



**TECH  
TALENT**  
SOUTH

# **SQL Part 2: SQL Fundamentals**

# Review: Fundamentals of SQL

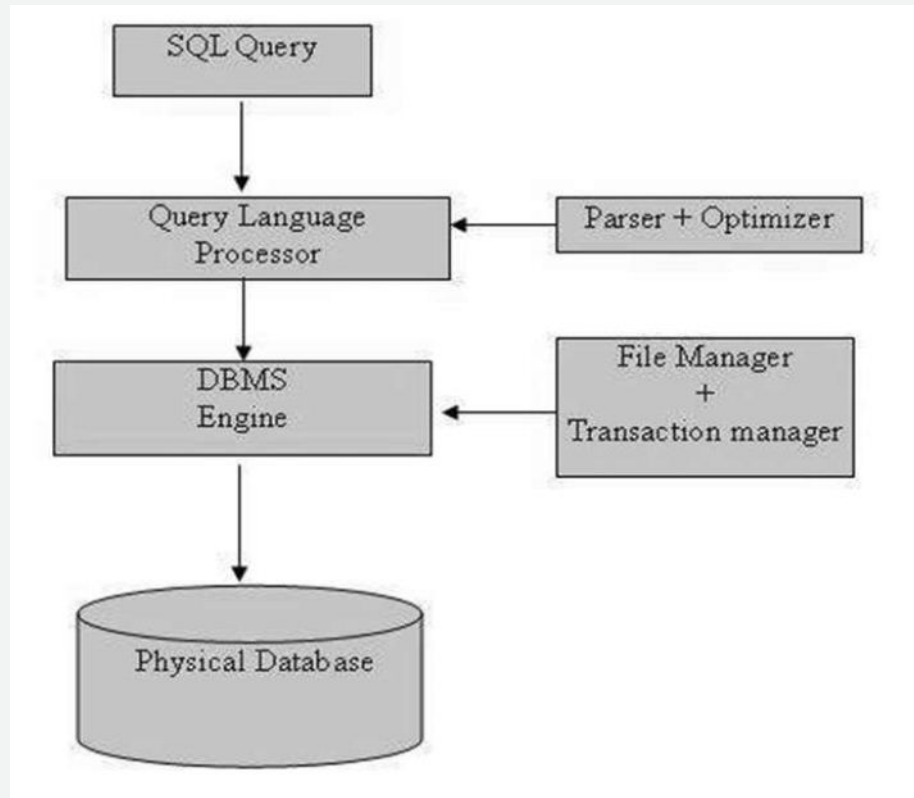
SQL stands for Structured Query Language.

All SQL databases are Relational Databases, but not all databases are SQL and not all databases are relational.



# SQL Architecture

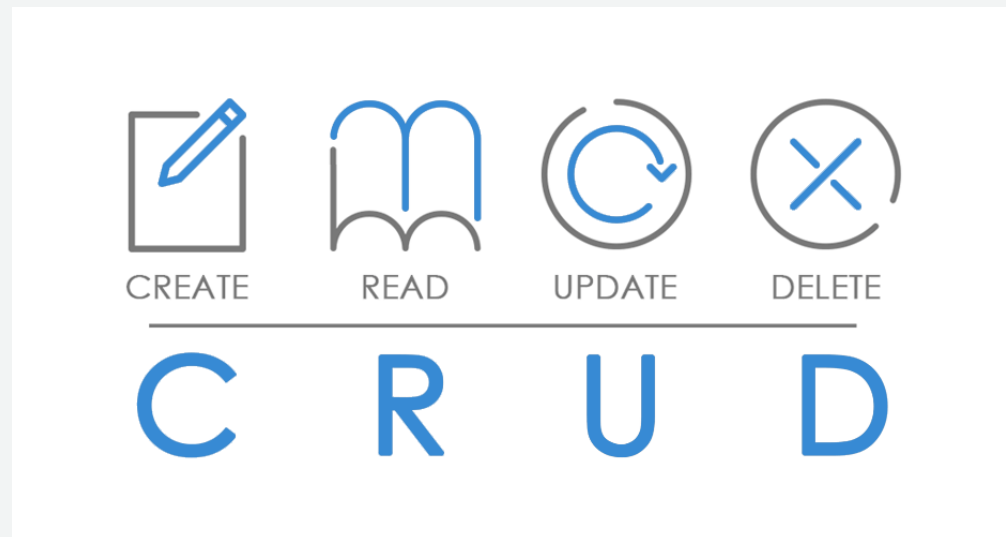
Executing an SQL command goes through the following cycle.



