SQL Fundamentals - Cheatsheet

This cheatsheet contains a collection of SQL commands and syntax that I have found useful while working with SQL. I created this cheatsheet while learning SQL and thought I'd combine them all in a single place so it's easier for me as well as someone else to refer them whenever needed.

Table of Contents

Introduction to SQL

Structured Query Language (SQL) is a standard language to communicate with a database and perform tasks such as querying, updating, inserting, and deleting data. It is used to interact with relational databases such as MySQL, PostgreSQL, SQL Server, Oracle, etc.

SQL vs MySQL

- SQL is a language and MySQL is a database management system
- SQL is used to communicate with a database and MySQL is used to store and manage the data
- Other database management systems that use SQL are PostgreSQL, SQL Server, Oracle, etc.
- All SQL DBMSs have their own implementation of features

Database vs DBMS

- A database is a collection of data like a phonebook, music library, etc.
- A database management system (DBMS) is software that is used to manage the database like MySQL, PostgreSQL, SQL Server, Oracle, etc.

SQL vs NoSQL

- SQL is a relational database management system (RDBMS) and NoSQL is a non-relational database management system
- SQL databases are table-based and NoSQL databases are document, key-value, wide-column, or graph-based
- SQL databases are better for complex queries and NoSQL databases are better for hierarchical data storage
- SQL databases are vertically scalable and NoSQL databases are horizontally scalable
- SQL databases are better for multi-row transactions and NoSQL databases are better for unstructured data like documents or JSON

Installation and Setup

MySQL can be installed from the official website or using a package manager like Homebrew.

MySQL Server

- MySQL server is used to store and manage the data
- It listens on port 3306 by default
- It can be started, stopped, and restarted using the command line

MySQL Workbench

- Allows writing and running SQL queries in files
- Supports saving and opening SQL files

Download and Install

1. Visit MySQL's official website
2. Go to MySQL Installer Method and download the installer
3. Open the installer and check the tools you want to install
    1. Select the custom option
    2. Remove the Visual Studio option (2GB)
4. Click on Next and Next to download the packages

5. Click on execute to install the packages
6. Configuration can be left as default
7. Create a strong root user password
    1. Password would be used to login to the MySQL server
    2. REMEMBER THE PASSWORD
8. Keep clicking next and finish to complete the installation

Getting CLI Up and Running

1. Search for MySQL Command Line Client
2. Enter the root user password
3. The client is now connected to the MySQL server
4. Run SQL queries directly from the command line
5. Run SQL files using the command - `source file_name.sql`
6. Type `quit` to exit the client

Getting Workbench Up and Running

1. Search for MySQL Workbench
2. Open the needed connection
3. Enter the root user password
4. Workbench is now connected to the MySQL server
5. Run SQL queries
6. Click on the lightning bolt to execute the query
7. Click on the floppy disk to save the query

Setting up SQL in VS Code

1. Open VS Code.
2. Go to extensions in sidebar.
3. Search for **Database Client** and **Database Client JDBC** and install both.
4. A Database icon will appear in sidebar, all the databases and their tables will appear here.
5. Click on the database icon and setup a new connection.
6. Create a new file with .sql extension(ex- test.sql).
7. Now you can write your sql commands here and execute them.

> TIP - To make your sql code more readable install **SQL Formatter** extension and set it as default formatter.

Databases

A database is a collection of tables. It is a container for tables and other objects. We can have multiple databases in a single SQL Server instance.

Queries

- Get the list of databases on the server

```
SHOW DATABASES;
```

- Create a new database

```
CREATE DATABASE my_database;
```

- Use a database

```
USE my_database;
```

- Delete a database

```
DROP DATABASE my_database;
```

- Show current database

```
SELECT DATABASE();
```

Naming Conventions

- SQL terms should be in uppercase
- DB name should be obvious and descriptive
- Use underscores to separate words instead of spaces

Tables

A table is a collection of related data held in a structured format within a database.

