**SQL**

* SQL stands for Structured Query Language
* SQL lets you access and manipulate databases
* SQL can execute queries against a database
* SQL can retrieve data from a database
* SQL can insert records in a database
* SQL can update records in a database
* SQL can delete records from a database
* SQL can create new databases
* SQL can create new tables in a database
* SQL can create stored procedures in a database
* SQL can create views in a database
* SQL can set permissions on tables, procedures, and views

In Tables there will be rows and columns in that we use the primary key to uniquely identify the values in that keys there are 2 types one is surrogate key and natural key

1) surrogate key is just some unique value that we use to map the attributes in the table

2) Natural keys are the values which has a meaning in the real world like social security number

**Foreign key**

-> it is nothing but one table having another table primary keys as a column

**DATA Types**

INT

DECIMAL(M,N) where M= digit length N = decimal length

VARCHAR(20)

BLOB (used for content like images and big files)

DATE

TIMESTAMP

**Creating Table (used to create table)**

CREATE TABLE student(

ID INT PRIMARY KEY,

Name VARCHAR(20),

Major VARCHAR(20)

);

**DESCRIBE (is used to see the table structure )**

DESCRIBE student;

Pragma table\_info(stduent);

**DROP (is used to delete table or column of table)**

DROP TABLE student;

**ALTER (is used to modify the table like we can add the column to the table)**

ALTER TABLE student ADD Marks VARCHAR(20) ;

ALTER TABLE student DROP COLUMN Marks;

**INSERTING DATA into TABLE**

INSERT INTO student VALUES(1,’SAM’,’Biology’);

INSERT INTO student(Std\_ID, NAME)  
VALUES (2, 'Stev');

**SELECT**

SELECT \* FROM student;

SELECT NAME,MAJOR

FROM students;

SELECT \* FROM student

WHERE Major=’ARTS’;

**NOT NULL (when we define not null this column can't have null values)**

**UNIQUE (we use this to make column take only unique values)**

**DEFAULT is used to assign the value if we don’t insert the value**

**AUTO-INCREMENT is used for the automatically increment the values by it-self**

CREATE TABLE student(

ID INT AUTO\_INCREMENT,

Name VARCHAR(20) NOT NULL,

Major VARCHAR(20) UNIQUE,

CLASS VARCHAR(20) DEFAULT ‘A’

);

**UPDATE**

UPDATE student

SET Major = ‘BIO’ , Name = ‘Tom’ // we can set multiple values by ,

WHERE Major = ‘Biology’;

We can use conditions after where like = , <>(not equal) , < , >, <= and >=

OR AND operations can be also done

**DELETE (we can delete the rows in a table by using delete)**

DELETE FROM student // it will delete all rows but not the table like drop

DELETE FROM student

WHERE Std\_ID = 1;

**SELECT DISTINCT (it is used to get the unique values by filtering the duplicates )**

SELECT DISTINCT NAME,MAJOR,AGE FROM students;

**ORDER BY** (used to sort in ascending or descending order)

SELECT \* FROM students

Order BY NAME;

SELECT \* FROM students

Order BY NAME DESC; // DESC is used to sort in descending order

**LIMIT (it is used to limit the numbers of tables rows to display )**

SELECT \* FROM students

ORDER BY NAME;

LIMIT 2;

MORE on SELECT

SELECT \* FROM student

WHERE Name IN ('John','David');

CREATE TABLE employee (

emp\_id INT PRIMARY KEY,

first\_name VARCHAR(40),

last\_name VARCHAR(40),

birth\_day DATE, sex VARCHAR(1),

salary INT,

super\_id INT,

branch\_id INT );

CREATE TABLE branch (

branch\_id INT PRIMARY KEY,

branch\_name VARCHAR(40),

mgr\_id INT,

mgr\_start\_date DATE,

FOREIGN KEY(mgr\_id) REFERENCES employee(emp\_id) ON DELETE SET NULL );

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

ALTER TABLE employee

ADD FOREIGN KEY(branch\_id)

REFERENCES branch(branch\_id) ON DELETE SET NULL;

**Order BY using multiple things so first it will order the M and F then within it will order by their names**

SELECT \* FROM Employee

ORDER BY SEX , First\_name,Last\_name ;