# CRUD

CRUD stands for Create, Read, Update, and Delete.

Think of CRUD actions as those which only affect records (or rows).



We can use SQL to accomplish all the CRUD actions, but we can also create, drop (delete), or alter (update) tables and columns.

# How CRUD-y is SQL?

## CRUD

## HTML Verbs

## SQL Commands

Create

post

insert

Read

get

select

Update

patch/put

update

Delete

delete

delete

# SQL Commands

**Create** – as its name suggests, this command is used to develop a new table, table view or any other database object.

**Alter** – this command is used to modify any database object.

**Drop** – if you want to delete any database component like the table, column value or any other object, this is the command to use.

**Select** – it is used to highlight any row from the table.

**Insert** – it is used to create a row.

**Update** – this command is used to modify or alter the rows (records).

**Delete** – rows can be deleted by this command.

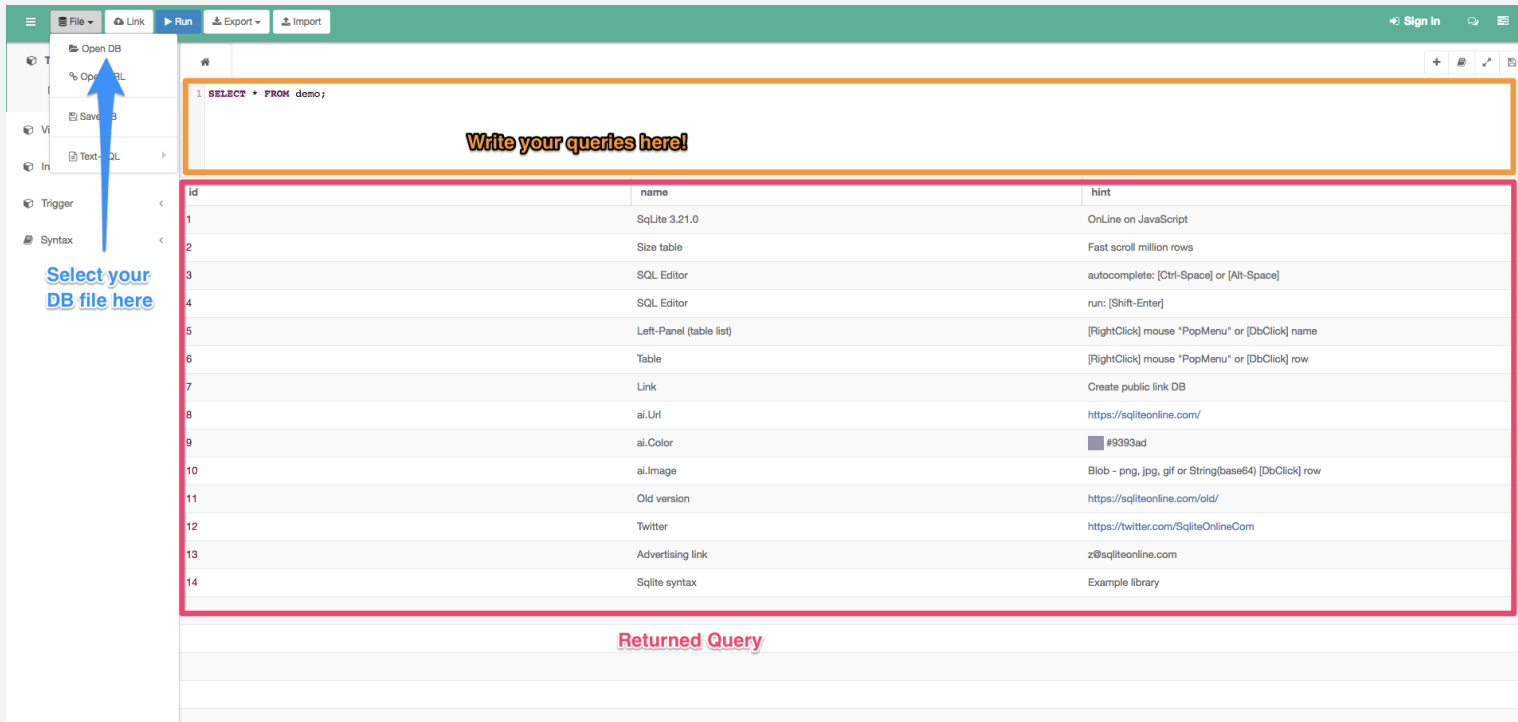
**Grant** – this command provides a user with certain privileges.

