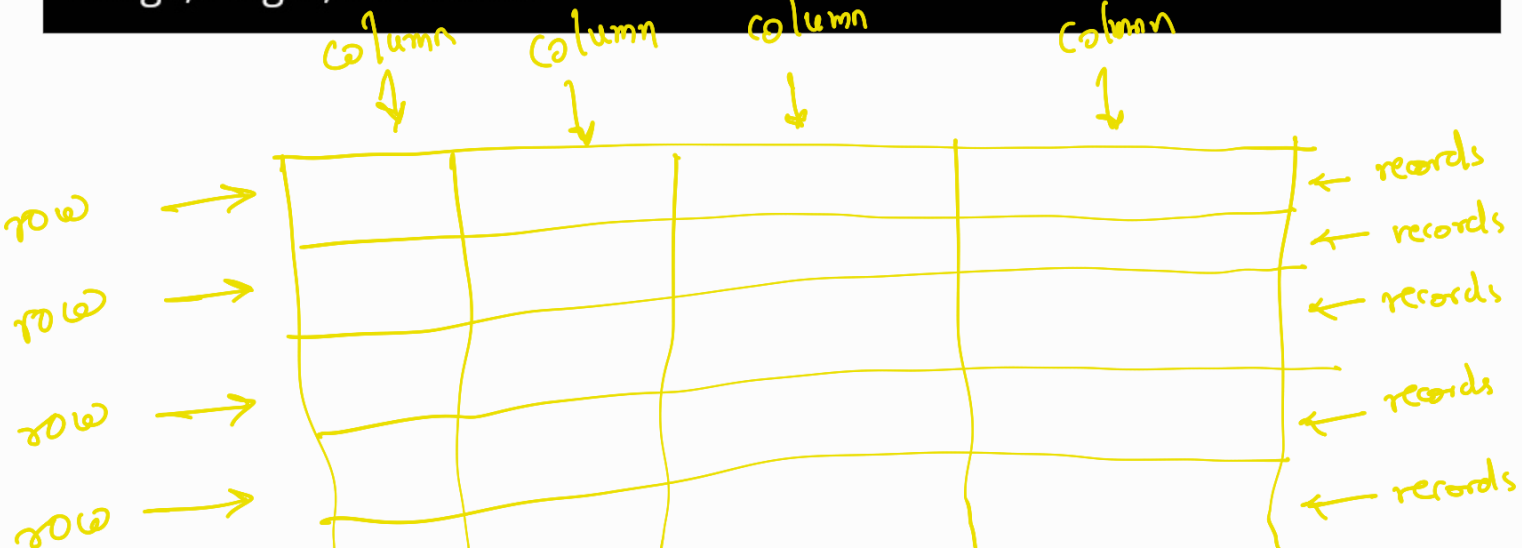
The main purpose of SQL is to operate and retrieve information from the relational database. It allows us to create new databases, views, tables, stored procedures, and functions.

Table

A table is a database object that presents data in columns and rows.

Records and fields

Rows are described as records. The columns are described as fields that represent the category of the records. For example, a table of student details contains a row for each student and a column for each detail such as age, height, and so on.



Name	Age	Gender	COLUMN Eye olor
ROW Kelly	26	Female	Blue
Jim	52	Male	Brown
Marge	87	Female	Green

ID	Firstname	Lastname	Gender	DOB
1	John	Lennon	M	9/10/1940
2	Ringo	Starr	M	7/7/1940
3	Paul	McCartney	M	18/6/1942
4	George	Harrison	M	25/2/1943

Record Field Customer Table

SQL data types

The basic things required to create a column are names and data types. SQL supports the following three data types:

- String data type
- Numeric data type
- Date data type

SQL clauses

SQL is a case-insensitive language. The major three clauses in SQL are **Select**, **from**, **where**.

Select

Select is the most important clause in SQL. It helps us retrieve data from the table. Simply, it answers **What data should we show?**

from

This retrieves data from a specific table in a database. Simply, it answers **Where do we get the data from?**

where

This retrieves specific data records in a table. Simply, it answers **Which category data should we show?**

```
Select  * from Customers where CustomerId = 1
```

90 %

Results | Messages

	CustomerId	CustomerName	Address	City	State	Country	PostalCode
1	1	Anbu	KK Nagar	Chennai	Tamil Nadu	India	600028

SQL categories

There are five categories in SQL:

- **Data definition language:** It performs **Create**, **Alter**, **Drop** operations.
- **Data manipulation language:** It performs **Insert**, **Update**, **Delete** operations.
- **Data control language**: It performs **Grant** and **Revoke** operations.
- **Data query language:** It performs the **Select** operation.
- **Transaction control language:** It performs **Commit** and **Rollback** operations.