Data Types

Each column in a table has a specific data type. The data type defines the kind of data that can be stored in the column. There are a lot of data types in SQL but the most common ones are:

- **Numeric**: INT, DECIMAL, FLOAT
- **String**: CHAR, VARCHAR
- **Date and Time**: DATE, TIME, DATETIME

Table Queries

- Create a new table

```
CREATE TABLE my_table (
  id INT,
  name VARCHAR(100),
  age INT
);
```

- Show all tables in the current database

```
SHOW TABLES;
```

- Show the structure of a table

```
SHOW COLUMNS FROM my_table;
-- or
DESCRIBE my_table;
-- or
DESC my_table;
```

- Delete a table

```
DROP TABLE my_table;
```

MySQL Comments

- Single line comment

```
-- This is a single-line comment
```

- Multi-line comment

```
/* This is a
multi-line comment */
```

Constraints

Constraints are used to specify rules for the data in a table. They are used to limit the type of data that can go into a table.

NOT NULL

- Enforces a column to not accept NULL values.

```
CREATE TABLE my_table (
  id INT NOT NULL,
  name VARCHAR(100) NOT NULL,
  age INT
);
```

DEFAULT

- Used to set a default value for a column.

```
CREATE TABLE my_table (
  id INT,
  name VARCHAR(100),
  age INT DEFAULT 18
);
```

UNIQUE

- Ensures that all values in a column are different.

```
CREATE TABLE my_table (
  id INT UNIQUE,
  name VARCHAR(100),
  email VARCHAR(100) UNIQUE,
  age INT
);
```

CHECK

- Ensures that all values in a column satisfy a specific condition.

```
CREATE TABLE my_table (
  id INT,
  name VARCHAR(100),
  age INT,
  CHECK (age >= 18)
);
```

PRIMARY KEY

- Uniquely identifies each record in a table
- IDs are used as the most common primary keys

```
CREATE TABLE my_table (
  id INT PRIMARY KEY,
  name VARCHAR(100),
  age INT
);
```

AUTO_INCREMENT

- Automatically generates a unique number for each row

```
CREATE TABLE my_table (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100),
  age INT
);
```

Named Constraints

- Constraints can also be named

```
CREATE TABLE my_table (
  id INT,
  name VARCHAR(100),
  age INT,
  CONSTRAINT pk_id PRIMARY KEY (id)
);
```

Multiple Column Constraints

- Constraints can be created from multiple columns

```
CREATE TABLE my_table (
  id INT,
  name VARCHAR(100),
  age INT,
  CONSTRAINT pk_id_name PRIMARY KEY (id, name)
);
```

Inserting Data

Insert statements are used to insert data into a table. They take the table name, columns and values to be inserted.

Insert Queries

- Inset data into a table

```
INSERT INTO my_table (id, name, age) VALUES (1, 'John', 25);
```

- Retrieve all rows from a table

```
SELECT * FROM my_table;
```

- Insert multiple rows into a table

```
INSERT INTO my_table (id, name, age)
VALUES
  (2, 'Jane', 30),
  (3, 'Doe', 35);
```

CRUD Operations

CRUD stands for Create, Read, Update, and Delete. These are the four basic operations that can be performed on a database.

Create

- INSERT statement is used to add new rows to a table

```
INSERT INTO my_table (id, name, age) VALUES (1, 'John', 25);
```

Read

- SELECT statement is used to retrieve data from a table

```
SELECT * FROM my_table;

SELECT name, age FROM my_table;
```

WHERE Clause

- Used to extract only those records that fulfill a specified condition

```
SELECT * FROM my_table WHERE age > 25;
```

Aliases

- Used to give a table, or a column in a table, a temporary name

```
SELECT name AS full_name, age AS years FROM my_table;
```

Update

- Used to modify the existing records in a table

```
UPDATE my_table SET age = 30 WHERE name = 'John';
```

- Can also update multiple columns at once.

```
UPDATE my_table SET age = 30, name = 'John Doe' WHERE id = 1;
```

- If there is no WHERE clause, all records will be updated.

Delete

- Used to delete records from a table.

```
DELETE FROM my_table WHERE name = 'John';
```

- If there is no WHERE clause, all records will be deleted.

> **Rule of thumb:** If update or delete statements are by mistake, they can cause a lot of damage. Therefore, it's always a good practice to first run a SELECT statement with the same WHERE clause to see which records will be affected.

Drop vs Delete

- DROP is used to delete a table from the database
- DELETE is used to delete records from a table

String Functions

String functions are used to perform operations on strings like searching for a substring, replacing a substring, etc.