**Revoke** – this command is used to take back the user privileges.

# Open your DB

Let's start practicing! Open up [sqliteonline.com](https://sqliteonline.com).

Also, upload the songs database, "my\_ipod.sqlite3" from this [GitHub repository](#).



The screenshot shows the SQLiteOnline.com web application. On the left, the 'File' menu is open, with a blue arrow pointing to 'Open DB'. Below the menu, a text link says 'Select your DB file here'. The main area has a text input field with the placeholder 'Write your queries here!'. Below this is a table with 14 rows and 3 columns: 'id', 'name', and 'hint'. The table is highlighted with a pink border. Below the table, there is a section labeled 'Returned Query'.

id	name	hint
1	SQLite 3.21.0	OnLine on JavaScript
2	Size table	Fast scroll million rows
3	SQL Editor	autocomplete: [Ctrl-Space] or [Alt-Space]
4	SQL Editor	run: [Shift-Enter]
5	Left-Panel (table list)	[RightClick] mouse "PopupMenu" or [DbClick] name
6	Table	[RightClick] mouse "PopupMenu" or [DbClick] row
7	Link	Create public link DB
8	ai.Url	<a href="https://sqliteonline.com/">https://sqliteonline.com/</a>
9	ai.Color	#9393ad
10	ai.Image	Blob - png, jpg, gif or String(base64) [DbClick] row
11	Old version	<a href="https://sqliteonline.com/old/">https://sqliteonline.com/old/</a>
12	Twitter	<a href="https://twitter.com/SqliteOnlineCom">https://twitter.com/SqliteOnlineCom</a>
13	Advertising link	z@sqliteonline.com
14	Sqlite syntax	Example library

# CRUD: CREATE

In SQL, we create  
Tables using **CREATE**

Let's begin by creating  
two tables

```
CREATE TABLE User
(
    UserID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    FirstName VARCHAR NOT NULL,
    LastName VARCHAR NOT NULL,
    Birthplace VARCHAR NOT NULL,
    CreatedAt DATETIME NOT NULL,
    UpdatedAt DATETIME NULL
);

CREATE TABLE Favorite
(
    FavoriteID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    UserID INTEGER NOT NULL,
    ArtistID INTEGER,
    AlbumID INTEGER,
    SongID INTEGER,
    GenreID INTEGER,
    CreatedAt DATETIME NOT NULL,
    UpdatedAt DATETIME NULL,
    FOREIGN KEY (UserID) REFERENCES User (UserID),
    FOREIGN KEY (ArtistID) REFERENCES artists (id),
    FOREIGN KEY (AlbumID) REFERENCES albums (id),
    FOREIGN KEY (SongID) REFERENCES songs (id),
    FOREIGN KEY (GenreID) REFERENCES genres (id)
);
```



