# SQL Tutorial

Total - 4:20:39

- Freecodecamp

## *Introduction :

- RDBMS is software used to create, manage databases. (we will use POPSQL)
- MySql is a database which interacts with RDBMS

→ schema is called to different tables and relations.

## *What is a database ? :

## *Tables and keys :

- primary key is very important

* Q = what is seroqate key ?
  Ans = A number which does not have any connection to the real world.

→ SSN = Social Security Number

## *SQL Basics :

## * Creating Tables :

most common data types (there are more)

- INT → Whole Numbers
- DECIMAL (M,N) → Decimal Numbers - Exact value
- VARCHAR (l) → String of text of length l (string of 5 characters) VARCHAR(5)
- BLOB → Binary Large Object, Stores large data.
- DATE → 'YYYY-MM-DD'
- TIMESTAMP → 'YYYY-MM-DD HH:MM:SS' used for recording

DECIMAL (23, 59)

23.59
M N

→ most of the people stores images, videos etc. in BLOB

→ To write SQL codes in capital letters is a good practice, by this we can distinguis b/w SQL codes and other words.

→ you have to write ";" after every sql command

Task :

| student_id | name | major |
|---|---|---|
| 1 | Jack | Biology |
| 2 | Kate | Sociology |
| 3 | Claire | English |
| 4 | Jack | Biology |
| 5 | Mike | Comp. Sci |

It is the primary key in this table

```
CREATE TABLE student(
    student_id INT PRIMARY KEY,
    name VARCHAR(20),
    major VARCHAR(20)
);
```

→ you can also declare primary key in second line

```
student_id INT,
PRIMARY KEY(student_id)
```

- `DESCRIBE student;` → to get description of the table
- `DROP TABLE student;` → to delete the table

- `ALTER TABLE student ADD gpa DECIMAL(3,2);`

  ↳ to add new column of gpa in the table

- `ALTER TABLE student DROP COLUMN gpa;`

  ↳ to delete the gpa column from table

## * Inserting Data :

```
SELECT * FROM student;
INSERT INTO student VALUES(1, 'Jack', 'Biology');
INSERT INTO student VALUES(2, 'kate', 'Sociology');
.
.
.
(insert one by one)
```

↓ remember to maintain order of student_id name and major.

OR,

- `INSERT INTO student (student_id, name) VALUES (2, 'kate');`

  ↳ if you don't want to add data in 'major' column

→ Primary key cannot be same for 2 different rows or more, it has to be unique.

## * Constraints :

```
name  VARCHAR(20)  NOT NULL,
major VARCHAR(20)  UNIQUE,
```

NOT NULL → means this field cannot be null

UNIQUE → means this field has to be unique

```
INSERT INTO student VALUES (3, NULL, 'Chemistry');
```

↳ error, because of the

→ NOT NULL and UNIQUE are constraints

→ DEFAULT is a constraint

- ```
  major VARCHAR(20) DEFAULT 'undecided',
  ```

↳ if ~~major~~ you don't give any value to major, then by default it will write 'undecided'

```
INSERT INTO student (student_id, name) VALUES (2, 'kate');
```

↳ in this case, major is not given so here in place of major, 'undecided' will be written because of this.

- ```
  student_id INT AUTO_INCREMENT,
  PRIMARY KEY (student_id),
  ```

↳ now, you don't have to write primary key i.e. student_id by yourself, it will take it automatically and will increment it.

```
INSERT INTO student (name, major) VALUES ('Jack', 'Biology');
```

*Update & Delete :

```
SELECT * FROM student;
UPDATE student
SET major = 'Bio'
WHERE major = 'Biology';
```

→ if you don't write this line, then all the majors will become "Bio"

```
UPDATE student
SET major = 'Comp Sci'
WHERE student_id = 4;
```

→ you can also write ">" or "<" instead of "=".

```
UPDATEE student
SET major = 'Biochemistry'
WHERE major = 'Bio' OR major = 'Chemistry';
```

↳ it will update major as 'Biochemistry' where is Bio or chemistry.

```
UPDATE student
SET name = 'Tom', major = 'undecided'
WHERE student_id = 1;
```

```
UPDATE student
SET major = 'undecided';
```

→ all the majors will have value 'undecided'

```
DELETE FROM student
WHERE student_id = 5;
```

→ the row will be deleted where student_id = 5.