CONCAT

- Used to combine strings

```
SELECT CONCAT('H', 'e', 'y'); -- Hey
SELECT CONCAT(f_name. ' ', l_name) FROM users; -- John Doe
```

CONCAT_WS

- Stands for `CONCAT` *With Separator*
- Takes separator as first argument

```
SELECT CONCAT_WS('/', '04', '08', '2001'); -- 04/08/2001
```

SUBSTRING

- Used to extract a part of a string
- Takes input string, start position, and length as arguments

```
SELECT SUBSTRING('Hello World', 1, 5); -- Hello

-- changing starting point
SELECT SUBSTRING('Hello World', 7, 5); -- World

-- specify starting point only
SELECT SUBSTR('Hello World', 7); -- World

-- negative starting point
SELECT SUBSTR('Hello World', -3); -- rld
```

REPLACE

- Used to replace all occurrences of a substring within a string
- Takes input string, old substring, and new substring as arguments

```
SELECT REPLACE('www.example.com', 'w.', 'W.'); -- wwW.example.com
```

REVERSE

- Used to reverse a string

```
SELECT REVERSE('Hello World'); -- dlroW olleH
```

CHAR_LENGTH

- Used to get the length of a string

```
SELECT CHAR_LENGTH('Hello World'); -- 11
```

UPPER and LOWER

- Used to convert a string to upper or lower case

```
SELECT UPPER('Hello World'); -- HELLO WORLD
SELECT LOWER('Hello World'); -- hello world
```

INSERT

- Used to insert a substring into a string at a specified position
- Takes input string, start position, how many characters to remove, and new substring as arguments

```
SELECT INSERT('Hello Bobby', 7, 4, 'There'); -- Hello Therey
```

LEFT and RIGHT

- Used to extract a specified number of characters from a string, starting from the left or right

```
SELECT LEFT('Hello World', 5); -- Hello
SELECT RIGHT('Hello World', 5); -- World
```

REPEAT

- Used to repeat a string a specified number of times

```
SELECT REPEAT('Hello', 3); -- HelloHelloHello
```

TRIM

- Used to remove leading and trailing spaces from a string

```
SELECT TRIM(' Hello '); -- Hello
```

- Can also remove leading and trailing characters

```
SELECT TRIM(LEADING 'x' FROM 'xxxHello'); -- Hello
SELECT TRIM(TRAILING 'x' FROM 'Helloxxx'); -- Hello
SELECT TRIM(BOTH 'x' FROM 'xxxHelloxxx'); -- Hello
```

Refining Selections

When we want to retrieve data from a table, we can use the SELECT statement. We can also refine the selection by using a few clauses.

DISTINCT

- Used to return only distinct values

```
SELECT DISTINCT age FROM my_table;
```

ORDER BY

- Used to sort the result set in ascending or descending order

```
SELECT * FROM my_table ORDER BY age;
SELECT * FROM my_table ORDER BY age DESC;
```

- Can also sort by multiple columns

```
SELECT * FROM my_table ORDER BY age, name;
```

LIMIT

- Used to limit the number of records returned

```
SELECT * FROM my_table LIMIT 5;
```

- Specify the starting point and number of records to return

```
SELECT * FROM my_table LIMIT 5, 10;
```

LIKE

- Used to search by finding records that include the specified value

```
SELECT * FROM my_table WHERE name LIKE 'J%';
SELECT * FROM my_table WHERE name LIKE 'J_n';

-- any no. of characters before or after D
SELECT * FROM my_table WHERE name LIKE '%D%';
```

Wildcards

- `%` -> Represents zero or more characters
- `_` -> Represents a single character

Aggregate Functions

Aggregate functions are used to perform calculations on a set of values and return a single value.

COUNT

- Used to count the number of rows in a table

```
SELECT COUNT(*) FROM my_table;
```

- Can also count the number of non-NULL values in a column

```
SELECT COUNT(age) FROM my_table;
```

GROUP BY

- Used to group rows that have the same values

```
-- group by age and count the number of people in each age group
SELECT age, COUNT(*) FROM my_table GROUP BY age;
```

- One or more columns can be used to group the data

```
SELECT age, name, COUNT(*) FROM my_table GROUP BY age, name;
```

MIN and MAX

- Used to get the minimum and maximum value in a column

```
SELECT MIN(age) FROM my_table;
SELECT MAX(age) FROM my_table;
```