# CRUD: CREATE

INSERT adds a row (or record) into an existing table:



## SYNTAX

INSERT INTO TABLE\_NAME (column 1, column 2...)  
VALUES (value 1, value 2...)

```
INSERT INTO User
(
  FirstName,
  LastName,
  Birthplace,
  CreatedAt
)
VALUES
(
  'Wesley',
  'Chambers',
  'Orlando, Florida',
  CURRENT_DATE
);

INSERT INTO User
(
  FirstName,
  LastName,
  Birthplace,
  CreatedAt
)
VALUES
(
  'Michael',
  'Jordan',
  'Brooklyn, New York',
  CURRENT_TIMESTAMP
);
```

# CRUD: READ

SELECT "reads" tables and returns every record which satisfies the statement.

**SYNTAX:** SELECT column 1, column 2 FROM table\_name

SELECT everything from a table:

```
SELECT * FROM artists; --We selected everything: all columns and all rows.
```

SELECT one column from a table:

```
SELECT name FROM songs; --We selected one column and all rows.
```

# CRUD: READ

SELECT some but not all columns from a table:

```
SELECT
    id,
    name,
    created_at
FROM songs; --You get the idea!
```

As you can see, a bare select statement will return all rows for whatever columns are selected.

But, we can limit rows as well, with the help of a WHERE clause:

```
SELECT * FROM songs WHERE length > '5:00';
--We selected only those rows with songs that are longer than five minutes.
(The time is in single tick marks because that column is of data type VARCHAR.)
```

# CRUD: READ

SELECT records whose LastName matches a value within a set of values:

```
SELECT * FROM User WHERE LastName IN ('Jordan', 'Brown', 'Ashley', 'Carnegie');
```

More examples: AND / OR / BETWEEN-AND / LIKE / IS / GLOB:

```
SELECT * FROM User WHERE FirstName = 'Wesley' AND LastName = 'Chambers';
SELECT * FROM User WHERE FirstName = 'Wesley' OR LastName = 'Jordan';
--Note the difference between these first two SELECT statements.

SELECT * FROM User WHERE UserID > 1 OR (UpdatedAt IS NULL AND FirstName = 'Florence');

SELECT * FROM User WHERE FirstName LIKE '%ley';
SELECT * FROM User WHERE FirstName LIKE '%esl%';
SELECT * FROM User WHERE FirstName LIKE '%sle%' OR LastName LIKE 'Jor%';
```

# CRUD: READ

## SQL Logical Operators

The operators shown in the last slide are SQL's 'logical' operators.

Many of these should be familiar from the programming you've already done, but some are slightly different.

<b>ALL</b>	TRUE if all the subquery values meet the condition
<b>AND</b>	TRUE if all the conditions separated by AND are TRUE
<b>ANY</b>	TRUE if any of the subquery values meet the condition
<b>BETWEEN</b>	TRUE if the operand is within the range of the comparisons

# CRUD: READ

## SQL Logical Operators cont...

<b>IN</b>	TRUE if the operand is equal to one of a list of expressions
<b>NOT</b>	Displays a record if the condition(s) is NOT TRUE
<b>OR</b>	TRUE if any of the conditions separated by OR is TRUE
<b>EXISTS</b>	TRUE if the subquery returns one or more records
<b>LIKE</b>	TRUE if the operand matches a pattern

# CRUD: READ

## SQL Logical Operators notes...

BETWEEN/AND is inclusive:



What if you want to return 1-6 exclusive?

Try this instead:

```
SELECT * FROM artists WHERE id > 1 AND id < 6;
```

# CRUD: READ

## SQL Logical Operators notes...

Suppose you want to determine whether a string matches a specific pattern.

