# SQL_Cheat_Sheet

February 15, 2018

## 1 SQL Basics Course

- SQL stands for Structured Query Language. SQL has been around since the 1970s!
- Reading data from a database is known as querying.
- Further Reading
  - SQL Entry in Wikipedia
  - Schema

#### SQL Database Websites

- MySQL
- PostgreSQL
- SQLite
- Microsoft SQL Server
- Oracle

#### NoSQL Database Websites

- MongoDB
- CouchBase
- Redis

#### Types of Data

- Text Type Examples
  - TEXT
  - VARCHAR
- Numeric Type Examples
  - INT
  - INTEGER
- Date Type Examples
  - DATETIME
  - DATE
  - TIMESTAMP

Here's documentation sites for some other databases where you can see the similarities and differences in data types. * MySQL Data Types * SQLite Data Types * PostgreSQL Data Types * Microsoft SQL Data Types

**Different Database Tools**

- Mode Analytics used for businesses to get insights from their database.
- pgAdmin for PostgreSQL
- phpMyAdmin and MySQL Workbench for MySQL

- Syntax - The vocabulary and grammatical rules surrounding the structure of your code.
- Keywords - The vocabulary words of a programming language used to issue commands to a computer.

### 1.0.1 Selecting all records from a single database

```
SELECT * FROM <table name>;
```

Example:

```
SELECT * FROM loans;
```

- The asterisk or star symbol (*) means all columns.
- The semi-colon (;) terminates the statement like a period in sentence or question mark in a question.

See all of the SQL used in SQL Basics in the SQL Basics Cheat Sheet.

### 1.0.2 Retrieving a single column:

```
SELECT <column name> FROM <table name>;
```

Examples:

```
SELECT email FROM users;
SELECT first_name FROM users;
SELECT name FROM products;
SELECT zip_code FROM addresses;
```

### 1.0.3 Retrieving multiple columns:

```
SELECT <column name 1>, <column name 2>, ... FROM <table name>;
```

Examples:

```
SELECT first_name, last_name FROM customers;
SELECT name, description, price FROM products;
SELECT title, author, isbn, year_released FROM books;
SELECT name, species, legs FROM pets;
```

### 1.0.4 Aliasing

```
SELECT <column name> AS <alias> FROM <table name>;
SELECT <column name> <alias> FROM <table name>;
```

Examples:

```
SELECT username AS Username, first_name AS "First Name" FROM users;
SELECT title AS Title, year AS "Year Released" FROM movies;
SELECT name AS Name, description AS Description, price AS "Current Price" FROM products;
SELECT name Name, description Description, price "Current Price" FROM products;
```

### 1.0.5 A WHERE Clause

```
SELECT <columns> FROM <table> WHERE <condition>;
```

### 1.0.6 Equality Operator

Find all rows that a given value matches a column's value.

```
SELECT <columns> FROM <table> WHERE <column name> = <value>;
```

Examples:

```
SELECT * FROM contacts WHERE first_name = "Andrew";
SELECT first_name, email FROM users WHERE last_name = "Chalkley";
SELECT name AS "Product Name" FROM products WHERE stock_count = 0;
SELECT title "Book Title" FROM books WHERE year_published = 1999;
```

### 1.0.7 Inequality Operator

- Find all rows that a given value doesn't match a column's value.

```
SELECT <columns> FROM <table> WHERE <column name> != <value>;
SELECT <columns> FROM <table> WHERE <column name> <> <value>;
```

The not equal to or inequality operator can be written in two ways != and <>. The latter is less common.