- Data Definition language :- Create, Alter, Drop
- Data Manipulation Language :- Insert, Update, Delete

- Data Query language :- Select
- Data control language :- Grant, Revoke
- Transaction control language :- Commit, Rollback

#1: CREATE

This keyword enables us to create a new table.

Syntax:

```
Create table <<table name>>
(
  <<column name1>> datatype,
  <<column name2>> datatype
)
```

```
Create table Customers
(
    CustomerId int Not Null,
    CustomerName varchar(max),
    Address varchar(max),
    City varchar(max),
    State varchar(max),
    Country varchar(max),
    PostalCode int
 )
```

Output:

```
Select * from Customers
```

%

Results | Messages

CustomerId	CustomerName	Address	City	State	Country	PostalCode

#2: Insert into

This keyword is used to insert new records (rows) in a table.

Syntax:

```
Insert into <<table name>>

values (value1, value2, ....)
```

Example:

```
Insert into Customers
values (1,'Anbu','KK Nagar','Chennai', 'Tamil Nadu','India',600028)
```

Output:

```
Select * from Customers
```

90 %

Results | Messages

	CustomerId	CustomerName	Address	City	State	Country	PostalCode
1	1	Anbu	KK Nagar	Chennai	Tamil Nadu	India	600028

Similarly, you can add multiple entries at a time like in the following image.

Syntax:

```
Insert into <<table name>>
values (value1, value2, ....),
       (value1, value2, ....),
       (value1, value2, ....),
```

Example:

```
Insert into Customers
values
(2,'Priya','SH Lane','Mumbai', 'Maharashtra','India',411044),
(3,'George','1st Cross Street','Delhi', 'New Delhi','India',32872),
(4,'Hari','12/A, SH Lane','XYZ', 'S','India',23442),
(5,'Grace','12/A, SH Lane','ABC', 'T','India',42422)
```

Output:

```
Select * from Customers
```

90 %

Results | Messages

	CustomerId	CustomerName	Address	City	State	Country	PostalCode
1	1	Anbu	KK Nagar	Chennai	Tamil Nadu	India	600028
2	2	Priya	SH Lane	Mumbai	Maharashtra	India	411044
3	3	George	1st Cross Street	Delhi	New Delhi	India	32872
4	4	Hari	12/A, SH Lane	XYZ	S	India	23442
5	5	Grace	12/A, SH Lane	ABC	T	India	42422

#3: Update

Updates the existing records in a table.

Syntax:

```
Update <<table name>>>> set <<column name>> = <<value>> where <<column name>> = <<value>>
```

Example:

```
Update Customers set City = 'Bangalore' where CustomerId = 4
```

Output:

Before Update:

```
Select * from Customers where CustomerId = 4
```

90 %

Results | Messages

	CustomerId	CustomerName	Address	City	State	Country	PostalCode
1	4	Hari	12/A, SH Lane	XYZ	S	India	23442

After Update:

```
Select * from Customers where CustomerId = 4
```

90 %

Results | Messages

	CustomerId	CustomerName	Address	City	State	Country	PostalCode
1	4	Hari	12/A, SH Lane	Bangalore	S	India	23442

#4: Distinct

This keyword removes duplicate records and gets the unique records from a table.

Syntax:

```
Select distinct * from <<table name>>
```

Example:

```
Select distinct * from Customers
```

90 %

Results | Messages

	CustomerId	CustomerName	Address	City	State	Country	PostalCode
1	1	Anbu	KK Nagar	Chennai	Tamil Nadu	India	600028
2	2	Priya	SH Lane	Mumbai	Maharashtra	India	411044
3	3	George	1st Cross Street	Delhi	New Delhi	India	32872
4	4	Hari	12/A, SH Lane	XYZ	S	India	23442
5	5	Grace	12/A, SH Lane	ABC	T	India	42422

Refer to the following image to retrieve particular distinct column values.

Syntax:

```
Select distinct <<column name>> from <<table name>>
```

Example:

```
Select distinct Address from Customers
```

90 %

Results | Messages

	Address
1	12/A, SH Lane
2	1st Cross Street
3	KK Nagar
4	SH Lane

#5: top

Use this keyword to get the top values in a table.

