
SQL
Introduction
to SQL

What is SQL?

SQL is a standard language designed for managing data in relational database management system.

SQL is a standard programming language specifically designed for storing, retrieving, managing or manipulating the data inside a relational database management system (RDBMS). SQL became an ISO standard in 1987.

What is SQL?

SQL is the most widely-implemented database language and supported by the popular relational database systems:

MySQL,

SQL Server,

Oracle.

What You Can Do with SQL

- You can create a database.
- You can create tables in a database.
- You can query or request information from a database.
- You can insert records in a database.
- You can update or modify records in a database.
- You can delete records from the database.
- You can set permissions or access control within the database for data security.
- You can create views to avoid typing frequently used complex queries.

What is Relational Database

A relational database is a database divided into logical units called tables, where tables are related to one another within the database

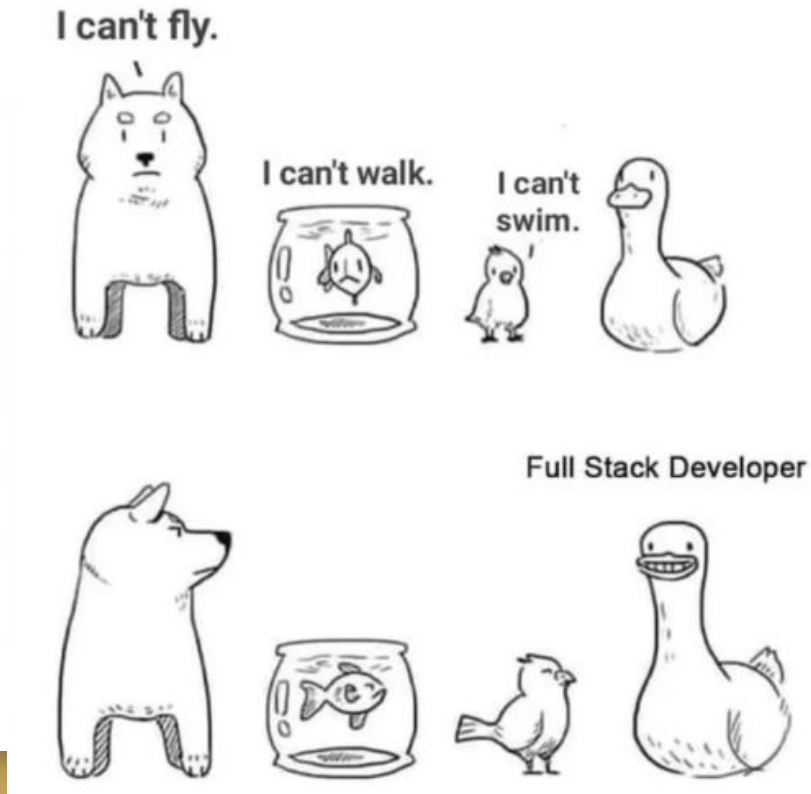
Setting Up Work Environment for Practicing SQL

https://dev.mysql.com/downloads/mysql/

https://www.microsoft.com/en-in/download/details.aspx?id=30438.

http://www.wampserver.com/en/

SQL Svntax



SQL Statements

An SQL statement is composed of a sequence of keywords, identifiers, etc. terminated by a semicolon (;).

SQL

```
SELECT emp_name, hire_date, salary FROM employees WHERE salary > 5000;
```

```
1  SELECT emp_name, hire_date, salary
2  FROM employees
3  WHERE salary > 5000;
```

SQL

Use semicolon at the end of an SQL statement — it terminates the statement or submits the statement to the database server

Case Sensitivity in SQL

```
SELECT emp_name, hire_date, salary FROM employees;

select  emp_name, hire_date, salary FROM employees;
```

SQL Comments

A comment is simply a text that is ignored by the database engine. Comments can be used to provide a quick hint about the SQL statement.

```sql
-- Select all the employees
SELECT * FROM employees;
```

SQL CREATE DATABASE Statement

Creating a Database

Syntax

```
CREATE DATABASE database_name;
```

Selecting the Database

```
mysql> USE demo;

mysql> SHOW databases;
```

SQL CREATE TABLE Statement

Syntax

```
CREATE TABLE table_name (
    column1_name data_type constraints,
    column2_name data_type constraints,
    ....
);
```