LIKE will not get the job done.  
Use GLOB instead:

```
SELECT * FROM songs WHERE name GLOB 'Man*';  
--the '*' means any amount of character or number  
  
SELECT * FROM songs WHERE name GLOB '*Know';  
  
SELECT * FROM songs WHERE name GLOB '*ake*';  
  
SELECT * FROM songs WHERE name GLOB '*8*';  
  
SELECT * FROM songs WHERE name GLOB '??? Learn';  
--the question mark means exactly one character or number  
  
SELECT * FROM songs WHERE name GLOB 'Mary ????';  
  
SELECT * FROM songs WHERE name GLOB '??on??';
```

SELECT \* FROM tableName ...WHERE columnName GLOB [pattern];



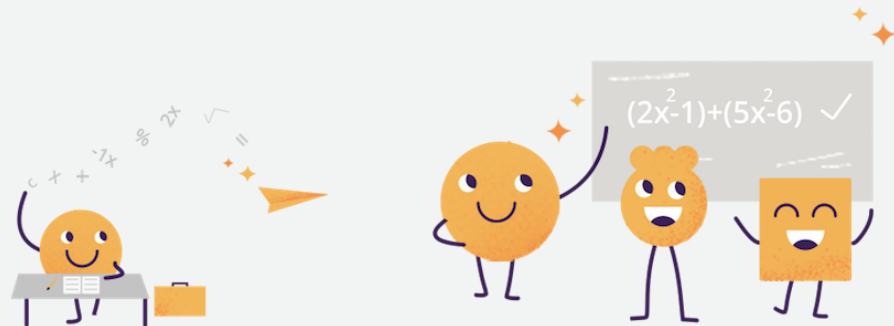
# Practice Problem

INSERT multiple rows into an existing table:

```
INSERT INTO User
(
    FirstName,
    LastName,
    Birthplace,
    CreatedAt
)
VALUES
(
    'Florence',
    'Griffith-Joyner',
    'Little Rock, California',
    CURRENT_TIMESTAMP
),
(
    'Tom',
    'Brady',
    'San Mateo, California',
    CURRENT_DATE
);
```

Compare the two time-stamps, `CURRENT_TIMESTAMP` versus `CURRENT_DATE`.

SELECT the relevant records to see how they differ.



# CRUD: READ

## SQL Comparison Operators

As we've started to notice, SQL also has comparison operators. The following table will highlight some of the comparison operators.

=	evaluates equivalency
!=	Displays a record if the condition(s) is NOT TRUE
<>	not equal
<	is the left less than the right
>	is the left greater than the right

# CRUD: READ

## SQL Comparison Operators cont...

$\geq$	greater than OR equal to
$\leq$	less than OR equal to
$\nless$	not less than
$\ngtr$	not greater than

Comparative operators like  $>$ ,  $<$ ,  $=$ ,  $\neq$ , return true or false, so long as the items compared are not NULL.

# CRUD: READ

## SQL Comparison Operators cont...

This is different for **IS** and **IS NOT** true or false even if one or both operands are NULL.

If both are NULL, **IS** returns 1 (true) and the **IS NOT** returns 0 (false). If one operand is NULL and the other is not, **IS** returns 0 (false) and **IS NOT** returns 1 (true).



**IS** and **IS NOT** never return NULL, whereas = and != and <> sometimes do.

# CRUD: READ

## Summary:

- the columns you want to return are what follows the *SELECT*
- if you want it all, *SELECT \**
- Rows can be filtered in a *WHERE*

Theoretically, you can select just about anything.

# CRUD: UPDATE

With this action, we are not creating, but rather changing, one or more rows.

UPDATE one column for one row:

```
UPDATE artists SET name = 'Janet Jackson' WHERE name LIKE 'janet Jackson';
```

UPDATE multiple columns for one row:

```
UPDATE User
SET FirstName = 'Wesley A.',
    LastName = 'Chambers IV',
    UpdatedAt = CURRENT_TIMESTAMP
WHERE UserID = 1;
```

**SYNTAX:** UPDATE table\_name  
SET column 1 = value 1  
WHERE (condition)

# CRUD: UPDATE

Before we discuss changing multiple rows, which can have a lot of drastic consequences, let's look at a little trick.