Syntax:

```
Select top Numeric Value * from <<table name>>
```

Example:

```
Select top 1 * from Customers
```

90 %

Results | Messages

	CustomerId	CustomerName	Address	City	State	Country	PostalCode
1	1	Anbu	KK Nagar	Chennai	Tamil Nadu	India	600028

#6: order by

We can use this keyword to sort the records in ascending or descending order.

Syntax:

```
Select * from <<table name>> order by <<column name>> asc
```

To sort the records in ascending order, use the keyword **asc.**

Example:

```
Select  * from Customers order by CustomerId asc
```

90 %

Results | Messages

	CustomerId	CustomerName	Address	City	State	Country	PostalCode
1	1	Anbu	KK Nagar	Chennai	Tamil Nadu	India	600028
2	2	Priya	SH Lane	Mumbai	Maharashtra	India	411044
3	3	George	1st Cross Street	Delhi	New Delhi	India	32872
4	4	Hari	12/A, SH Lane	Bangalore	S	India	23442
5	5	Grace	12/A, SH Lane	ABC	T	India	42422

To sort the records in descending order, use the Keyword **desc.**

Syntax:

```
Select  * from <<table name>>  order by <<column name>> desc
```

Example:

```
Select  * from Customers order by CustomerId desc
```

90 %

Results | Messages

	CustomerId	CustomerName	Address	City	State	Country	PostalCode
1	5	Grace	12/A, SH Lane	ABC	T	India	42422
2	4	Hari	12/A, SH Lane	Bangalore	S	India	23442
3	3	George	1st Cross Street	Delhi	New Delhi	India	32872
4	2	Priya	SH Lane	Mumbai	Maharashtra	India	411044
5	1	Anbu	KK Nagar	Chennai	Tamil Nadu	India	600028

#7: and

This keyword is used to display the records that satisfy all the conditions in the **where** clause.

Syntax:

> Select * from <<table name>> where <<column name>> = <<value>> and <<column name>> = <<value>>

```
Select  * from Customers where Address = '12/A, SH Lane' and Customerid = 5
```

90 %

Results | Messages

	CustomerId	CustomerName	Address	City	State	Country	PostalCode
1	5	Grace	12/A, SH Lane	ABC	T	India	42422

#8: or

This keyword displays the records that satisfy any one of the conditions in the **where** clause.

Syntax:

> Select * from <<table name>> where <<column name>> = <<value>> or <<column name>> = <<value>>

```
Select  * from Customers where Address = '12/A, SH Lane' or Customerid = 5
```

90 %

Results | Messages

	CustomerId	CustomerName	Address	City	State	Country	PostalCode
1	4	Hari	12/A, SH Lane	Bangalore	S	India	23442
2	5	Grace	12/A, SH Lane	ABC	T	India	42422

#9: NOT

This keyword displays the records that don't satisfy the provided condition.

Syntax:

```
Select * from <<table name>> where not <<column name>> = <<value>>
```

```
Select * from Customers where not CustomerId = 5
```

90 %

Results | Messages

	CustomerId	CustomerName	Address	City	State	Country	PostalCode
1	2	Priya	SH Lane	Mumbai	Maharashtra	India	411044
2	3	George	1st Cross Street	Delhi	New Delhi	India	32872
3	4	Hari	12/A, SH Lane	NULL	S	India	23442
4	1	Anbu	KK Nagar	Chennai	Tamil Nadu	India	600028

#10: MIN

This keyword displays the smallest value in a column.

Syntax:

```
Select min(<<column name>>>>) from <<table name>>
```

```
Select min(PostalCode) from Customers
```

