

Lesson 9: Intro to SQL



What is SQL / Basic SQL commands / SQL Joins / SQL practice

What is SQL?

SQL stands for **Structured Query Language**. It's essentially a language that allows us to "communicate" with databases. But let's break it down a little further:

1. **Structured:** SQL is structured, meaning that it has a defined format and syntax. The rules and structure of SQL allow us to describe exactly what we want from a database.
2. **Query:** A query is a request for data. When you want to retrieve, insert, update, or delete data from a database, you make a query.
3. **Language:** SQL is a language designed for a specific purpose - to interact with relational databases.

What is SQL?

Some commonly used relational database management systems that use SQL include MySQL, Oracle, SQLite, PostgreSQL, and SQL Server.

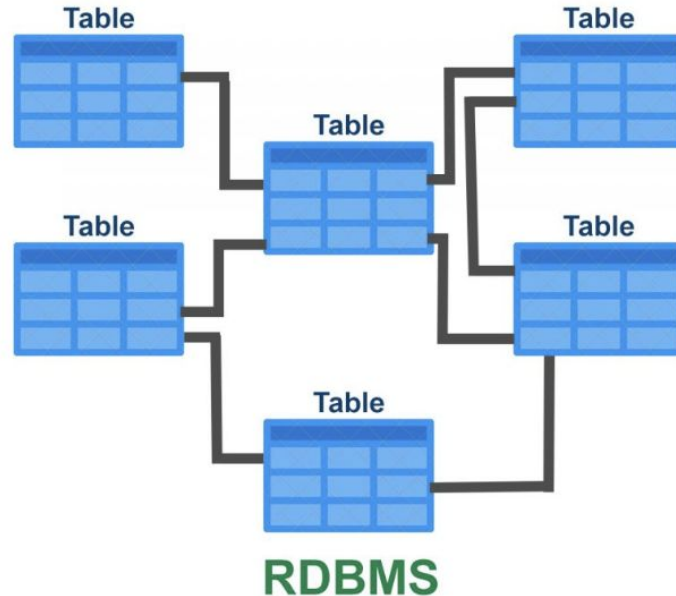
With SQL, we can perform tasks like:

- Retrieving data
- Inserting new data
- Updating existing data
- Deleting data
- Creating new databases and tables
- Maintaining database structures



RDBMS

RDBMS refers to a database management system where data is organized across one or more tables, and these tables can be related to each other. Examples of RDBMS include MySQL, Oracle Database, PostgreSQL, and SQLite.



SQL & RDBMS

Key points to consider about SQL and RDBMS:

1. **Relational Databases:** The "relational" part of RDBMS refers to the way data is stored and organized. In a relational database, data is stored in tables, much like a spreadsheet, where each row represents a single record and each column represents a data field. Each table is linked to others based on common keys or attributes, forming a "relationship."
2. **Querying Data:** SQL allows users to "query" data, meaning you can retrieve specific data that meets certain criteria. For example, a query could be "find all records of customers who live in California."
3. **Data Manipulation:** SQL also provides a means of manipulating the data in the tables. This includes adding new data (INSERT), changing existing data (UPDATE), or removing data (DELETE).
4. **Database Management:** Beyond just working with data, SQL includes commands for creating new databases, creating new tables in a database, and modifying the structure of existing tables (CREATE, ALTER, DROP).
5. **Access Control:** SQL also provides ways to control who has access to what data, and what they can do with it (GRANT, REVOKE).
6. **Standardization:** SQL is a standard language recognized by the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO). Despite this, many database systems include their own additional proprietary extensions that are usually only used on their system.
7. **Wide Adoption:** SQL is widely adopted and used across many industries. Whether data is stored in a small SQLite database running on an IoT device, a MySQL database powering a web application, or a massive Oracle database running a multinational corporation, the SQL language provides the means to work with the data.

SQL Dialects

- Each RDBMS uses a slightly different dialect of SQL, which may include proprietary extensions or syntax variations.
- PSQL, SQLite, and MySQL have their own unique set of functions, data types, and operators.
- Different systems may have optimizations for certain types of queries or operations.
- For instance, PostgreSQL offers advanced indexing and performance optimization features, while MySQL might have different indexing strategies.