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

↳ the row will be deleted where name is Tom and major is undecided

```
DELETE FROM student;
```

→ to delete every row from the table.

## * Basic Queries :

```
SELECT *
FROM student;
```

↳ to get all the columns from student table

```
Select name
FROM student;
```

↳ to get only name columns from student table.

```
Select name, major
FROM student;
```

or,

```
SELECT student.name, student.major
FROM student;
```

```
SELECT student.name, student.major
FROM student
ORDER BY name;
```

↳ to order the columns by name of alphabetical order (by default it will be in Ascending order)

```
SELECT student.name, student.major
FROM student
ORDER BY name DESC;
```

↳ now it will be ordered in descending order of alphabetical order

```
ORDER BY student_id DESC;
```

→ to order by student_id in descending order

```
ORDER BY student-id ASC;
```

↳ to order in Ascending order

```
ORDER BY major, student_id;
```
→ it will order by major first and if there are 2 rows with same major then it will order those 2 by their student_id

```
SELECT *
FROM student
LIMIT 2;
```
→ it will give only 2 rows

```
SELECT *
FROM student
ORDER BY student_id DESC
LIMIT 2;
```
→ it will only show 2 rows

```
Select *
FROM student
WHERE major = 'Biology';
```

```
Select name, major
FROM student
WHERE major = 'chemistry' OR major = 'Biology';
                          OR name = 'kate';
```
→ to show only name, major

→ you can also use <, >, <=, >=, =, <>, AND, OR

<> ↓ Not equal to sign

```
WHERE student_id <3;
```

```
WHERE student_id <=3 AND name <> 'Jack';
```

```
Where name IN ('Clare', 'Kate');
```
→ it will select where name is Clare and Kate

```
WHERE major IN ('Biology', 'Chemistry')
```

```
WHERE major IN('Biology', 'Chemistry') AND student_id>2;
```

## *Company Database Intro :

→ super_id and branch_id are boriegn keys in this table

→ green cell is for boriegn key and red cell is for primary key.

## *Creating Company Database :

→ you can have more than one primary and boriegn key.

## * More Basic Queries :

```
SELECT * FROM employees;
```

```
--Find all employees ordered by salary
SELECT *
FROM employee
ORDER BY salary;
```

```
--Find all employees ordered by sex then name
SELECT *
FROM employee
ORDER BY sex, first_name, last_name;
```

```
--Find the first 5 employees in the table
SELECT *
FROM employee
LIMIT 5;
```

```
--Find the first and last name of all employees
SELECT first_name, last_name
FROM employee;
```

```
--Find the forename and surnames of all employees
SELECT first_name AS forename, last_name AS surname
FROM employee;
```

↳ first_name and last_name will be printed with name forename and surname

```
--Find out all the different genders
SELECT DISTINCT sex
FROM employee
```

or SELECT DISTINCT branch_id

## *Functions (SQL Functions) :

```
--Find the number of employees
SELECT COUNT(emp_id)
FROM employee;
```

or SELECT COUNT(super_id)

↳ output : 9

```
--Find the number of female employees born after 1970
SELECT COUNT(emp_id)
FROM employee
WHERE sex = 'F' AND birth_date > '1971-01-01';
```

↳ output : 2

```
--Find the average of all employee's salaries
SELECT AVG(salary)
FROM employee;
```

↳ output : 92888.8889

or,

```
SELECT AVG (salary)
FROM employee
WHERE sex = 'M';
```

→ average salary of Male employees

```
--Find the sum of all employee's salaries
SELECT SUM (salary)
FROM employee;
```

→ 836000

```
--Find out how many males and females there are
SELECT COUNT (sex), sex
FROM employee
GROUP BY sex;
```

→ this is called aggregation

| COUNT(sex) | sex |
|---|---|
| 3 | F |
| 6 | M |

```
--Find the total sales of each salesman
SELECT SUM(total_sales), client_id
FROM works_with
GROUP BY client_id
```

*Wildcards:

→ LLC = Limited Liable Company

```
--Find any client's who are in LLC
SELECT *
FROM client
WHERE client_name LIKE '%LLC';
```