90 %

Results | Messages

	(No column name)
1	23442

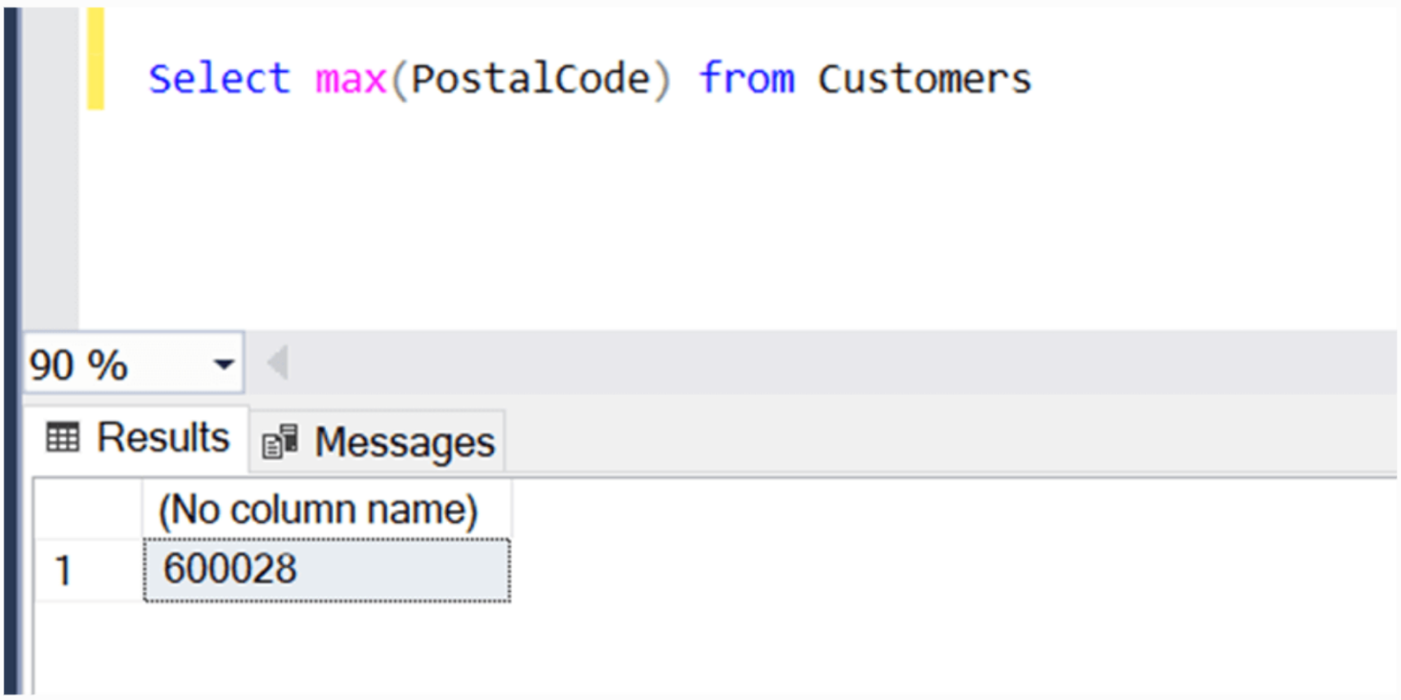
#11: MAX

This keyword displays the largest value in a column.

Syntax:

```
Select max(<<column name>>) from <<table name>>
```

Example:



#12: SUM

This keyword displays the total sum value for the numeric column.

Syntax:

```
Select sum(<<column name>>) from <<table name>>
```

```
Select sum(PostalCode) from Customers
```

90 %

Results | Messages

	(No column name)
1	1109808

#13: in

This keyword retrieves multiple values that satisfy the condition in the **where** clause.

Syntax:

```
Select  * from <<table name>>  where <<column name>> in (<<value1>>, <<value2>>, ....)
```

```
Select * from Customers where CustomerId in (1,2)
```

90 %

Results | Messages

	CustomerId	CustomerName	Address	City	State	Country	PostalCode
1	2	Priya	SH Lane	Mumbai	Maharashtra	India	411044
2	1	Anbu	KK Nagar	Chennai	Tamil Nadu	India	600028