SQL commands



“SQL: The language that is neither structured, nor strictly about queries, and certainly isn't limited to just language.” - A wise traveler

SELECT command

- The SELECT command is used to select data from a database.
- The data returned is stored in a result table, also called the result-set.

```
SELECT column1, column2, ...  
FROM table_name;
```

```
SELECT * FROM table_name;
```

Check out: https://www.w3schools.com/sql/sql_select.asp

INSERT INTO command

- The INSERT INTO statement is used to insert new rows of data into a table.
- You can insert data into specific columns or all columns of the table.

```
INSERT INTO table_name (column1, column2, column3, ...)  
VALUES (value1, value2, value3, ...);
```

Check out: https://www.w3schools.com/sql/sql_insert.asp

UPDATE command

- The UPDATE statement is used to modify the existing records in a table.
- It's usually used with the WHERE clause to update specific rows, otherwise all rows would be updated.

UPDATE *table_name*

SET *column1 = value1, column2 = value2, ...*

Check out: https://www.w3schools.com/sql/sql_update.asp

DELETE command

- The DELETE statement is used to delete existing records in a table.
- Like UPDATE, it's commonly used with the WHERE clause to delete specific rows, otherwise all rows would be deleted.

DELETE FROM *table_name* **WHERE** *condition*;

Check out: https://www.w3schools.com/sql/sql_delete.asp

WHERE clause

- The WHERE clause is used to filter records.
- It's used with SELECT, UPDATE, and DELETE commands to perform operations on rows that satisfy the specified condition.

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition;
```

Check out: https://www.w3schools.com/sql/sql_where.asp

CREATE command

The **CREATE** command in **SQL** is used to create a Database, or a Table within a database

- Syntax to create a database:

```
CREATE DATABASE database_name;
```

- Syntax to create a table in a database:

```
CREATE TABLE table_name (  
    column1 datatype,  
    column2 datatype,  
    ...  
);
```

ALTER command

The ALTER command is used to modify the structure of an existing table in a database. It can be used to add, modify, or delete columns. It can also be used to add and drop constraints on existing tables.

- **Adding a column:** You can add a new column to an existing table:

```
ALTER TABLE table_name  
ADD column_name datatype;
```

- **Modifying a column:** You can modify the data type of a column in a table:

```
ALTER TABLE table_name  
MODIFY COLUMN column_name datatype;
```

DROP command

The DROP command is used to delete an entire database, or just a table within a database.

(Be very cautious when using the DROP command because once the table or database is dropped, all the information available in the table or database will be lost permanently.)

- **Syntax for dropping a table:**

```
DROP TABLE table_name;
```

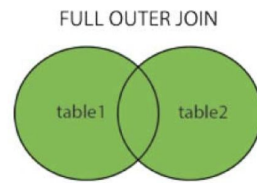
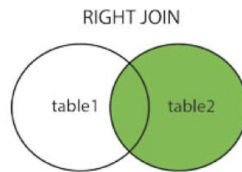
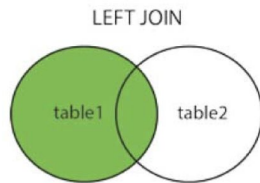
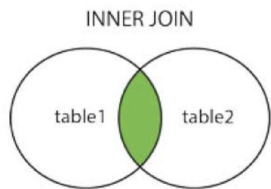
- **Syntax for dropping a database:**

```
DROP DATABASE database_name;
```

SQL Joins

SQL Joins are used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

1. **INNER JOIN:** The INNER JOIN keyword selects records that have matching values in both tables. Also known as a "simple join".
2. **LEFT (OUTER) JOIN:** The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL on the right side if there is no match.
3. **RIGHT (OUTER) JOIN:** The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL on the left side if there is no match.
4. **FULL (OUTER) JOIN:** The FULL OUTER JOIN keyword returns all records when there is a match in either the left (table1) or the right (table2) table records.



SQL Joins example

Now let's imagine we have two tables, one for Customers, and one for Orders.

Customers table:

CustomerID	CustomerName
1	John
2	Jane
3	Alice

Orders table:

OrderID	Product	CustomerID
1	Apples	1
2	Bananas	2
3	Grapes	2
4	Oranges	4

Inner Join

1. INNER JOIN:

This command will match and return rows where there's a CustomerID present in both tables.

sql

 Copy code

```
SELECT Customers.CustomerName, Orders.Product
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
```

Result:


CustomerName	Product
John	Apples
Jane	Bananas
Jane	Grapes

Left Join

2. LEFT JOIN:

This command will return all customers, and any matching orders if available (even if a customer has no orders).

sql

 Copy code

```
SELECT Customers.CustomerName, Orders.Product
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
```

Result:


CustomerName	Product
John	Apples
Jane	Bananas
Jane	Grapes
Alice	NULL

Right Join

3. RIGHT JOIN:

This command will return all orders, and any matching customers if available (even if an order does not have a corresponding customer).

sql

 Copy code