- Can also be used with GROUP BY

```
SELECT age, MIN(name) FROM my_table GROUP BY age;
SELECT age, MAX(name) FROM my_table GROUP BY age;
```

Subqueries

- Used to nest a query within another query

```
-- first get the maximum age and then get the name of the person with that age
SELECT name FROM my_table WHERE age = (SELECT MAX(age) FROM my_table);
```

SUM

- Used to get the sum of a column

```
SELECT SUM(quantity) FROM orders;
```

- Can also be used with GROUP BY

```
SELECT category, SUM(quantity) FROM orders GROUP BY category;
```

AVG

- Used to get the average value of a column

```
SELECT AVG(age) FROM my_table;
```

- Can also be used with GROUP BY

```
SELECT category, AVG(quantity) FROM orders GROUP BY category;
```

Data Types

There are many data types in SQL but only a few are used frequently.

CHAR and VARCHAR

- CHAR is fixed length and VARCHAR is variable length
- CHAR is faster for fixed-length data and VARCHAR is faster for variable-length data

```
CREATE TABLE my_table (
  marks CHAR(2),
  email VARCHAR(100)
);
```

Integer Types

- INT, TINYINT, SMALLINT, MEDIUMINT, BIGINT
- Different in memory and ranges

```
CREATE TABLE my_table (
  id INT,
  age TINYINT
);
```

- Range of INT: -2147483648 to 2147483647
- Range of TINYINT: -128 to 127

Signed and Unsigned

- Signed can store both positive and negative numbers
- Unsigned can store only positive numbers

```
CREATE TABLE my_table (
  id INT UNSIGNED,
  age TINYINT UNSIGNED
);
```

- Range of INT UNSIGNED: 0 to 4294967295
- Range of TINYINT UNSIGNED: 0 to 255

Floating-Point Types

- DECIMAL, FLOAT and DOUBLE
- DECIMAL is used for exact values and FLOAT and DOUBLE are used for approximate values

```
CREATE TABLE my_table (
  price DECIMAL(10, 2),
  weight FLOAT
);
```

- DECIMAL(10, 2) can store 8 digits before the decimal point and 2 digits after the decimal point

FLOAT vs DOUBLE

- FLOAT is a single-precision floating-point number and DOUBLE is a double-precision floating-point number
- DOUBLE is faster and more accurate but takes more space
- FLOAT takes 4 bytes and DOUBLE takes 8 bytes

Date and Time

Date and time data types are used to store date and time values.

DATE

- Used to store date values in the format 'YYYY-MM-DD'

```
CREATE TABLE my_table (
  dob DATE
);
```

- Example: '2021-08-04'

TIME

- Used to store time values in the format 'HH:MM:SS'

```
CREATE TABLE my_table (
  time TIME
);
```

- Example: '12:30:45'

DATETIME

- Used to store date and time values in the format 'YYYY-MM-DD HH:MM:SS'

```
CREATE TABLE my_table (
  created_at DATETIME
);
```

- Example: '2021-08-04 12:30:45'

TIMESTAMP

- Used to store the current date and time in the format 'YYYY-MM-DD HH:MM:SS'
- Consumes less space than DATETIME
- Supports smaller range of values (1970-2038)

```
CREATE TABLE my_table (
  updated_at TIMESTAMP
);
```

- Suitable for fields like `created_at` and `updated_at`
- Automatically updates when a record is created or updated

```
CREATE TABLE my_table (
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT NOW ON UPDATE NOW()
);
```

CURDATE and CURTIME

- Used to get the current date and time

```
SELECT CURDATE();
SELECT CURTIME();
```

- Example: '2021-08-04' and '12:30:45'

NOW

- Used to get the current date and time

```
SELECT NOW();
```

- Example: '2021-08-04 12:30:45'

Date Functions

Date functions are used to perform operations on date and time values.

- Get the day of the month

```
SELECT DAY('2021-08-04'); -- 4
```

- Get the name of the day

```
SELECT DAYNAME('2021-08-04'); -- Wednesday
```

- Get the day of the year

```
SELECT DAYOFYEAR('2021-08-04'); -- 216
```

- Get the name of the month

```
SELECT MONTHNAME('2021-08-04'); -- August
```

Time Functions

Time functions are used to perform operations on time values.