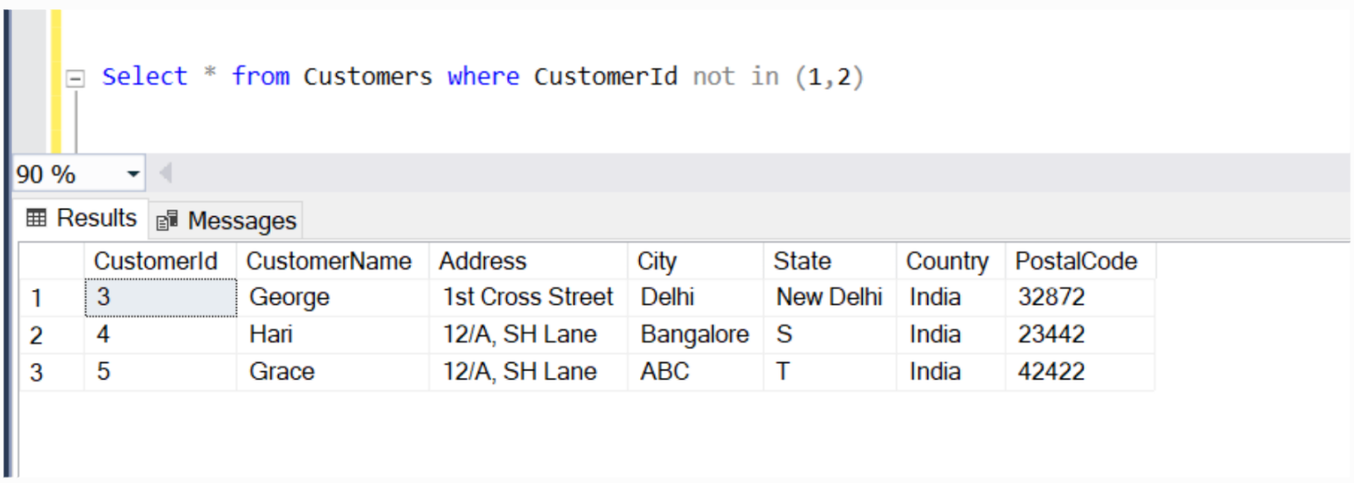
#14: not in

This keyword retrieves multiple values that don't satisfy the condition in the where clause.

Syntax:

```
Select * from <<table name>> where <<column name>> not in (<<value1>>, <<value2>>, ....)
```

Example:



	CustomerId	CustomerName	Address	City	State	Country	PostalCode
1	3	George	1st Cross Street	Delhi	New Delhi	India	32872
2	4	Hari	12/A, SH Lane	Bangalore	S	India	23442
3	5	Grace	12/A, SH Lane	ABC	T	India	42422

#15: Count

Use the Count keyword to return the total number of rows in a table. We can use this to return the number of rows that satisfy the specified condition.

Syntax:

```
Select count(*) from <<table name>>
```

```
Select count(*) from Customers
```

90 %

Results | Messages

	(No column name)
1	5

#16: AVG

This keyword returns the average value of a column.

Syntax:

```
Select avg(<<column name>>) from <<table name>>
```

Example:

```
Select avg(CustomerId) from Customers
```

90 %

Results | Messages

	(No column name)
1	3

#17: ANY

This keyword is used to check whether the required records exist or not in a table.

Syntax:

```
Select * from <<table name>> where <<column name>> = any(Select  * from <<table name>>  where <<column name>> = <<value>>)
```

Example:

```
Select * from Customers where CustomerId = any(Select CustomerId from Customers where CustomerId = 4)
```

90 %

Results | Messages

	CustomerId	CustomerName	Address	City	State	Country	PostalCode
1	4	Hari	12/A, SH Lane	XYZ	S	India	23442

#18: Like

This keyword retrieves a specified pattern in a column.

Syntax:

```
Select * from <<table name>> where <<column name>> like '%Value%'
```

```
Select * from Customers where Address like '%SH Lane%'
```

90 %

Results | Messages

	CustomerId	CustomerName	Address	City	State	Country	PostalCode
1	2	Priya	SH Lane	Mumbai	Maharashtra	India	411044
2	4	Hari	12/A, SH Lane	XYZ	S	India	23442
3	5	Grace	12/A, SH Lane	ABC	T	India	42422