Examples:

```
SELECT * FROM contacts WHERE first_name != "Kenneth";
SELECT first_name, email FROM users WHERE last_name != "L:one";
SELECT name AS "Product Name" FROM products WHERE stock_count != 0;
SELECT title "Book Title" FROM books WHERE year_published != 2015;
```

## 1.1 Filtering by Comparing Values

### 1.1.1 Relational Operators

There are several relational operators you can use: * < less than * <= less than or equal to * > greater than * >= greater than or equal to

These are primarily used to compare numeric and date/time types.

```
SELECT <columns> FROM <table> WHERE <column name> < <value>;
SELECT <columns> FROM <table> WHERE <column name> <= <value>;
SELECT <columns> FROM <table> WHERE <column name> > <value>;
SELECT <columns> FROM <table> WHERE <column name> >= <value>;
```

Examples:

```
SELECT first_name, last_name FROM users WHERE date_of_birth < '1998-12-01';
SELECT title AS "Book Title", author AS Author FROM books WHERE year_released <= 2015;
SELECT name, description FROM products WHERE price > 9.99;
SELECT title FROM movies WHERE release_year >= 2000;
```

## 1.2 Filtering on More than one condition

You can compare multiple values in a WHERE condition. If you want to test that both conditions are true use the AND keyword, or either conditions are true use the OR keyword.

```
SELECT <columns> FROM <table> WHERE <condition 1> AND <condition 2> ...;
SELECT <columns> FROM <table> WHERE <condition 1> OR <condition 2> ...;
```

Examples:

```
SELECT username FROM users WHERE last_name = "Chalkley" AND first_name = "Andrew";
SELECT * FROM products WHERE category = "Games Consoles" AND price < 400;
SELECT * FROM movies WHERE title = "The Matrix" OR title = "The Matrix Reloaded" OR title = "T
SELECT country FROM countries WHERE population < 1000000 OR population > 100000000;
```

## 1.3 Searching within a Set of Values

```
SELECT <columns> FROM <table> WHERE <column> IN (<value 1>, <value 2>, ...);
```

Examples:

```
SELECT name FROM islands WHERE id IN (4, 8, 15, 16, 23, 42);
SELECT * FROM products WHERE category IN ("eBooks", "Books", "Comics");
SELECT title FROM courses WHERE topic IN ("JavaScript", "Databases", "CSS");
SELECT * FROM campaigns WHERE medium IN ("email", "blog", "ppc");
```

To find all rows that are not in the set of values you can use NOT IN.

```
SELECT <columns> FROM <table> WHERE <column>  NOT IN (<value 1>, <value 2>, ...);
```

Examples:

```
SELECT answer FROM answers WHERE id IN (7, 42);
SELECT * FROM products WHERE category NOT IN ("Electronics");
SELECT title FROM courses WHERE topic NOT IN ("SQL", "NoSQL");
```

### 1.4 Finding data that matches a pattern

Placing the percent symbol (%) any where in a string in conjunction with the LIKE keyword will operate as a wildcard. Meaning it can be substituted by any number of characters, including zero!

```
SELECT <columns> FROM <table> WHERE <column> LIKE <pattern>;
```

Examples:

```
SELECT title FROM books WHERE title LIKE "Harry Potter%Fire";
SELECT title FROM movies WHERE title LIKE "Alien%";
SELECT * FROM contacts WHERE first_name LIKE "%drew";
SELECT * FROM books WHERE title LIKE "%Brief History%";
```

PostgreSQL Specific Keywords * LIKE in PostgreSQL is case-sensitive. To case-insensitive searches do ILIKE.

```
SELECT * FROM contacts WHERE first_name ILIKE "%drew";
```

## 2 Adding a Row to a table

Inserting a single row:

```
INSERT INTO <table> VALUES (<value 1>, <value 2>, ...);
```

This will insert values in the order of the columns prescribed in the schema.
Examples:

```
INSERT INTO users VALUES  (1, "chalkers", "Andrew", "Chalkley");
INSERT INTO users VALUES  (2, "ScRiPtKiDdIe", "Kenneth", "Love");

INSERT INTO movies VALUES (3, "Starman", "Science Fiction", 1984);
INSERT INTO movies VALUES (4, "Moulin Rouge!", "Musical", 2001);
```