Creating a Table

```
CREATE TABLE persons (
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL,
    birth_date DATE,
    phone VARCHAR(15) NOT NULL UNIQUE
);
```

Data types

`INT`	Stores numeric values in the range of -2147483648 to 2147483647
`DECIMAL`	Stores decimal values with exact precision.
`CHAR`	Stores fixed-length strings with a maximum size of 255 characters.
`VARCHAR`	Stores variable-length strings with a maximum size of 65,535 characters.
`TEXT`	Stores strings with a maximum size of 65,535 characters.
`DATE`	Stores date values in the YYYY-MM-DD format.
`DATETIME`	Stores combined date/time values in the YYYY-MM-DD HH:MM:SS format.
`TIMESTAMP`	Stores timestamp values. `TIMESTAMP` values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:01' UTC).

Describe Table

DESC table_name

Create Table If Not Exists

```
CREATE TABLE IF NOT EXISTS persons (
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL,
    birth_date DATE,
    phone VARCHAR(15) NOT NULL UNIQUE
);
```

Show Table

SHOW TABLES

SQL Constraints

SQL constraints are used to specify rules for the data in a table.

The following constraints are commonly used in SQL:

- `NOT NULL` - Ensures that a column cannot have a NULL value
- `UNIQUE` - Ensures that all values in a column are different
- `PRIMARY KEY` - A combination of a `NOT NULL` and `UNIQUE`. Uniquely identifies each row in a table
- `FOREIGN KEY` - Prevents actions that would destroy links between tables
- `CHECK` - Ensures that the values in a column satisfies a specific condition
- `DEFAULT` - Sets a default value for a column if no value is specified
- `CREATE INDEX` - Used to create and retrieve data from the database very quickly
-

NOT NULL Constraint

The `NOT NULL` constraint specifies that the column does not accept `NULL` values.

This means if `NOT NULL` constraint is applied on a column then you cannot insert a new row in the table without adding a non-NULL value for that column.

NOT NULL Constraint

```
CREATE TABLE persons (
    id INT NOT NULL,
    name VARCHAR(30) NOT NULL,
    birth_date DATE,
    phone VARCHAR(15) NOT NULL
);
```

0 vs NULL

PRIMARY KEY Constraint

The `PRIMARY KEY` constraint identify the column or set of columns that have values that uniquely identify a row in a table. No two rows in a table can have the same primary key value. Also, you cannot enter `NULL` value in a primary key column.

PRIMARY KEY Constraint

```
CREATE TABLE persons (
    id INT NOT NULL PRIMARY KEY,
    name VARCHAR(30) NOT NULL,
    birth_date DATE,
    phone VARCHAR(15) NOT NULL
);
```

UNIQUE Constraint

The `UNIQUE` constraint restricts one or more columns to contain unique values within a table.

Although both a `UNIQUE` constraint and a `PRIMARY KEY` constraint enforce uniqueness, use a `UNIQUE` constraint instead of a `PRIMARY KEY` constraint when you want to enforce the uniqueness of a column, or combination of columns, that is not the primary key.

UNIQUE Constraint

```
CREATE TABLE persons (
    id INT NOT NULL PRIMARY KEY,
    name VARCHAR(30) NOT NULL,
    birth_date DATE,
    phone VARCHAR(15) NOT NULL UNIQUE
);
```

DEFAULT Constraint

The `DEFAULT` constraint specifies the default value for the columns.

A column default is some value that will be inserted in the column by the database engine when an `INSERT` statement doesn't explicitly assign a particular value