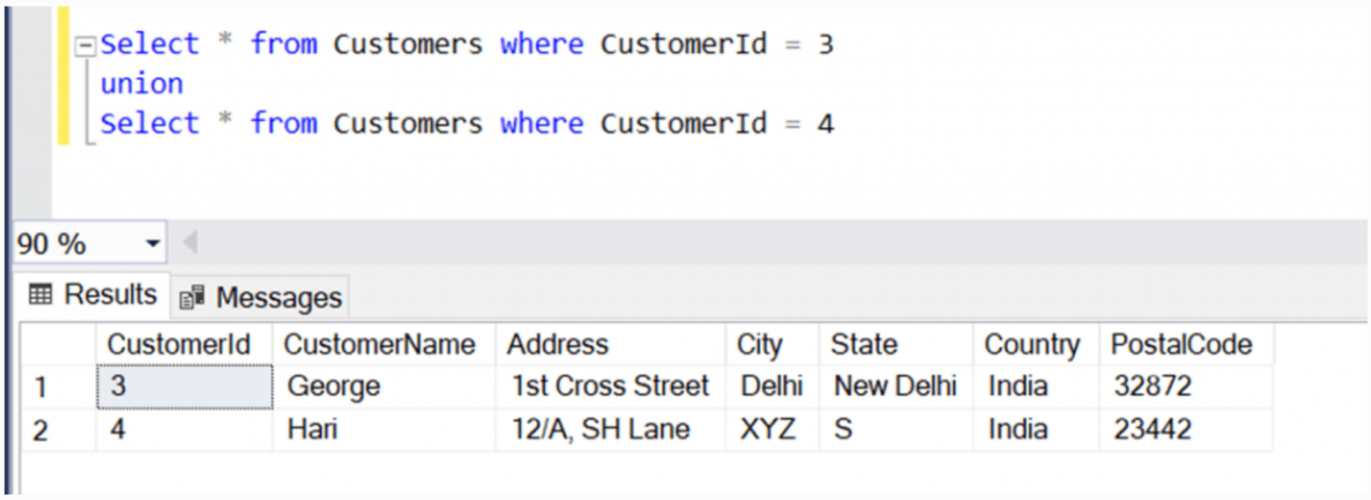
#19: UNION

This keyword is used to combine two or more select statements.

Syntax:

```
Select * from <<table name1>>
union
Select * from <<table name2>>
```

Example:



```
Select * from Customers where CustomerId = 3
union
Select * from Customers where CustomerId = 4
```

90 %

Results | Messages

	CustomerId	CustomerName	Address	City	State	Country	PostalCode
1	3	George	1st Cross Street	Delhi	New Delhi	India	32872
2	4	Hari	12/A, SH Lane	XYZ	S	India	23442

#20: Is NULL

This keyword retrieves the rows that satisfy the null value in a specific column.

Syntax:

```
Select * from <<table name>> where <<column name>> is null
```

Example:

```
Select * from Customers where City is null
```

90 %

Results | Messages

	CustomerId	CustomerName	Address	City	State	Country	PostalCode
1	4	Hari	12/A, SH Lane	NULL	S	India	23442

#21: IS NOT NULL

This keyword retrieves the rows that satisfy the not null value in a specific column.

Syntax:

```
Select * from <<table name>> where <<column name>> is not null
```

Example:

```
Select * from Customers where City is not null
```

90 %

Results | Messages

	CustomerId	CustomerName	Address	City	State	Country	PostalCode
1	2	Priya	SH Lane	Mumbai	Maharashtra	India	411044
2	3	George	1st Cross Street	Delhi	New Delhi	India	32872
3	5	Grace	12/A, SH Lane	ABC	T	India	42422
4	1	Anbu	KK Nagar	Chennai	Tamil Nadu	India	600028

#22: WILDCARDS

This keyword is used instead of a particular character in a string to retrieve all possible values. Some common wildcard values in SQL are: _ * ? [] ! –

Syntax:

```
Select * from <<table name>> where <<column name>> like '<<value with WildCardValue>>'
```

Example:

```
Select * from Customers where PostalCode like '4_422'
```

90 %

Results | Messages

	CustomerId	CustomerName	Address	City	State	Country	PostalCode
1	5	Grace	12/A, SH Lane	ABC	T	India	42422

#23: GROUP BY

This keyword is used to group records. The main purpose of this statement is to find how many records have the same values in a table.

Syntax:

```
Select count(<<column name>>), <<column name>>  from <<table name>> group by <<column name>>
```

Example:

```
Select count(Address), Address from Customers group by Address
```

90 %

Results | Messages

	(No column name)	Address
1	2	12/A, SH Lane
2	1	1st Cross Street
3	1	KK Nagar
4	1	SH Lane

#24: Aliases

We can give a temporary name for a column in the table. Here, the CustomerID column is named **Id,** and the CustomerName column is named **Name**.

Syntax:

```
Select <<column name>> as <<temporary name>> from <<table name>>
```

Example:

```
Select CustomerID as Id,
CustomerName as Name
from Customers
```

90 %

Results | Messages

	Id	Name
1	1	Anbu
2	2	Priya
3	3	George
4	4	Hari
5	5	Grace

#25: Between

This keyword retrieves the values within the given range.

Syntax:

```
Select * from <<table name>> where <<column name>> between <<value>> and <<value>>
```

Example:

```
Select * from Customers where CustomerId between 2 and 4
```

90 %

Results | Messages

	CustomerId	CustomerName	Address	City	State	Country	PostalCode
1	2	Priya	SH Lane	Mumbai	Maharashtra	India	411044
2	3	George	1st Cross Street	Delhi	New Delhi	India	32872
3	4	Hari	12/A, SH Lane	NULL	S	India	23442

#26: Join

This keyword combines records from two or more tables using the common field in them. There are four types of join in SQL:

- Inner Join
- Left Join
- Right Join
- Full Join

Let's look at these Join keywords with example data.