**AS (it is used to name the columns by alias )**

SELECT First\_name AS Surname , Last\_name AS Name

FROM Employee ;

**SQL FUNCTIONS**

**COUNT ->** (SELECT COUNT(Emp\_ID) FROM Employee)

Example

SELECT COUNT(emp\_id) FROM Employee

WHERE sex =’F’ AND Birth\_day > ‘1971-01-01';

**AVG ->** SELECT AVG(salary) FROM Employee;

SIMILARY

SUM()

MAX()

MIN()

MOD(ID,2)

SELECT COUNT(CITY)-COUNT(DISTINCT CITY) FROM STATION ;

**GROUP BY (It is used with Aggregate functions in sql to differenceate the data )**

SELECT COUNT(branch\_id),branch\_id FROM Employee

GROUP BY branch\_id;

SELECT SUM(salary) as TOTAL,sex FROM Employee

GROUP BY sex;

**LIKE** (used to search for the certain strings that ends with that string)

%String -> ends with that string

String% -> starts with that string

%string% -> anywhere with that string

\_ is used when we know specific no of characters like date 1970-10-12 so we can say like SELECT \* FROM Employee WHERE birth\_day ‘\_\_\_\_-10'

SELECT \* FROM Client

WHERE client\_name LIKE '%LLC';

**UNION (is used to combine the columns of different tables )**

SELECT client\_name FROM Client

UNION

SELECT first\_name FROM Employee;

**INNER JOIN**

ALL the rows which has a common columns in both table will be displayed

SELECT Customers.customer\_id , Customers.first\_name ,Orders.item

FROM Customers

JOIN Orders

ON Customers.customer\_id = Orders.customer\_id;

**LEFT JOIN**

Rows from TABLE 1 will be displayed and Rows from Table 2 which matches with column will be displayed

SELECT Customers.customer\_id , Customers.first\_name ,Orders.item

FROM Customers

LEFT JOIN Orders

ON Customers.customer\_id = Orders.customer\_id;

**RIGHT JOIN**

Opposite to the left table

**ALL OUTER JOIN**

IT will display all the rows of TABLE 1 and TABLE 2

Self JOIN

**NESTED SELECT**

SELECT Customers.first\_name,Orders.item , Orders.amount

FROM Orders

JOIN Customers

ON Customers.customer\_id = Orders.customer\_id

WHERE Orders.customer\_id IN (

SELECT Customers.customer\_id

FROM Customers

WHERE Customers.age <30

);

SELECT client\_name AS Clients\_of\_Michel

FROM Client

WHERE Client.branch\_id = (

SELECT Branch.branch\_id

FROM Branch

WHERE mgr\_id=102

);

**ON DELETE (**It will set the values related to other table to NULL when it get deleted**)**

CREATE TABLE branch (

branch\_id INT PRIMARY KEY,

branch\_name VARCHAR(40),

mgr\_id INT,

mgr\_start\_date DATE,

FOREIGN KEY(mgr\_id) REFERENCES employee(emp\_id) **ON DELETE SET NULL );**

ON DELETE CASCADE ( on delete on values realted to other table it will delete all related things )

FOREIGN KEY(client\_id) REFERENCES client(client\_id) ON DELETE CASCADE

**TRIGGERS (Triggers are used to do certain actions when user perform some queries ) they have to performed in the sql command line**

DELIMITER $$

CREATE TRIGGER my\_trigger BEFORE INSERT

ON employee

FOR EACH ROW BEGIN

INSERT INTO trigger\_test VALUES(NEW.first\_name);

END$$

DELIMITER ;

DELIMITER $$

CREATE TRIGGER my\_trigger BEFORE INSERT

ON employee

FOR EACH ROW BEGIN

IF NEW.sex = 'M'

THEN INSERT INTO trigger\_test VALUES('added male employee'); ELSEIF

NEW.sex = 'F' THEN INSERT INTO trigger\_test VALUES('added female');

ELSE INSERT INTO trigger\_test VALUES('added other employee'); END IF;

END$$

DELIMITER ;

**DataBase Queries**

CREATE DATABASE *databasename*;

DROP DATABASE *databasename*;

BACKUP DATABASE testDB  
TO DISK = 'D:\backups\testDB.bak';

Injection

SELECT UserId, Name, Password FROM Users WHERE UserId = 105 or 1=1;

**SELECT TOP** (USed to get no of rows from the table)

SELECT TOP 3 \* FROM Customers;

**Having** (It is used to do aggregate functions which we can't do using WHERE)

SELECT count(emp\_id),sex

FROM Employee

GROUP BY sex

HAVING COUNT(sex)>4;

**CASE**

SELECT  
 name,  
 no\_of\_views,  
 CASE  
 WHEN no\_of\_views <= 10000 THEN 'poor'  
 WHEN no\_of\_views > 100000 THEN 'good'  
 ELSE 'average'  
 END AS category  
 FROM  
 VIDEO

**BETWEEN** (range)

SELECT \* FROM Products  
WHERE Price BETWEEN 10 AND 20;

**PROCEDURE (It is used to create functions )**

CREATE PROCEDURE SelectAllCustomers  
AS  
SELECT \* FROM Customers  
GO;

EXEC SelectAllCustomers;

**INSERT INTO SELECT**

INSERT INTO Customers (CustomerName, City, Country)  
SELECT SupplierName, City, Country FROM Suppliers;

**ALIAS for names**

SELECT o.customer\_id FROM Customers

JOIN Orders o

ON Customers.customer\_id=o.customer\_id

Order by o.Customer\_id;

**EXTRAS**

LENGTH() -> used to find the length of string

SELECT MIN(CITY),LENGTH(MIN(CITY)) FROM STATION;

SELECT MAX(CITY), LENGTH(MAX(CITY)) FROM STATION;

SELECT CITY c, LENGTH(CITY) l

FROM STATION

ORDER BY l DESC , c asc

limit 1;

SELECT CITY c, LENGTH(CITY) l

FROM STATION

ORDER BY l asc , c asc

limit 1;

SELECT first\_name FROM Customers

WHERE first\_name like 'j%' OR first\_name like 'r%';

SELECT DISTINCT(CITY)

FROM STATION

WHERE SUBSTR(CITY, 1, 1) IN ('A', 'E', 'I', 'O', 'U’);

**SELECT** **DISTINCT** CITY **FROM** STATION **WHERE** CITY REGEXP '^[AEIOU]'

SELECT DISTINCT (CITY)

FROM STATION

WHERE

LEFT(CITY,1) IN ('A','E','I','O','U')

AND

RIGHT(CITY,1) IN ('a','e','i','o','u');

**Cast() (caste is used to convert the value into a specific datatype )**

SELECT CAST(150 AS CHAR);

Floot(val) -> it is used to remove the decimals

CIEL(val)

SELECT \*,replace(age,1,'') as age FROM Customers; -> it is used to replace things

SELECT ROUND(SQRT(POWER(MAX(LAT\_N)-MIN(LAT\_N),2)+(POWER(MAX(LONG\_W)-MIN(LONG\_W), 2))), 4)  
FROM Station;

DATE\_ADD(CURRENT\_DATE,INTERVAL- 24 MONTH)

SELECT CONCAT('SQL', ' is', ' fun!');

**CREATIN A TABLE FROM OTHER TABLES**

CREATE TABLE test AS

SELECT Customers.customer\_id,Orders.amount FROM Customers

JOIN Orders

on Customers.customer\_id=Orders.customer\_id;

**Selecting all columns from one table and few from the other**

SELECT DISTINCT CUSTOMERS.\*,item FROM Customers

join Orders

ON Customers.customer\_id=Orders.customer\_id;

SELECT MAX(salary\*months) as earning," ",count(employee\_id) FROM Employee

WHERE salary\*months=(SELECT MAX(salary\*months) FROM Employee)

**JOINS ON MULTIPLE THINGS CAN BE DONE TOO AND IF Condition**

SELECT if(b.grade >= 8,a.name,'NULL'),b.grade,a.Marks FROM Students a

JOIN Grades b

ON a.Marks Between b.Min\_mark AND b.Max\_mark

ORDER BY b.Grade DESC,Name ASC;

declare

num1 number:= 10;

num2 number:= 20;

begin

if num1 > num2 then

dbms\_output.put\_line('num1 small');

end if;

dbms\_output.put\_line('I am Not in if');

end;