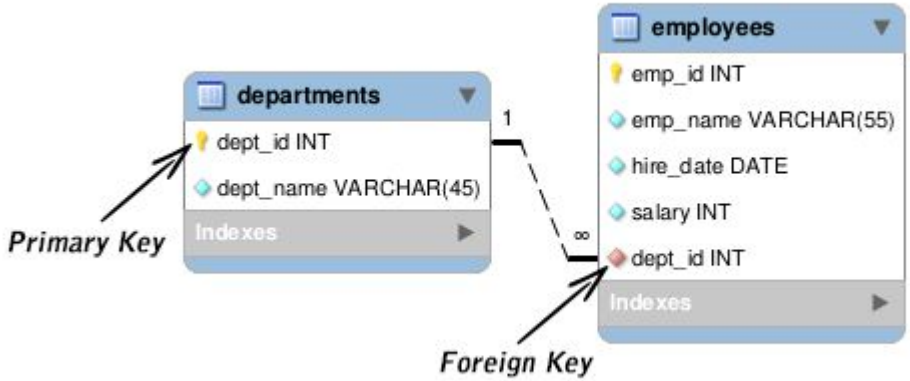
DEFAULT Constraint

```
CREATE TABLE persons (
    id INT NOT NULL PRIMARY KEY,
    name VARCHAR(30) NOT NULL,
    birth_date DATE,
    phone VARCHAR(15) NOT NULL UNIQUE,
    country VARCHAR(30) NOT NULL DEFAULT 'Australia'
);
```

FOREIGN KEY Constraint

A foreign key (FK) is a column or combination of columns that is used to establish and enforce a relationship between the data in two tables.

FOREIGN KEY Constraint



FOREIGN KEY Constraint

```
CREATE TABLE employees (
    emp_id INT NOT NULL PRIMARY KEY,
    emp_name VARCHAR(55) NOT NULL,
    hire_date DATE NOT NULL,
    salary INT,
    dept_id INT,
    FOREIGN KEY (dept_id) REFERENCES departments(dept_id)
);
```

CHECK Constraint

The `CHECK` constraint is used to restrict the values that can be placed in a column.

CHECK Constraint

```
CREATE TABLE employees (
    emp_id INT NOT NULL PRIMARY KEY,
    emp_name VARCHAR(55) NOT NULL,
    hire_date DATE NOT NULL,
    salary INT NOT NULL CHECK (salary >= 3000 AND salary <= 10000),
    dept_id INT,
    FOREIGN KEY (dept_id) REFERENCES departments(dept_id)
);
```

SQL INSERT Statement

Syntax

```
INSERT INTO table_name (column1,column2,...) VALUES (value1,value2,...);
```

Step 1: View Table Structure

```
mysql> DESCRIBE persons;
```

Step 2: Adding Records to a Table

```
INSERT INTO persons (name, birth_date, phone)

VALUES ('Peter Wilson', '1990-07-15', '0711-020361');
```

Adding Records to a Table

```
INSERT INTO persons (name, birth_date, phone)

VALUES ('Carrie Simpson', '1995-05-01', '0251-031259');
```

SQL SELECT Statement

The `SELECT` statement is used to select or retrieve the data from one or more tables. You can use this statement to retrieve all the rows from a table in one go, as well as to retrieve only those rows that satisfy a certain condition or a combination of conditions.

Syntax

```
SELECT column1_name, column2_name, columnN_name FROM table_name;
```

Select All from Table

```
SELECT * FROM table_name;
```

```
SELECT * FROM employees;
```

Select Columns from Table

```
SELECT emp_id, emp_name, hire_date, salary

FROM employees;
```

SQL WHERE Clause

The `WHERE` clause is used with the `SELECT`, `UPDATE`, and `DELETE`. However, you'll see the use of this clause with other statements in upcoming chapters.

Syntax

```
SELECT column_list FROM table_name WHERE condition;
```

```
SELECT * FROM table_name WHERE condition;
```

```
SELECT * FROM table_name WHERE condition;
```

Filter Records with WHERE Clause

```
SELECT * FROM employees
WHERE salary > 7000;
```

==========

```
SELECT emp_id, emp_name, hire_date, salary
FROM employees
WHERE salary > 7000;
```

```sql
SELECT * FROM employees

WHERE emp_id = 2;
```

Operators Allowed in WHERE Clause

=	Equal	`WHERE id = 2`
>	Greater than	`WHERE age > 30`
<	Less than	`WHERE age < 18`
>=	Greater than or equal	`WHERE rating >= 4`
<=	Less than or equal	`WHERE price <= 100`
LIKE	Simple pattern matching	`WHERE name LIKE 'Dav'`
IN	Check whether a specified value matches any value in a list or subquery	`WHERE country IN ('USA', 'UK')`
BETWEEN	Check whether a specified value is within a range of values	`WHERE rating BETWEEN 3 AND 5`

https://github.com/datacharmer/test_db

https://dev.mysql.com/doc/index-other.html

SQL AND & OR Operators

The AND Operator

```
SELECT column1_name, column2_name, columnN_name
FROM table_name
WHERE condition1 AND condition2;
```

Using WHERE Clause with AND Operator

```
SELECT * FROM employees

WHERE salary > 7000 AND dept_id = 5;
```

The OR Operator

the `OR` operator is also a logical operator that combines two conditions, but it returns `TRUE` when either of the conditions is `TRUE`.