Inserting a single row with values in any order:

```
INSERT INTO <table> (<column 1>, <column 2>) VALUES (<value 1>, <value 2>);
INSERT INTO <table> (<column 2>, <column 1>) VALUES (<value 2>, <value 1>);
```

Examples:

```
INSERT INTO users (username, first_name, last_name) VALUES ("chalkers", "Andrew", "Chalkley");
INSERT INTO users (first_name, last_name, username) VALUES  ("Kenneth", "Love", "ScRiPtKiDdIe")

INSERT INTO movies (title, genre, year_released) VALUES ("Starman", "Science Fiction", 1984);
INSERT INTO movies (title, year_released, genre) VALUES ("Moulin Rouge!", 2001,  "Musical");
```

See all of the SQL used in Modifying Data With SQL in the Modifying Data With SQL Cheat-sheet.

## 3 Inserting Multiple Rows

Inserting multiple rows in a single statement:

```
INSERT INTO <table> (<column 1>, <column 2>, ...)
             VALUES
                    (<value 1>, <value 2>, ...),
                    (<value 1>, <value 2>, ...),
                    (<value 1>, <value 2>, ...);
```

Examples:

```
INSERT INTO users (username, first_name, last_name)
    VALUES
                  ("chalkers", "Andrew", "Chalkley"),
                  ("ScRiPtKiDdIe", "Kenneth", "Love");

INSERT INTO movies (title, genre, year_released)
     VALUES
                   ("Starman", "Science Fiction", 1984),
                   ("Moulin Rouge!", "Musical", 2001);
```

## 4 Updating Rows

An update statement for all rows:

```
UPDATE <table> SET <column> = <value>;
```

The = sign is different from an equality operator from a WHERE condition. It's an assignment operator because you're assigning a new value to something.

Examples:

```
UPDATE users SET password = "thisisabadidea";
UPDATE products SET price = 2.99;
```

Update multiple columns in all rows:

```
UDPATE <table> SET <column 1> = <value 1>, <column 2> = <value 2>;
```

Examples:

```
UPDATE users SET first_name = "Anony", last_name = "Moose";
UPDATE products SET stock_count = 0, price = 0;
```

## 5 Updating Specific Rows

An update statement for specific rows:

```
UPDATE <table> SET <column> = <value> WHERE <condition>;
```

Examples:

```
UPDATE users SET password = "thisisabadidea" WHERE id = 3;
UPDATE blog_posts SET view_count = 1923 WHERE title = "SQL is Awesome";
```

Update multiple columns for specific rows:

```
UDPATE <table> SET <column 1> = <value 1>, <column 2> = <value 2> WHERE <condition>;
```

Examples:

```
UPDATE users SET entry_url = "/home", last_login = "2016-01-05" WHERE id = 329;
UPDATE products SET status = "SOLD OUT", availability = "In 1 Week" WHERE stock_count = 0;
```

## 6 Deleting Rows

To delete all rows from a table:

```
DELETE FROM <table>;
```

Examples:

```
DELETE FROM logs;
DELETE FROM users;
DELETE FROM products;
```

## 7 Deleting From Specific Rows

To delete specific rows from a table:

```
DELETE FROM <table> WHERE <condition>;
```

Examples:

```
DELETE FROM users WHERE email = "andrew@teamtreehouse.com";
DELETE FROM movies WHERE genre = "Musical";
DELETE FROM products WHERE stock_count = 0;
```

## 8 Perfroming Transactions

Definitions * Autocommit - every statement you write gets saved to disk. * Seeding - populating a database for the first time. * Script file - a file containing SQL statements.