Details aside, queries or statements in SQL are called transactions. If you want to see what a transaction will do without actually doing it, you can put it within a TRANSACTION window and roll it back.

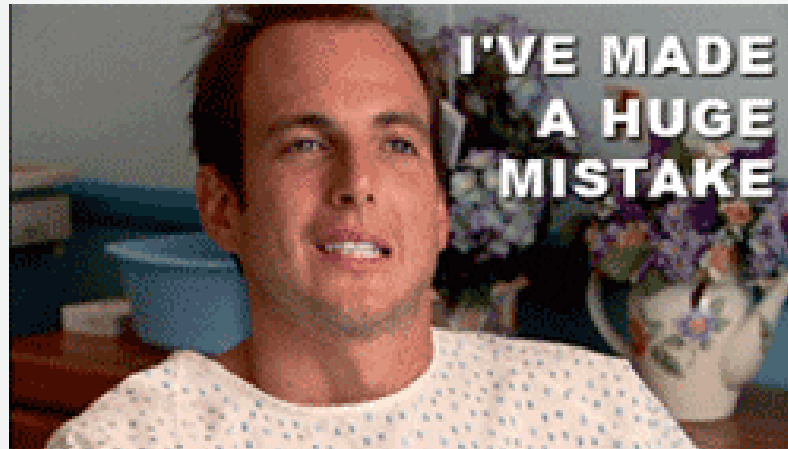
Try this:

```
BEGIN TRANSACTION;  
    UPDATE User SET FirstName = 'Hey, change my name me back!';  
    SELECT * FROM User;  
ROLLBACK;
```

# CRUD: UPDATE

Now, if you execute "SELECT \* FROM User;" outside the transaction, you will see that all is normal. Thankfully, since you rolled back the transaction, nothing changed.

This is a good thing to do when UPDATING or DELETING, since it is easy to make a mistake.





# CRUD: DELETE

Permanently DELETE one or more rows from a table.

As you might expect, I don't want to actually delete anything right now so I'll wrap all of the statements in rollback transaction.

```
BEGIN TRANSACTION;  
  
DELETE FROM User;  
  
SELECT * FROM User;  
--this SELECT is here to help you see what would have happened in the delete statement.  
--You don't need it in order to delete.  
  
ROLLBACK;
```

## SYNTAX:

DELETE FROM table\_name  
WHERE (condition)

# CRUD: DELETE

Since the select was made before the delete action was rolled back, nothing returned. If you select with no transaction window, you will see that nothing was changed.

DELETE specific rows:

```
BEGIN TRANSACTION;  
  DELETE FROM songs WHERE id BETWEEN 5 AND 10;  
  
  SELECT * FROM songs WHERE id < 11;  
  --See that record 5-10 would have been deleted if the rollback had not occurred.  
  
ROLLBACK;
```

# Homework/Challenge

1. SELECT everything from a table
2. SELECT exactly one column from a table. SELECT more than one but not all columns from a table
3. SELECT everything from the songs table whose id numbers are even and greater than 50, or odd and less than ten.
4. INSERT ten new records into the User table. They can be real or fictional individuals.
5. UPDATE the User table and change some names. When you UPDATE, do not forget to put CURRENT\_TIMESTAMP into UpdatedAt. That way, you always know when the last update was.
6. Use a TRANSACTION/ROLLBACK window to DELETE everything from the User table.
7. For each user in the User table, add a record to the Favorite table, but only INSERT for UserID (the FK), and CreatedAt.
8. Use a TRANSACTION/ROLLBACK window to DELETE everything from the Favorite table WHERE the FavoriteID is odd, or even but greater than 10
9. Put "SELECT \* FROM Favorite;" in the window to make sure you delete only those records you should!