- Get the hour

```
SELECT HOUR('12:30:45'); -- 12
```

- Get the minute

```
SELECT MINUTE('12:30:45'); -- 30
```

Formatting Dates

We can format date and time values using the `DATE_FORMAT` function and a bunch of format specifiers.

```
SELECT DATE_FORMAT('2021-08-04', '%d-%m-%Y'); -- 04-08-2021
```

```
SELECT DATE_FORMAT('12:30:45', '%H:%i:%s'); -- 12:30:45
```

Maths and Dates

We can perform mathematical operations on date and time values.

- Add a date or time interval to a date

```
SELECT DATE_ADD('2021-08-04', INTERVAL 1 DAY); -- 2021-08-05
```

- Subtract a date or time interval from a date

```
SELECT DATE_SUB('2021-08-04', INTERVAL 1 DAY); -- 2021-08-03
```

- Get the difference between two dates

```
SELECT DATEDIFF('2021-08-04', '2021-08-01'); -- 3
```

- Get the difference between two timestamps

```
SELECT TIMESTAMPDIFF(MINUTE, '2021-08-04 12:30:45', '2021-08-04 12:45:45'); -- 15
```

Comparison and Logical Operators

Comparison and logical operators are used to compare values and combine multiple conditions in a WHERE clause.

Comparison Operators

Comparison operators are used to compare two values.

NOT EQUAL

- Used to compare if two values are not equal

```
SELECT * FROM my_table WHERE age != 25;
```

GREATER THAN

- Used to compare if a value is greater than another value

```
SELECT * FROM my_table WHERE age > 25;
```

LESS THAN

- Used to compare if a value is less than another value

```
SELECT * FROM my_table WHERE age < 25;
```

GREATER THAN OR EQUAL TO

- Compare if a value is greater than or equal to another value

```
SELECT * FROM my_table WHERE age >= 25;
```

LESS THAN OR EQUAL TO

- Compare if a value is less than or equal to another value

```
SELECT * FROM my_table WHERE age <= 25;
```

BETWEEN

- Used to select values within a range

```
SELECT * FROM my_table WHERE age BETWEEN 20 AND 30;
```

IN

- Used to specify multiple values in a WHERE clause

```
SELECT * FROM my_table WHERE age IN (20, 25, 30);
```

NOT LIKE

- Used to compare if a value is not like another value

```
SELECT * FROM my_table WHERE name NOT LIKE 'J%';
```

IS NULL

- Used to compare if a value is NULL

```
SELECT * FROM my_table WHERE age IS NULL;
```

IS NOT NULL

- Used to compare if a value is not NULL

```
SELECT * FROM my_table WHERE age IS NOT NULL;
```

Logical Operators

Logical operators are used to combine multiple conditions.

AND

- Used to combine multiple conditions

```
SELECT * FROM my_table WHERE age > 20 AND age < 30;
```

OR

- Used to combine multiple conditions

```
SELECT * FROM my_table WHERE age = 20 OR age = 30;
```

NOT

- Used to negate a condition

```
SELECT * FROM my_table WHERE NOT age = 20;
```

CASE Statement

The CASE statement is used to create different outputs based on different conditions.

```
SELECT
  name,
  age,
  CASE
    WHEN age < 20 THEN 'Young'
    WHEN age >= 20 AND age < 30 THEN 'Adult'
    ELSE 'Old'
  END AS age_group
FROM my_table;
```

Alter Table

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

- Add a new column

```
ALTER TABLE my_table ADD email VARCHAR(100);
```

- Delete a column

```
ALTER TABLE my_table DROP COLUMN email;
```

- Rename a table

```
ALTER TABLE my_table RENAME TO new_table;

RENAME TABLE my_table TO new_table;
```

- Renaming a column

```
ALTER TABLE my_table RENAME COLUMN old_name TO new_name;
```

- Modify a column

```
ALTER TABLE my_table MODIFY COLUMN name VARCHAR(200);
```

- Rename column and modify data type

```
ALTER TABLE my_table CHANGE COLUMN old_name new_name VARCHAR(200);
```

- Add or drop constraints

```
ALTER TABLE my_table ADD CONSTRAINT PRIMARY KEY (id);
ALTER TABLE my_table DROP CONSTRAINT PRIMARY KEY;
```

Data Relationships

Data relationships are used to connect data in different tables. There are three types of relationships:

