# SQL CHEAT SHEET

AlyssaDataHub

## BASICS

**SELECT**
Retrieves specific columns or data from a database table.

**FROM**
Specifies the table or source of the data to be queried.

**WHERE**
Filters rows based on specified conditions.

**GROUP BY**
Groups rows with the same values in specified columns for aggregation.

**DISTINCT**
Eliminates duplicate rows from the result set, returning only unique values.

**LIKE**
Filters rows based on a specified pattern, often using wildcards like % and _

**ORDER BY**
Sorts the result set by one or more columns, in ascending or descending order

**HAVING**
Filters grouped data based on aggregate conditions.

**AS**
Assigns an alias to a column or table for readability or convenience.

**LIMIT**
Restricts the number of rows returned by a query.

**BETWEEN**
Filters rows within a specified range, including the boundary values.

**IN**
Filters rows by matching values in a specified list or subquery.

## AGGREGATE FUNCTIONS

**COUNT()**
Returns the number of rows in a group or table, including or excluding **NULLs**.

**SUM()**
Calculates the total of numeric values in a column.

**AVG()**
Returns the average value of a numeric column. (Mean)

**MIN()**
Finds the smallest value in a column.

**MAX()**
Finds the largest value in a column.

**MEDIAN()**
Returns the median value from a group of values.

**MODE()**
Finds the most frequent value in a group.

**VARIANCE()**
Calculates the statistical variance of numeric values in a column.

**STDDEV()**
Calculates the standard deviation of numeric values in a column.

**PERCENTILE_CONT()**
Calculates a continuous percentile value within a group.

**PERCENTILE_DISC()**
Returns the discrete percentile rank within a group.

**GROUP_CONCAT()**
Concatenates values from a group into a single string, with optional delimiters.

**STRING_AGG()**
Concatenates values from a group into a single string, similar to **GROUP_CONCAT**.

# JOINS

### JOIN
Combines rows from two or more tables based on a related column, defaulting to an **INNER JOIN** if no type is specified.

### INNER JOIN
Returns rows where there is a match in both tables based on the specified condition.

### OUTER JOIN
Returns all rows from one or both tables, including unmatched rows, with **NULLs** for missing values; it can be further divided into **LEFT JOIN**, **RIGHT JOIN**, and **FULL OUTER JOIN**.

### LEFT JOIN
Returns all rows from the left table and matched rows from the right table, with **NULLs** for unmatched right-side rows.

### RIGHT JOIN
Returns all rows from the right table and matched rows from the left table, with **NULLs** for unmatched left-side rows.

### ON
Specifies the condition used to match rows between tables in a join operation.

### UNION
Allows you to combine two or more tables if they have a common column.

### FULL OUTER JOIN
Combines all rows from both tables, with **NULLs** for unmatched rows from either table.

### CROSS JOIN
Returns the Cartesian product of two tables, pairing every row from one table with every row from the other.

### SELF JOIN
Joins a table with itself to compare rows within the same table.

### NATURAL JOIN
Automatically joins tables based on all columns with the same name and compatible data types, without requiring an explicit **ON** condition.

# STRING FUNCTIONS

### LENGTH()/CHAR_LENGTH()
Returns the number of characters in a string (including spaces); **CHAR_LENGTH()** focuses on characters, ignoring byte differences.

### UPPER()/LOWER()
Converts all characters in a string to uppercase **(UPPER()**) or lowercase (**LOWER()**)

### TRIM()/LTRIM()/RTRIM()
Removes whitespace or specified characters:

- **TRIM()** removes from both ends.
- **LTRIM()** removes from the start.
- **RTRIM()** removes from the end.

### SUBSTRING()/LEFT()/RIGHT()
Extracts a portion of a string:

- **SUBSTRING()** extracts based on position and length.
- **LEFT()** retrieves characters from the start.
- **RIGHT()** retrieves characters from the end.

### REPLACE()
Replaces all occurrences of a substring with another substring in a string.

### LOCATE()/POSITION()
Finds the position of the first occurrence of a substring in a string.

### CONCAT()
Combines two or more strings into one.

### REVERSE()
Reverses the characters in a string.

### REPEAT()
Repeats a string a specified number of times.

### FORMAT()
Formats a number as a string with commas or other formatting.

# CASE STATEMENT

Require a **WHEN** clause for conditions, a **THEN** clause for results, an **END** statement to complete the logic, and can optionally use **AS** to rename the output.