Switch autocommit off and begin a transaction:

```
BEGIN TRANSACTION;
```

Or simply:

```
BEGIN;
```

To save all results of the statements after the start of the transaction to disk:

```
COMMIT;
```

## 9 Retrieving Results in a Particular Order

Ordering by a single column criteria:

```
SELECT * FROM <table name> ORDER BY <column> [ASC|DESC];
```

ASC is used to order results in ascending order.
DESC is used to order results in descending order.
Examples:

```
SELECT * FROM books ORDER BY title ASC;
SELECT * FROM products WHERE name = "Sonic T-Shirt" ORDER BY stock_count DESC;
SELECT * FROM users ORDER BY signed_up_on DESC;
SELECT * FROM countries ORDER BY population DESC;
```

Ordering by multiple column criteria:

```
SELECT * FROM <table name> ORDER BY <column> [ASC|DESC],
                                    <column 2> [ASC|DESC],
                                    ...,
                                    <column n> [ASC|DESC];
```

Ordering is prioritized left to right.
Examples:

```
SELECT * FROM books ORDER BY    genre ASC,
                                title ASC;

SELECT * FROM books ORDER BY    genre ASC,
                                year_published DESC;

SELECT * FROM users ORDER BY    last_name ASC,
                                first_name ASC;
```

## 10 Limiting the Number or results

SQLite, PostgreSQL and MySQL

To limit the number of results returned, use the LIMIT keyword.

```
SELECT <columns> FROM <table> LIMIT <# of rows>;
```

MS SQL

To limit the number of results returned, use the TOP keyword.

```
SELECT TOP <# of rows> <columns> FROM <table>;
```

Oracle

To limit the number of results returned, use the ROWNUM keyword in a WHERE clause.

```
SELECT <columns> FROM <table> WHERE ROWNUM <= <# of rows>;
```

## 11 Paging Through Results

SQLite, PostgreSQL and MySQL

To page through results you can either use the OFFSET keyword in conjunction with the LIMIT keyword or just with LIMIT alone.

```
SELECT <columns> FROM <table> LIMIT <# of rows> OFFSET <skipped rows>;
SELECT <columns> FROM <table> LIMIT <skipped rows>, <# of rows>;
```

MS SQL and Oracle

To page through results you can either use the OFFSET keyword in conjunction with the FETCH keyword. Cannot be used with TOP.

```
SELECT <columns> FROM <table> OFFSET <skipped rows> ROWS FETCH NEXT <# of rows> ROWS ONLY;
```

## 12 SQL Functions

Syntax definitions

Keywords: Commands issued to a database. The data presented in queries is unaltered.

Operators: Performs comparisons and simple manipulation

Functions: Presents data differently through more complex manipulation

A function looks like:

```
<function name>(<value or column>)
```

Example:

```
UPPER("Andrew Chalkley")
```

## 13 Adding Text Columns Together

SQLite, PostgreSQL and Oracle
Use the concatenation operator ||.

```
SELECT <value or column> || <value or column> || <value or column>  FROM <table>;
```

MS SQL
Use the concatenation operator +.

```
SELECT <value or column> + <value or column> + <value or column>  FROM <table>;
```

MySQL, Postgres and MS SQL
Use the CONCAT() function.

```
SELECT CONCAT(<value or column>, <value or column>, <value or column>) FROM <table>;
```

## 14 Finding The Length of Text

To obtain the length of a value or column use the LENGTH() function.

```
SELECT LENGTH(<value or column>) FROM <tables>;
```

## 15 Changing the Case of Text Columns

Use the UPPER() function to uppercase text.

```
SELECT UPPER(<value or column>) FROM <table>;
```

Use the LOWER() function to lowercase text.

```
SELECT LOWER(<value or column>) FROM <table>;
```

## 16 Creating Excerpts From Text

To create smaller strings from larger piece of text you can use the SUBSTR() funciton or the substring function.'

```
SELECT SUBSTR(<value or column>, <start>, <length>) FROM <table>;
```

## 17 Replacing Portions of Text

To replace piece of strings of text in a larger body of text you can use the REPLACE() function.

```
SELECT REPLACE(<original value or column>, <target string>, <replacement string>) FROM <table>
```