- One-to-One
- One-to-Many
- Many-to-Many

One-to-One

- A record in one table is related to only one record in another table

```
CREATE TABLE users (
  id INT PRIMARY KEY,
  name VARCHAR(100),
  email VARCHAR(100)
);

CREATE TABLE profiles (
  id INT PRIMARY KEY,
  user_id INT,
  bio TEXT,
  FOREIGN KEY (user_id) REFERENCES users(id)
);
```

One-to-Many

- A record in one table is related to many records in another table

```
CREATE TABLE users (
  id INT PRIMARY KEY,
  name VARCHAR(100),
  email VARCHAR(100)
);

CREATE TABLE posts (
  id INT PRIMARY KEY,
  user_id INT,
  title VARCHAR(100),
  body TEXT,
  FOREIGN KEY (user_id) REFERENCES users(id)
);
```

Many-to-Many

- Many records in one table are related to many records in another table

```
CREATE TABLE students (
  id INT PRIMARY KEY,
  name VARCHAR(100)
);

CREATE TABLE courses (
  id INT PRIMARY KEY,
  title VARCHAR(100)
);

CREATE TABLE student_courses (
  student_id INT,
  course_id INT,
  FOREIGN KEY (student_id) REFERENCES students(id),
  FOREIGN KEY (course_id) REFERENCES courses(id)
);
```

Foreign Key

- Used to link two tables together

- A unique key in one table is used to refer to a primary key in another table

```
CREATE TABLE users (
  id INT PRIMARY KEY,
  name VARCHAR(100),
  email VARCHAR(100)
);

CREATE TABLE orders (
  id INT PRIMARY KEY,
  user_id INT,
  FOREIGN KEY (user_id) REFERENCES users(id)
);
```

On Delete Cascade

- Used to delete related records in another table when a record is deleted from a table

```
CREATE TABLE users (
  id INT PRIMARY KEY,
  name VARCHAR(100),
  email VARCHAR(100)
);

CREATE TABLE orders (
  id INT PRIMARY KEY,
  user_id INT,
  FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);
```

Joins

Joins combine rows from two or more tables based on a related column between them.

Cross Joins

- Returns the Cartesian product of the two tables
- It returns all rows from the left table and all rows from the right table

```
SELECT users.name, orders.product
FROM users
CROSS JOIN orders;
```

Inner Joins

- Returns records that have matching values in both tables
- The most common type of join

```
SELECT users.name, orders.product
FROM users
INNER JOIN orders ON users.id = orders.user_id;

-- or

SELECT users.name, orders.product
FROM users
JOIN orders ON users.id = orders.user_id;
```

- Inner join with GROUP BY

```
SELECT users.name, COUNT(orders.product) AS total_products
FROM users
INNER JOIN orders ON users.id = orders.user_id
GROUP BY users.name;
```

Left Joins

- Returns all records from the left table and the matched records from the right table

- The result is NULL from the right side if there is no match

```
SELECT users.name, orders.product
FROM users
LEFT JOIN orders ON users.id = orders.user_id;
```

- Left join with GROUP BY

```
SELECT users.name, COUNT(orders.product) AS total_products
FROM users
LEFT JOIN orders ON users.id = orders.user_id
GROUP BY users.name;
```

Right Joins

- Returns all records from the right table and the matched records from the left table
- The result is NULL from the left side if there is no match

```
SELECT users.name, orders.product
FROM users
RIGHT JOIN orders ON users.id = orders.user_id;
```

- Right join with GROUP BY

```
SELECT users.name, COUNT(orders.product) AS total_products
FROM users
RIGHT JOIN orders ON users.id = orders.user_id
GROUP BY users.name;
```

Views and SQL Modes

Views

- A view is a virtual table based on the result of an SQL statement
- It's a named query stored in the database so that we can reuse it

```
CREATE VIEW my_view AS
SELECT name, age FROM my_table WHERE age > 25;
```

- Can be used as a starting point for other queries

```
SELECT * FROM my_view;
```

- Can be updated

```
CREATE OR REPLACE VIEW my_view AS
SELECT name, age FROM my_table WHERE age > 30;
```

- Can be deleted

```
DROP VIEW my_view;
```

- Can be altered

```
CREATE OR REPLACE VIEW my_view AS
SELECT name, age FROM my_table WHERE age > 35;

-- Alter
ALTER VIEW my_view AS
SELECT name, age FROM my_table WHERE age > 25;
```