```
SELECT Customers.CustomerName, Orders.Product
FROM Customers
RIGHT JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
```

Result:


CustomerName	Product
John	Apples
Jane	Bananas
Jane	Grapes
NULL	Oranges

Full Join

4. FULL JOIN:

This command will return all records when there's a match in the left table (Customers) or the right table (Orders).

sql

 Copy code

```
SELECT Customers.CustomerName, Orders.Product
FROM Customers
FULL JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
```

Result:

CustomerName	Product
John	Apples
Jane	Bananas
Jane	Grapes
NULL	Oranges
Alice	NULL

Setting up an example SQL database



For our “Users” and “Invoices” tables

Remember our schema from before?

Users

-
id PK int
username String UNIQUE
password String
adminKey String default=null

Invoices

-
id PK int
payment_request Int UNIQUE
value Int
memo String
fees Int
send Bool
settled Bool
settle_date Timestamp
created_at Timestamp default=GETUTCDATE()
userId int FK >- Users.id

Users	
id	int
username	String
password	String
adminKey	Int

Invoices	
id	int
payment_request	Int
value	Int
memo	String
fees	Int
send	Bool
settled	Bool
settle_date	Timestamp
created_at	Timestamp
userId	int



Create Users table

```
CREATE TABLE Users (  
  id INTEGER PRIMARY KEY,  
  username TEXT UNIQUE,  
  password TEXT,  
  adminKey TEXT DEFAULT null  
);
```


Breakdown

```
CREATE TABLE Users (  
    id INTEGER PRIMARY KEY,  
    username TEXT UNIQUE,  
    password TEXT,  
    adminKey TEXT DEFAULT null  
);
```

- The **CREATE TABLE** command is used to create a new table named **Users**.
- This table contains four columns:
 - a. **id**: This is an integer column that serves as the primary key. The **PRIMARY KEY** constraint uniquely identifies each record in a database table. Primary keys must contain unique values.
 - b. **username**: This is a text column that must contain unique values, as indicated by the **UNIQUE** constraint. The **UNIQUE** constraint ensures that all values in a column are different.
 - c. **password**: This is a text column used to store passwords.
 - d. **adminKey**: This is a text column with a default value of null. If no specific value is provided during data insertion, SQLite will automatically insert null.
- The data types for each column are clearly defined, i.e., integer for **id**, text for **username**, **password** and **adminKey**.
- Constraints like **PRIMARY KEY** and **UNIQUE** are used to maintain the integrity of the data.

Create Invoices table