The OR Operator

```sql
SELECT * FROM employees
WHERE salary > 7000 OR dept_id = 5;
```

Combining AND & OR Operator

Combining AND & OR Operator

```
SELECT * FROM employees

WHERE salary > 5000 AND (dept_id = 1 OR dept_id = 5);
```

SQL IN & BETWEEN Operators

The IN Operator

```
SELECT column_list FROM table_name

WHERE column_name IN (value1, value1,...);
```

example

```
SELECT * FROM employees

WHERE dept_id IN (1, 3);
```

The BETWEEN Operator

```
SELECT column1_name, column2_name, columnN_name
FROM table_name
WHERE column_name BETWEEN min_value AND max_value;
```

Define Numeric Ranges

```
SELECT * FROM employees

WHERE salary BETWEEN 7000 AND 9000;
```

Define Date Ranges

When using the `BETWEEN` operator with date or time values, use the `CAST()` function to explicitly convert the values to the desired data type for best results

```
SELECT * FROM employees WHERE hire_date

BETWEEN CAST('2006-01-01' AS DATE) AND CAST('2016-12-31' AS DATE);
```

Define String Ranges

```
SELECT * FROM employees

WHERE emp_name BETWEEN 'O' AND 'Z';
```

SQL ORDER BY Clause

Syntax

```
SELECT column_list FROM table_name ORDER BY column_name
ASC|DESC;
```

The `ORDER BY` clause is used to sort the data returned by a query in ascending or descending orde

Sorting Single Column

```
SELECT * FROM employees
ORDER BY emp_name ASC;
```

```
SELECT * FROM employees
ORDER BY emp_name;
```

DESC

```
SELECT * FROM employees

ORDER BY salary DESC;
```

Sorting Multiple Columns

```
SELECT * FROM trainees
ORDER BY first_name;
```

```
SELECT * FROM trainees
ORDER BY first_name, last_name;
```

SQL TOP / MySQL LIMIT Clause

SQL TOP Syntax

```
SELECT TOP number | percent column_list FROM table_name;
```

Example

emp_id	emp_name	hire_date	salary	dept_id
1	Ethan Hunt	2001-05-01	5000	4
2	Tony Montana	2002-07-15	6500	1
3	Sarah Connor	2005-10-18	8000	5
4	Rick Deckard	2007-01-03	7200	3
5	Martin Blank	2008-06-24	5600	NULL

```
SELECT TOP 3 * FROM employees

ORDER BY salary DESC;
```

emp_id	emp_name	hire_date	salary	dept_id
3	Sarah Connor	2005-10-18	8000	5
4	Rick Deckard	2007-01-03	7200	3
2	Tony Montana	2002-07-15	6500	1

MySQL LIMIT Syntax

```
SELECT column_list FROM table_name LIMIT number;
```

==========

```
SELECT * FROM employees
ORDER BY salary DESC LIMIT 3;
```

SQL DISTINCT Clause

Retrieving Distinct Values

Syntax

```
SELECT DISTINCT column_list FROM table_name;
```

Example

cust_id	cust_name	city	postal_code
1	Maria Anders	Berlin	12209
2	Fran Wilson	Madrid	28023
3	Dominique Perrier	Paris	75016
4	Martin Blank	Turin	10100
5	Thomas Hardy	Portland	97219
6	Christina Aguilera	Madrid	28001

```sql
SELECT city FROM customers;
```

```
+-----------+
| city      |
+-----------+
| Berlin    |
| Madrid    |
| Paris     |
| Turin     |
| Portland  |
| Madrid    |
+-----------+
```

Removing Duplicate Data

```
SELECT DISTINCT city FROM customers;
```

```
+-----------+
| city      |
+-----------+
| Berlin    |
| Madrid    |
| Paris     |
| Turin     |
| Portland  |
+-----------+
```

SQL UPDATE Statement

update records in a database table using SQL.