With ROLLUP

- Used to add subtotals for each group in a result set

```
SELECT age, name, COUNT(*) FROM my_table GROUP BY age, name WITH ROLLUP;
```

SQL Modes

- SQL modes are used to configure the behavior of the MySQL server
- They can be set globally, per session, or query

```
SET GLOBAL sql_mode = 'modes';
SET SESSION sql_mode = 'modes';
```

- View the current SQL mode

```
SELECT @@GLOBAL.sql_mode;
SELECT @@SESSION.sql_mode;
```

STRICT_TRANS_TABLES

- Prevents us from adding wrong data to a table

Window Functions

Window functions are used to perform calculations across a set of rows related to the current row.

OVER

- Used to define the window of rows that the function operates on

```
SELECT name, age, AVG(age) OVER () AS row_num FROM my_table;
```

PARTITION BY

- Used to divide the result set into partitions to which the function is applied

```
SELECT name, age, OVER (PARTITION BY age) AS row_num FROM my_table;
```

ORDER BY

- Used to sort the rows in each partition

```
SELECT name, age, OVER (PARTITION BY age ORDER BY name) AS row_num FROM my_table;
```

RANK

- Used to assign a rank to each row within a partition of a result set

```
SELECT name, age, RANK() OVER (PARTITION BY age ORDER BY name) AS row_num FROM my_table;
```

DENSE_RANK

- Used to assign a rank to each row within a partition of a result set

```
SELECT name, age, DENSE_RANK() OVER (PARTITION BY age ORDER BY name) AS row_num FROM my_table;
```

ROW_NUMBER

- Used to assign a unique number to each row within a partition of a result set

```
SELECT name, age, ROW_NUMBER() OVER (PARTITION BY age ORDER BY name) AS row_num FROM my_table;
```

LAG and LEAD

- Used to access data from a previous or next row in the result set

```
SELECT name, age, LAG(age) OVER (ORDER BY name) AS prev_age FROM my_table;
SELECT name, age, LEAD(age) OVER (ORDER BY name) AS next_age FROM my_table;
```

NTILE

- Used to divide the result set into a specified number of groups

```
SELECT name, age, NTILE(4) OVER (ORDER BY name) AS group_num FROM my_table;
```

FIRST_VALUE and LAST_VALUE

- Used to get the first and last value in a partition of a result set

```
SELECT name, age, FIRST_VALUE(age) OVER (PARTITION BY name ORDER BY age) AS first_age FROM my_table;
SELECT name, age, LAST_VALUE(age) OVER (PARTITION BY name ORDER BY age) AS last_age FROM my_table;
```

Database Triggers

A trigger is a set of SQL statements that automatically "fires" when a specific event occurs in a database.

BEFORE/AFTER INSERT

- Used to perform an action before or after a new record is inserted into a table

```
DELIMITER $$
  CREATE TRIGGER my_trigger
  BEFORE INSERT ON my_table
  FOR EACH ROW
  BEGIN
    INSERT INTO log_table (message) VALUES ('New record inserted');
  END;
$$
DELIMITER ;
```

BEFORE/AFTER UPDATE

- Used to perform an action before or after a record is updated in a table

```
DELIMITER $$
  CREATE TRIGGER my_trigger
  AFTER UPDATE ON my_table
  FOR EACH ROW
  BEGIN
    INSERT INTO log_table (message) VALUES ('Record updated');
  END;
$$
DELIMITER ;
```

BEFORE/AFTER DELETE

- Used to perform an action before or after a record is deleted from a table

```
DELIMITER $$
  CREATE TRIGGER my_trigger
  BEFORE DELETE ON my_table
  FOR EACH ROW
  BEGIN
    INSERT INTO log_table (message) VALUES ('Record deleted');
  END;
$$
DELIMITER ;
```

Managing Triggers

- View all triggers in a database

```
SHOW TRIGGERS;
```

- View the definition of a trigger

```
SHOW CREATE TRIGGER my_trigger;
```

- Drop a trigger

```
DROP TRIGGER my_trigger;
```

Conclusion

If you found the cheatsheet helpful please check out more of my work on yodkwtf.com or follow me on twitter. I also run a small youtube channel called Yodkwtf Academy.