```
CREATE TABLE Invoices (  
  id INTEGER PRIMARY KEY,  
  payment_request INTEGER UNIQUE,  
  value INTEGER,  
  memo TEXT,  
  fees INTEGER,  
  send BOOLEAN,  
  settled BOOLEAN,  
  settle_date DATETIME,  
  created_at DATETIME DEFAULT (datetime('now')),  
  userId INTEGER,  
  FOREIGN KEY (userId) REFERENCES Users(id)  
);
```

Breakdown

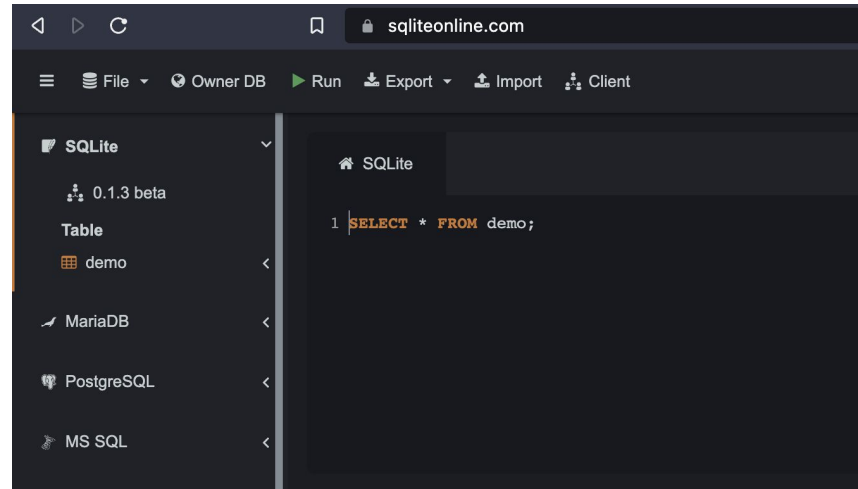
```
CREATE TABLE Invoices (  
  id INTEGER PRIMARY KEY,  
  payment_request INTEGER UNIQUE,  
  value INTEGER,  
  memo TEXT,  
  fees INTEGER,  
  send BOOLEAN,  
  settled BOOLEAN,  
  settle_date DATETIME,  
  created_at DATETIME DEFAULT  
(datetime('now')),  
  userId INTEGER,  
  FOREIGN KEY(userId) REFERENCES  
Users(id)  
);
```

- Creates a new table named **Invoices**.
- Defines the following columns:
 - **id**: an integer column that serves as the primary key.
 - **payment_request**: a unique integer column.
 - **value**: an integer column.
 - **memo**: a text column.
 - **fees**: an integer column.
 - **send**: a boolean column (stored as **0** for **false** and **1** for **true**).
 - **settled**: a boolean column (stored as **0** for **false** and **1** for **true**).
 - **settle_date**: a datetime column.
 - **created_at**: a datetime column that defaults to the current date and time.
 - **userId**: an integer column.
- Defines a foreign key constraint on **userId** that references the **id** column in the **Users** table.]

Visit SQL Online IDE

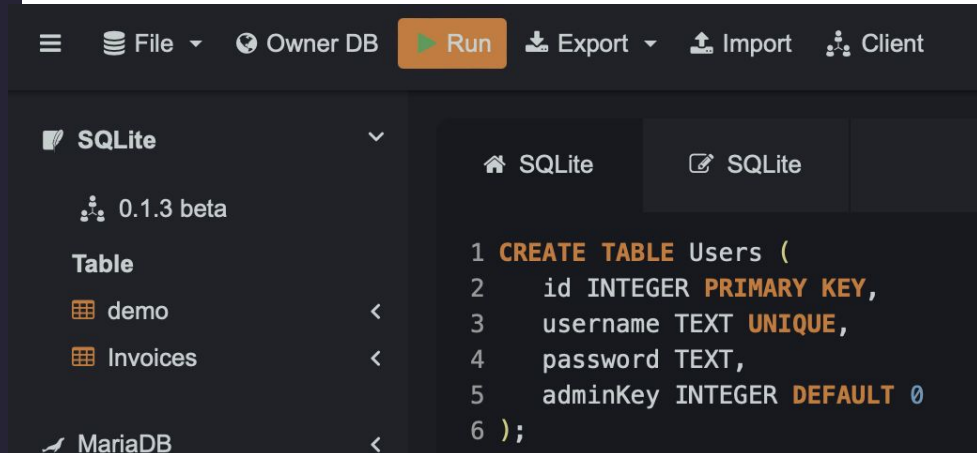
<https://sqliteonline.com/>

We will be using this online sqlite tool to setup a temporary example database in the browser



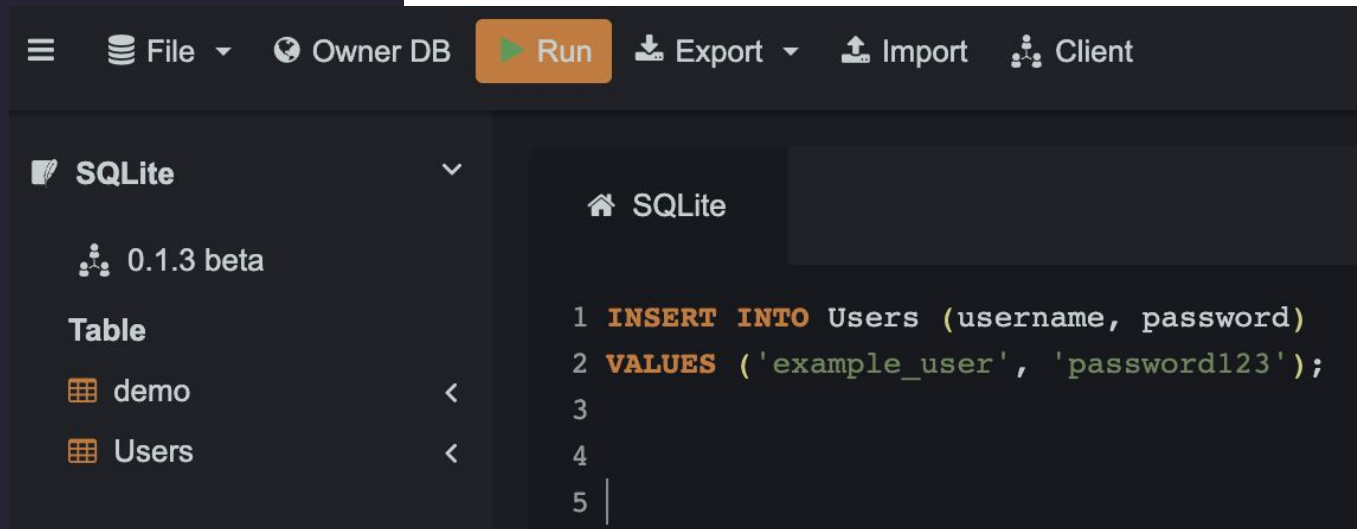
Create your Users table and run it