Table 1: Students

```
Select * from Students
```

90 %

Results | Messages

	RollNo	StudentName	JoiningDate
1	1	ABC	2016-09-01 00:00:00.000
2	2	XYZ	2016-11-23 00:00:00.000
3	3	QWT	2016-11-23 00:00:00.000
4	4	KML	2016-11-25 00:00:00.000
5	5	RRQ	2016-11-28 00:00:00.000
6	6	LKK	2016-12-02 00:00:00.000

Table 2: StudentsMarkInformation

```
Select * from StudentsMarkInformation
```

90 %

Results | Messages

	Id	StudentRollNo	Class	AcademicYear	TotalMarks	Grade
1	1	1	5	2016	1034	A
2	2	2	5	2016	827	C
3	3	3	5	2016	965	B
4	4	1	6	2017	756	C
5	5	2	6	2017	935	B
6	6	3	6	2017	955	B
7	7	1	7	2018	1022	A
8	8	2	7	2018	827	C
9	9	3	7	2018	1003	A
10	10	1	8	2019	994	B
11	11	2	8	2019	1056	A
12	12	3	8	2019	965	B

Inner Join:

The Inner Join returns the records that match the values in both tables. Inner Join is commonly referred to as just Join.

Syntax:

> Select <<column name>>
>
> from <<table name1>>
>
> **Inner join** <<table name2>> on <<table name1>>.<<column name>> = <<table name2>>.<<column name>>

Example:

```
Select Students.StudentName, StudentsMarkInformation.Class,StudentsMarkInformation.Grade
From Students
Inner Join StudentsMarkInformation on Students.RollNo = StudentsMarkInformation.StudentRollNo
```

90 %

Results | Messages

	StudentName	Class	Grade
1	ABC	5	A
2	XYZ	5	C
3	QWT	5	B
4	ABC	6	C
5	XYZ	6	B
6	QWT	6	B
7	ABC	7	A
8	XYZ	7	C
9	QWT	7	A
10	ABC	8	B
11	XYZ	8	A
12	QWT	8	B

Left Join:

Left join returns all the records from the left-side table and the matching records from the right-side table of the Join keyword.

Syntax:

> Select <<column name>>
>
> from <<table name1>>
>
> **Left join** <<table name2>> on <<table name1>>.<<column name>> = <<table name2>>.<<column name>>

Example:

```
Select Students.StudentName, StudentsMarkInformation.Class,StudentsMarkInformation.Grade
From Students
Left Join StudentsMarkInformation on Students.RollNo = StudentsMarkInformation.StudentRollNo
```

90 %

Results | Messages

	StudentName	Class	Grade
1	ABC	5	A
2	ABC	6	C
3	ABC	7	A
4	ABC	8	B
5	XYZ	5	C
6	XYZ	6	B
7	XYZ	7	C
8	XYZ	8	A
9	QWT	5	B
10	QWT	6	B
11	QWT	7	A
12	QWT	8	B
13	KML	NULL	NULL
14	RRQ	NULL	NULL
15	LKK	NULL	NULL

Right Join:

Right join returns all the records from the right-side table and the matching records from the left-side table of the Join keyword.

Syntax:

Select <<column name>>

From <<table name1>>

Right join <<table name2>> on <<table name1>>.<<column name>> = <<table name2>>.<<column name>>

Example:

```
Select Students.StudentName, StudentsMarkInformation.Class,StudentsMarkInformation.Grade
From Students
Right Join StudentsMarkInformation on Students.RollNo = StudentsMarkInformation.StudentRollNo
```

90 %

Results | Messages

	StudentName	Class	Grade
1	ABC	5	A
2	XYZ	5	C
3	QWT	5	B
4	ABC	6	C
5	XYZ	6	B
6	QWT	6	B
7	ABC	7	A
8	XYZ	7	C
9	QWT	7	A
10	ABC	8	B
11	XYZ	8	A
12	QWT	8	B

Full Join:

Full join finds the matching records from both the Left Join and Right Join tables and returns all the records for comparision. If there is no match found, then it will return a NULL value for those records. It is also referred as Full Outer Join.