→ % = any character, _ = one character

Wildcards

```
--Find any branch suppliers who are in the
  label business
SELECT *
FROM branch_supplier
WHERE supplier_name LIKE '%Label%';
```

```
--Find any employee born in October
SELECT *
FROM employee
WHERE birth_date LIKE '____-10%';
```

Skip it will ~~wait~~ 4 words then this

```
--Find any clients who are schools
SELECT *
FROM employee
WHERE client_name LIKE '%school%';
```

**Union :** (It is a special SQL operator)

```
--Find a list of employee and branch names
SELECT first_name
FROM employee
UNION
SELECT branch_name
FROM branch
UNION
SELECT client_name
FROM client;
```

make sure that data type and no. of columns should be same

```
SELECT first_name AS company_name
```

→ then column name will be company_name instead of first_name

```
--Find a list of all clients & branch suppliers names
SELECT client_name, branch_id
FROM client
UNION
SELECT supplier_name, branch_id
FROM branch_suppliers;
```

→ or, `SELECT client_name, client.branch_id`

→ or, `SELECT supplier_name, branch_supplier.branch_id`

```
--Find a list of all money spent or earned by the
  company
SELECT salary
FROM employee
UNION
SELECT total_sales
FROM works_with;
```

*Joins (It is used to combine 2 tables by the relation of one column)

```
--Find all branches and the names of the managers
SELECT employee.emp_id, employee.first_name, branch.branch
                                              _name
FROM employee
JOIN branch
ON employee.emp_id = branch.mgr_id
```

→ because emp_id and mgr_id are same

→ it will give only those employees who are managers

→ or `LEFT JOIN branch`

it will give all rows from left table

→ it will give all employees and will write "NULL" who are not managers.

or, RIGHT JOIN branch

↳ it will match [redacted] from right table and will give all rows from right table (all branch_name)

## *Nested Queries:

```
--Find names of all employees who have sold over
--30000 to a single client
[redacted]
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
);
```

```
--Find all clients who are handled by the branch that
--Michael Scott manages, Assume that you know
--Michael's ID
SELECT client.client_name
FROM client
WHERE client.branch_id = (
    SELECT branch.branch_id
    FROM branch
    WHERE branch.mgr_id = 102
    LIMIT 1
);
```

↳ it will run from bottom to up

## *ON Delete :

```
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
);
```

When emp_id is deleted, "Null" will be writen in place of mgr_id.

```
DELETE FROM employee
WHERE emp_id = 102;

SELECT * from branch;
```

to delete

- ON delete cascade

```
CREATE TABLE branch_supplier(
    branch_id INT,
    supplier_name VARCHAR(40),
    supply_type VARCHAR(40),
    PRIMARY KEY(branch_id, supplier_name),
    FOREIGN KEY(branch_id) REFERENCES branch(branch_id)
                ON DELETE CASCADE
);

DELETE FROM branch
WHERE branch_id = 2;

SELECT * FROM branch_supplier;
```

*Triggers :

mysql > use giraffe → database name

→ $$ and ; are some example of mysql delimiters

→ you have to write trigger codes in mysql terminal only.

*Into to ER Diagram :

→ underline primary key in ER Diagram

*Designing an ER Diagram :

*Converting ER Diagrams into Schemas :

*Triggers code :

```
mysql> DELIMITER $$
mysql> CREATE
        TRIGGER my_trigger BEFORE INSERT
        ON employee
        FOR EACH ROW BEGIN
            INSERT INTO trigger_test VALUES('added
                                    new employee');
        END$$
mysql> DELIMITER;
```

* 
```
mysql> DELIMITER $$
mysql> 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$$$
```

```
mysql> DELIMITER ;
```

## HackerRank Practice

*Q= Query a list of CITY names from STATION table for cities that have an even ID number. Print in any order, but exclude duplicates from the answer.

Ans=
```
SELECT DISTINCT (CITY) FROM STATION
WHERE (ID%2) =0;
```

*Q= Find the difference b/w the total number of CITY entries in the table and the number of distinct CITY entries in the table.

Ans=
```
SELECT COUNT(CITY) - COUNT(DISTINCT CITY)
FROM STATION;
```