Syntax

```
UPDATE table_name
SET column1_name = value1, column2_name = value2,...
WHERE condition;
```

Warning

Warning: The `WHERE` clause in the `UPDATE` statement specifies which record or records should be updated. If you omit the `WHERE` clause, all the records will be updated.

example

emp_id	emp_name	hire_date	salary	dept_id
1	Ethan Hunt	2001-05-01	5000	1
2	Tony Montana	2002-07-15	6500	5
3	Sarah Connor	2005-10-18	8000	3
4	Rick Deckard	2007-01-03	7200	4
5	Martin Blank	2008-06-24	5600	NULL

Updating a Single Column

```
UPDATE employees SET emp_name = 'Sarah Ann Connor'

WHERE emp_id = 3;
```

emp_id	emp_name	hire_date	salary	dept_id
1	Ethan Hunt	2001-05-01	5000	1
2	Tony Montana	2002-07-15	6500	5
3	Sarah Ann Connor	2005-10-18	8000	3
4	Rick Deckard	2007-01-03	7200	4
5	Martin Blank	2008-06-24	5600	NULL

Updating Multiple Columns

```
UPDATE employees
SET salary = 6000, dept_id = 2
WHERE emp_id = 5;
```

SQL DELETE Statement

Deleting Data from Tables

Syntax

```
DELETE FROM table_name WHERE condition;
```

Warning: The `WHERE` clause in the `DELETE` statement specifies which record or records should be deleted. It is however optional, but if you omit or forget the `WHERE` clause, all the records will be deleted permanently from the table.

SQL TRUNCATE TABLE Statement

The `TRUNCATE TABLE` statement removes all the rows from a table more quickly than a `DELETE`. Logically, `TRUNCATE TABLE` is similar to the `DELETE` statement with no `WHERE` clause.

Syntax

```
TRUNCATE TABLE table_name;
```

```
TRUNCATE TABLE employees;
```

TRUNCATE TABLE vs DELETE

- `TRUNCATE TABLE` statement drop and re-create the table in such a way that any auto-increment value is reset to its start value which is generally 1.

- `DELETE` lets you filter which rows to be deleted based upon an optional `WHERE` clause, whereas `TRUNCATE TABLE` doesn't support `WHERE` clause it just removes all the rows.

- `TRUNCATE TABLE` is faster and uses fewer system resources than `DELETE`, because `DELETE` scans the table to generate a count of rows that were affected then delete the rows one by one and records an entry in the database log for each deleted row, while `TRUNCATE TABLE` just delete all the rows without providing any additional information.

SQL DROP Statement

The `DROP TABLE` statement permanently erase all data from the table, as well as the metadata that defines the table in the data dictionary.

Syntax

```
DROP TABLE table1_name, table2_name, ...;
```

Warning: Dropping a database or table is irreversible. So, be careful while using the `DROP` statement, because database system generally do not display any alert like "Are you sure?". It will immediately delete the database or table and all of its data.

Removing Database

```
DROP DATABASE demo;
```

SQL LIKE Operator

Pattern Matching

```
SELECT * FROM employees
WHERE  BINARY emp_name LIKE 'S%';
```

Statement	Meaning	Values Returned
`WHERE name LIKE 'Da%'`	Find names beginning with *Da*	David, Davidson
`WHERE name LIKE '%th'`	Find names ending with *th*	Elizabeth, Smith
`WHERE name LIKE '%on%'`	Find names containing the *on*	Davidson, Toni
`WHERE name LIKE 'Sa_'`	Find names beginning with *Sa* and is followed by at most one character	Sam
`WHERE name LIKE '_oy'`	Find names ending with *oy* and is preceded by at most one character	Joy, Roy

`WHERE name LIKE '_an_'`	Find names containing *an* and begins and ends with at most one character	Dana, Hans
`WHERE name LIKE '%ar_'`	Find names containing *ar*, begins with any number of characters, and ends with at most one character	Richard, Karl
`WHERE name LIKE '_ar%'`	Find names containing *ar*, begins with at most one character, and ends with any number of characters	Karl, Mariya

SQL PRACTICE SESSION

--1. List the following details of each employee: employee number, last name, first name, gender, and salary.

SQL PRACTICE SESSION

--2. List employees who were hired in 1986. Limit to 10 (limit)