Syntax:

```
Select <<column name>>

from <<table name1>>

Full join <<table name2>> on <<table name1>>.<<column name>> = <<table name2>>.
<<column name>>
```

```
Select Students.StudentName, StudentsMarkInformation.Class,StudentsMarkInformation.Grade
From Students
Full Join StudentsMarkInformation on Students.RollNo = StudentsMarkInformation.StudentRollNo
```

90 %

Results | Messages

	StudentName	Class	Grade
1	ABC	5	A
2	ABC	6	C
3	ABC	7	A
4	ABC	8	B
5	XYZ	5	C
6	XYZ	6	B
7	XYZ	7	C
8	XYZ	8	A
9	QWT	5	B
10	QWT	6	B
11	QWT	7	A
12	QWT	8	B
13	KML	NULL	NULL
14	RRQ	NULL	NULL
15	LKK	NULL	NULL

#27: Primary Key

Primary Key ensures that a value in a record is unique. It never contains NULL values.

Syntax:

```
Create table <<table name>>
(
 <<column name1>> datatype not null primary key,
 <<column name2>> datatype
)
```

Example: Here, the RollNo column is marked as Primary Key, as each student has a unique roll number.

```
Create Table Students
(
   RollNo int Not Null Primary Key,
   StudentName varchar(max),
   JoiningDate datetime,
)
```

#28: Foreign Key

Foreign Key is a field in one table that points to the primary key in another table.

Syntax:

```
Create table <<table name>>

(

 <<column name1>> datatype not null primary key,

 <<column name2>> datatype foreign key references <<existing table name>>
(<<existing column name>>),

 <<column name3>> datatype,

 <<column name4>> datatype

)
```

Example:

```
Create Table StudentsMarkInformation
(
  Id int NOt Null Primary Key,
  StudentRollNo int Foreign Key References Students(RollNo),
  Class int,
  AcademicYear int,
  TotalMarks decimal,
  Grade varchar(20)
)
```

SQL comments

To comment on a particular line in SQL, use the double hyphen symbol (- –). The main purpose of the comments is to explain the process.

```
--Customers Table
Select * from Customers
```

Other interesting topics like stored procedures, views, and functions will be discussed in our upcoming blogs.

What is a Stored Procedure?

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

Stored Procedure Syntax

```
CREATE PROCEDURE procedure_name
AS
sql_statement
GO;
```

Execute a Stored Procedure

```
EXEC procedure_name;
```

Stored Procedure Example

The following SQL statement creates a stored procedure named "SelectAllCustomers" that selects all records from the "Customers" table:

Example

Get your own SQL Server

```
CREATE PROCEDURE SelectAllCustomers
AS
SELECT * FROM Customers
GO;
```

Execute the stored procedure above as follows:

Example

```
EXEC SelectAllCustomers;
```

Stored Procedure With Multiple Parameters

Setting up multiple parameters is very easy. Just list each parameter and the data type separated by a comma as shown below.

The following SQL statement creates a stored procedure that selects Customers from a particular City with a particular PostalCode from the "Customers" table:

Example

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30),
@PostalCode nvarchar(10)
AS
SELECT * FROM Customers WHERE City = @City AND PostalCode =
@PostalCode
GO;
```

Execute the stored procedure above as follows:

Example

```
EXEC SelectAllCustomers @City = 'London', @PostalCode = 'WA1
1DP';
```

• SQL Views :-

SQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

A view is created with the `CREATE VIEW` statement.

CREATE VIEW Syntax

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

Note: A view always shows up-to-date data! The database engine recreates the view, every time a user queries it.

SQL CREATE VIEW Examples

The following SQL creates a view that shows all customers from Brazil:

Example

Get your own SQL Server

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';
```

We can query the view above as follows:

Example

```
SELECT * FROM [Brazil Customers];
```

The following SQL creates a view that selects every product in the "Products" table with a price higher than the average price:

Example

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, Price
FROM Products
WHERE Price > (SELECT AVG(Price) FROM Products);
```

We can query the view above as follows:

Example

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

SQL Updating a View

A view can be updated with the `CREATE OR REPLACE VIEW` statement.

SQL CREATE OR REPLACE VIEW Syntax

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

The following SQL adds the "City" column to the "Brazil Customers" view:

Example

```
CREATE OR REPLACE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName, City
FROM Customers
WHERE Country = 'Brazil';
```

SQL Dropping a View

A view is deleted with the `DROP VIEW` statement.

SQL DROP VIEW Syntax

```
DROP VIEW view_name;
```

The following SQL drops the "Brazil Customers" view:

Example

```
DROP VIEW [Brazil Customers];
```