```
CREATE TABLE Users (  
    id INTEGER PRIMARY KEY,  
    username TEXT UNIQUE,  
    password TEXT,  
    adminKey TEXT DEFAULT null  
);
```



Insert a new user

```
INSERT INTO Users (username, password)
VALUES ('example_user', 'password123');
```



The Users table works!

SQLite 0.1.3 beta

Table

demo

Invoices

Users

MariaDB

PostgreSQL

MS SQL

SQLite

1 **SELECT** * **FROM** Users

id	username	password	adminKey
1	example_user	password123	0

History

Syntax | History

SQLite

SELECT * **FROM** Users

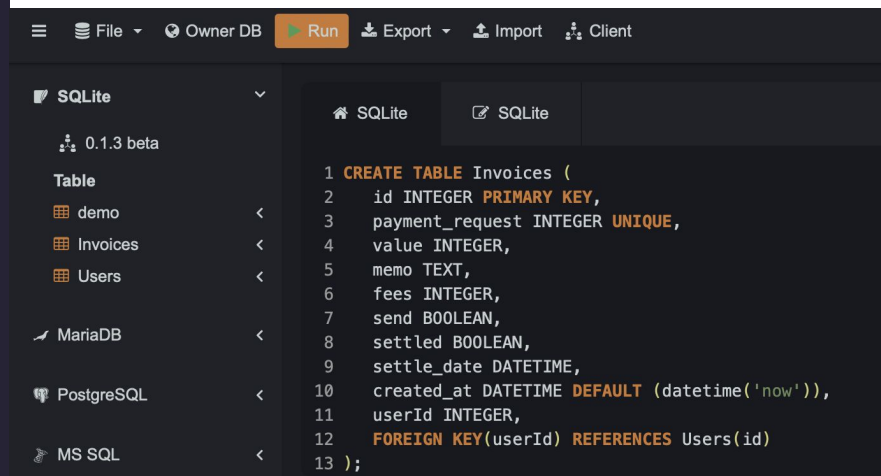
12:35:07

SQLite

INSERT INTO Users (username, password) **VALUES** ('example_user', 'password123')

Create your Invoices table and run it

```
CREATE TABLE Invoices (  
  id INTEGER PRIMARY KEY,  
  payment_request INTEGER UNIQUE,  
  value INTEGER,  
  memo TEXT,  
  fees INTEGER,  
  send BOOLEAN,  
  settled BOOLEAN,  
  settle_date DATETIME,  
  created_at DATETIME DEFAULT  
(datetime('now')),  
  userId INTEGER,  
  FOREIGN KEY(userId) REFERENCES Users(id)  
);
```



Insert a new invoice

```
INSERT INTO Invoices (payment_request, value, memo, fees, send, settled,  
settle_date, userId)  
VALUES (12345, 1000, 'Payment for services', 50, 1, 0, '2023-06-19', 1);
```

Run a query for the invoice

≡

File

Owner DB

Run

Export

Import

Client

SQLite 0.1.3 beta

Table

demo

Invoices

Users

MariaDB

PostgreSQL

MS SQL

SQLite

SQLite

1 **SELECT** * **FROM** Invoices

id	payme...	value	memo	fees	send	settled	settle_...	create...	userId
1	12345	1000	Paymen...	50	1	0	2023-06...	2023-06...	1

History

Syntax | History

SQLite

SELECT * **FROM** Invoices

12:46:56

SQLite

INSERT INTO Invoices (payment_reques
VALUES

Resources

- SQL Cheatsheet (graphic) -
<https://cheatography.com/fetttobse/cheat-sheets/sqlite/>
- Learn SQL In 15 Minutes for Beginners (video) -
<https://www.youtube.com/watch?v=kbKty5ZVKMY>
- SQL Joins Explained Visually (article) -
<https://dataschool.com/how-to-teach-people-sql/sql-join-types-explained-visually/>
- SQL Joins Explained (video) -
<https://www.youtube.com/watch?v=9yeOJ0ZMUYw>
- SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems (article) -
<https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql>