SQL - Structured Query Language

DB - Database

Any collection of related information

DBMS - A special software program that helps users create and maintain database.

1) Makes easy to maintain large data
2) Handles Security
3) Backups
4) Import / exporting data
5) Concurrency
6) Interacts with software applications

Ⓒreate
Ⓡetrieve or read
Ⓤpdate
Ⓓelete

Types Of Database

Relational DB

1) data into one or more tables

2) Each table has rows and columns

3) A unique key Identifies each row

4) eg spreadsheet

Non-Relational DB

1) organize data is anything but traditional table

2) key value stores
documents (JSON, XML etc)
Graphs
flexible tables

SQL – used to Query the language for databases like (MySql, oracle, PostgreSQL, mariaDB, etc).

↳ Language for interacting with RDBMS
↳ for doing CRUD Operations
↳ define tables & structures
↳ SQL code needs modification to use with other databases.

Non-Relational :- (uses other data structures)

which is anything not a relational database.

(no SQL)

1) Document (JSON, BLOB, XML, etc)

2) Graph (Relational nodes)

3) Key-values Hash

NRDBMS :-

eg: MongodB, dynamoDB, apache, cassendra, fire base etc...

1) There's no set of set language standards

2) Implement their own language for performing CRUD.

Query:

are request made to dBMS for specific information.

as the database's structure become more and more complex. It is tough to get the specific piece of information we want

↑ google search is a query.

Tables and keys :-

column — will have single attributes

Rows — instance of the specific entry eg (student info)

Primary key → 1) is gonna be unique
2) used to differentiate for same attributes
3) It can be anything eg (int, string)

Student_id	name	major
1	Kate	BIO
2	Jack	Eng
3	Jack	BIO

column ↓

→ row

Employee_id (no - mapping)
↳ surrogate key (no real world meaning).

natural key - aadhar number to differentiate people.
↳ real world attribute)

foreign key - to link with other table
↳ It is the primary key of Somewhere table.
Branch_id to know the branch
→ relationship with other tables
→ we can use many foreign keys to define relationship.

Composite key :- = Primary + Primary (key)

↳ only together identify each row

↳ only one combination can exist

two primary joined together

() special type of composite key = Foreign + Foreign

Two ~~Primary~~ foreign key - used as primary key to relate

Creating table :-

```
CREATE DATABASE mydatabase;

CREATE TABLE MyTable (

    id int,
    name varchar(10)

);
```

```
CREATE INDEX Indexname
ON Tablename (col1);
```

control ← grant, revoke

Definition DCL

DDL ← create, drop, alter, truncate, (comment), (rename)

Query DQL - select

manipulation DML = Insert, update, delete

Transaction TCL - commit, roll back, save point

control

Data definition language :-

1) Create
2) Alter
3) drop

Data manipulation language :-

1) Update
2) Delete
3) Insert
4) Select

1) Create table (;) sql query end with (;)

2) Data types in SQL :-

i) INT

ii) DECIMAL (M - Total numbers, N - num after decimal)

iii) VARCHAR (1) - storing text with length

iv) BLOB - Binary large object, Stores large data (Images or files).

v) DATE - "YYYY-MM-DD"

vi) TIMESTAMP - " HH:MM:SS (two recording when item includes into DB.

↳ There are few others, But these are Core.

PK Student_id	name	major
1	Jack	Bio
2	Kate	Sociology
3	Claire	Eng
4	Jack	Bio
5	Mike	Chemistry

```
CREATE TABLE student (          ← any name for DB
    student_id INT PRIMARY KEY,     ← Datatype
    name VARCHAR(20) NOT NULL,
    major VARCHAR(20) UNIQUE,
);
```

(or)

```
PRIMARY KEY (student_id)
```

DESCRIBE student; (To show in the output as table)

Delete : DROP TABLE Student ;

↳ will delete the Table

Add another Column :

ALTER TABLE Student ADD gpa DECIMAL(3,2);

delete the Column :

ALTER TABLE Student DROP COLUMN gpa ;

Inserting Data :-

```
CREATE TABLE Student (
    student_id int ~~PRI~~,
    name VARCHAR(20),
    major VARCHAR(20)
    PRIMARY KEY (student_id)
);
```

SELECT * FROM STUDENT ; (TO Show info in output table in o/p)

1) INSERT INTO Student VALUES(1, 'Jack', 'Biology');

2) INSERT INTO " " (2, 'kate', 'Sociology');

3) " " student(student_id, name) ~~VALUES~~ (3, Claire)

4) " " " (") VALUES (4, Claire)

o/p

student_id	name	major
1	Jack	Biology
2	kate	Sociology
3	Claire	NULL
4	Claire	NULL

Constraints:

NOT NULL - This row can't be null

UNIQUE - It should be unique.

See first (create).

default value,

Insert into student (student_id, name) values (1, Tao)

major VARCHAR(20) DEFAULT 'undecided'. ↳ Then only it can work

increase the table number or primary key number automatically

student_ID INT AUTO_INCREMENT

update and delete:

comparesor operators:

= equals

<> not equals

>

<

>=

<=

```
UPDATE Student
SET Major = 'Bio'
1) WHERE major = 'Biology';
2) WHERE major = 'Bio' OR major = 'chemistry';

SELECT * FROM student;
DELETE FROM Student
WHERE name = 'TOM' AND major = 'undecided';
```

Alter to NOT NULL and unique;

```
ALTER TABLE CUSTOMERS
MODIFY S-NO INT NOT NULL;
```

Basic Queries :-

SELECT - ask's the database to give some info.

(*) grab all the information

```
SELECT  name
From Student
ORDER By, name DESC;
```

↳ Alphabetical order

ASC - Ascending

1
2
3
4
5

```
order by major;
```

```
SELECT *
FROM STUDENT
```

1) ORDER BY major, student_id;

↳ if any have same major order by student_id)

2) LIMIT 2;

↳ limits the rows to 2.

```
SELECT *
FROM STUDENT
ORDER BY student_id DESC
LIMIT 2;
```

O/P
(Print last 2)

only return Specific major:

```
SELECT *
FROM Student
WHERE major = 'Biology';
```

Comment (--)

Select Specific names:

```
SELECT *
FROM Student
WHERE name IN ("Claire', 'Kate', 'Mike');
```

```
SELELT *
FROM Student
WHERE major IN ('Biology', 'chemistry') AND 'Student.id > 2;
```

S.NO	Name	Age	gender
1	Belson	21	male
2	Sakthi	20	male
3	Surya	20	male
4	Gunal	20	male
5	Dinesh	20	

FOREIGN KEY (mgr_id)
REFERENCES employee(emp_id)
ON DELETE SET NULL

ON DELETE CASCADE

(c) for two foreign key

More Basic Queries :- order by salary

1)
```
SELECT *
FROM employee
ORDER BY salary DESC ASC ;
```

FIRST 5 EMPLOYEE.

```
LIMIT 5;
```

2) Name only Query

```
SELECT FIRST_NAME, last_name
FROM employee;
```

3) Change name of first_name & last_name.

```
SELECT First_name AS forename, last_name AS surname
From employee;
```

4) Find out all the genders;

```
SELECT DISTINCT sex
FROM employee;
```

→ column name

→ shows distinct things

Functions:

1) Find number of employees;

```
SELECT COUNT(emp_id)
FROM employee;
```

2) Number of female employees born after 1970

```
SELECT COUNT (emp_id)
FROM employee
WHERE sex = 'F' AND BIRTH_DATE > '1970-01-01';
```

3) Average of all employee

```
SELECT AVG (salary)
FROM employee;
```

for male

```
WHERE sex = 'M';
```

(Aggregations)

Sum
Count
Average.

Aggregation: (4)

4) Sum of all employee's salary:

```
SELECT SUM(salary)
FROM employee;
```

5) How many male & How many female

```
SELECT COUNT(sex), sex
FROM employee
GROUP BY sex;
```

6) Total Sales of each salesman

```
SELECT SUM(total_sales), emp_id
FROM works_with
GROUP BY emp_id;
```

wildcards :

% = any number of characters

_ = one character

→ any thing can come before it but end should be this %.___

1) Find clients who are llc

```
SELECT *
FROM client
WHERE client_name LIKE '%LLC';
```

2) Branch Suppliers who are label business

```
SELECT *
FROM branch_supplier
WHERE supplier_name LIKE '%Label%';
```

anywhere

3) employee in october month

```
SELECT *
FROM employee
WHERE birth_date LIKE '____-10-__';
```

4) clients who are schools

```
SELECT *
FROM client
WHERE client_name LIKE '%school%';
```

UNION: (special sql operator used to combine the results multiple selects statements into one)

1) List of Branch and employee names:

```
SELECT first_name
FROM employee
UNION
SELECT branch_name
FROM branch;
```

Rules:
1) both should have only one column.
2) only work with similar data types

—x—

```
SELECT first_name AS Company_Names
FROM employee
UNION
SELECT branch_name
FROM branch
UNION
SELECT client_name
FROM client;
```

1) Find list of all clients & branch suppliers names

```
SELECT client_name , client.branch_id
FROM client
UNION
SELECT supplier_name, branch_id
FROM branch_supplier;
```

2) List of all money spent or earned by the company

```
SELECT salary
FROM employee
UNION
SELECT total_sales
FROM works_with;
```

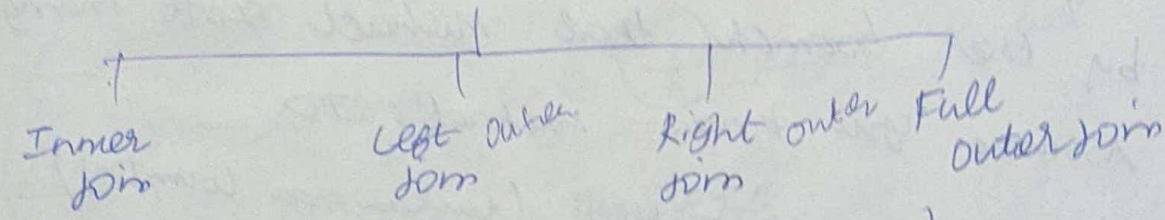
JOINS :

1) FIND all branches ~~emp-id~~ names of their managers.

```
SELECT employee.emp_id, employee.first_name, branch.
                                              branch_name
FROM employee → left
JOIN branch
ON employee.emp_id = branch.mgr_id;
```

emp_id	FN	LN
100	David	Wallace
102	Michael	Scarn
106	Josh	Stanton

Types of Joins



↳ We can't do it in MySql

select → write what we wanna print

first table name

inner join second table (on) common column = both table

Inner join

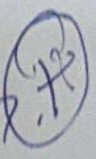
select column_name(s)

From table1

Inner join table2

on table.column_name = table2.column_name;

only change key word.
Left, right



prefer w3schools

Nested Queries :

1) Find names of all employees who have sold over 30,000 to a single client

```
SELECT employee.first_name, employee.last_name
FROM employee
WHERE employee.emp_id IN (
    SELECT works_with.emp_id
    FROM works_with
    WHERE works_with.total_sales > 30000
);
```

1) Inner will be executed first
2) outer second.

2) Find all clients who are handled by the branch that Michael Scott manges

Assume you know Michael's ID.

102 - 401 (Lackawanna country)
102 - 406 (Fed Ex)

400 | 404
401 | 406

```
SELECT branch.branch_id
FROM branch
WHERE branch.branch_id = 102;
```

```
SELECT client.client_name
FROM client
WHERE client.branch_id = (
    SELECT branch.branch_id
    FROM branch
    WHERE branch.mgr_id = 102
    LIMIT 1
);
```

use limit when we use "="

Scott can be manager in many branch so, limit is to 1

on delete:

1) on delete set null
2) on delete set cascade

Branch table code (+) Refer video

```
DELETE FROM employee
WHERE emp_id = 102;
```

@ after delete it will show null

— x —

```
DELETE FROM branch
WHERE branch_id = 2;
```

Use it when Foreign key is also a primary key

delete entire row (+)

If it is a primary key we use cascade (+)

(+)

Triggers :

Block of SQL code, which we write which will define a certain action that should happen when a certain action performed in database.

Delimiter is used to end the code.

Ex:

```
DELIMITER $$
CREATE PROCEDURE
    TRIGGER my_trigger BEFORE INSERT
    ON employee
    FOR EACH ROW BEGIN
        INSERT INTO trigger_test values
    END $$
DELIMITER ;
```

1st line

2nd line ('added new employee')

we should use it in command line client (command prompt) not in text editor

Create a trigger :

```
CREATE TABLE trigger_test (
    message VARCHAR(100)
);
```

```
INSERT INTO employee
VALUES ( 109, 'oscar', 'Martinez', D. M, 6 — )

SELECT * FROM trigger_test;
```

2) TRIGGER to show name:-

```
DELIMITER $$
CREATE
    TRIGGER my-trigger1 BEFORE INSERT
    ON employee
    FOR EACH ROW BEGIN
        INSERT INTO trigger_test values(NEW.firstname)

    END $$
DELIMITER ;
```

o/p

added new employe
Kevin

```
DELIMITER $$
    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
    //      values ('added female employee
```

```
ELSE
    INSERT INTO ll VALUES ('added other employee');
END IF;
END $$
DELIMITER ;
```

We can also make triggers for update, insert, delete (+).
and also After in place of Before in code.

—— X ——