**EXAMPLE:** WHEN (condition) THEN (result)

# TEMPORARY TABLES

### CREATE TEMPORARY TABLE
Creates a table that exists only for the duration of the database session or transaction

### INSERT INTO
Inserts data into the temporary table, similar to a regular table.

### SELECT INTO
Creates and populates a temporary table with the results of a query.

### SHOW TABLE
Lists all tables, including temporary tables, in the current session.

### DELETE
Removes rows from the temporary table

### DROP TABLE
Deletes the temporary table before the session ends or when it is no longer needed.

### TRUNCATE TABLE
Removes all rows from the temporary table without logging individual row deletions.

### UPDATE
Modifies data in the temporary table.

# Data Types

## NUMERICAL

**INT**
Stores whole numbers.

**BIGINT**
Stores larger whole numbers than **INT**.

**SMALLINT**
Stores smaller whole numbers than **INT**.

**TINY INT**
Stores very small whole numbers.

**DECIMAL/NUMERIC**
Stores fixed-point numbers with precision (p) and scale (s).

**FLOAT**
Stores approximate floating-point numbers.

**DOUBLE/REAL**
Stores double-precision floating-point numbers.

**BIT**
Stores binary values (0 or 1).

## STRING

**VARCHAR()**
Stores variable-length text, up to the specified size.

**CHAR()**
Stores fixed-length text, padding shorter values with spaces.

**TEXT**
Stores long text data (size limits vary by database).

**NVARCHAR()**
Stores variable-length Unicode text.

**NCHAR()**
Stores fixed-length Unicode text.

**BLOB**
Stores binary large objects, such as images or files.

## DATE/TIME

**DATE**
Stores dates in the format YYYY-MM-DD

**TIME**
Stores time in the format HH:MM:SS.

**DATETIME**
Stores combined date and time in the format YYYY-MM-DD HH:MM:SS.

**TIMESTAMP**
Stores a date and time, often used for tracking changes.

**YEAR**
Stores year values (e.g., YYYY).

NOT ALL DATA TYPES ARE INCLUDED, BUT THESE ARE THE BASICS

# SUBQUERIES

### ANY/SOME
Compares a value to any value returned by the subquery and evaluates to **TRUE** if at least one match exists.

### ALL
Compares a value to all values returned by the subquery and evaluates to **TRUE** only if the condition is true for every value.

### EXISTS
Returns **TRUE** if the subquery produces at least one row, often used to check for the existence of related data

## TYPES OF SUBQUERIES

**SCALAR SUBQUERIES**
Returns a single value, often used in SELECT or WHERE clauses.

**TABLE SUBQUERIES**
Returns a result set, used in the FROM clause as a derived table.

**CORRELATED SUBQUERIES**
References columns from the outer query, evaluated row-by-row.

**UNCORRELATED SUBQUERIES**
Independent of the outer query and evaluated once.

# WINDOW FUNCTIONS

**OVER**
Defines the window (set of rows) over which a window function operates.

**PARTITION BY**
Divides the result set into subsets (partitions) for applying the window function independently within each subset.

**RANK()**
Assigns a unique rank to each row within a partition, with gaps if there are ties.

**DENSE_RANK()**
Assigns a unique sequential number to each row within a partition, starting from 1.

**ROW_NUMBER()**
Assigns a unique rank to each row within a partition, without gaps for ties.

**NTILE(N)**
Divides rows in a partition into n equal groups and assigns a group number to each row.

**LEAD**
Retrieves the value of a column from the next row in the partition.

**LAG()**
Retrieves the value of a column from the previous row in the partition.

**FIRST_VALUE()/LAST_VALUE()**
Allows you to combine two or more tables if they have a common column.

# STORED PROCEDURES

Stored procedures are precompiled sets of SQL statements stored in the database that can be executed repeatedly to perform specific tasks, automate processes, and improve query efficiency.

**CREATE PROCEDURE**
Defines a new stored procedure in the database, encapsulating a set of SQL statements.

**BEGIN/END**
Marks the start and end of the procedural logic inside a stored procedure.

**DELIMITER**
Changes the default statement delimiter (;) to avoid conflicts when defining stored procedures.

**CALL**
Executes a stored procedure that has been defined in the database.

**P_ (PREFIX)**
A naming convention often used to indicate stored procedures for clarity and organization.

**CURSORS**
Used within stored procedures to iterate through rows in a query result set one row at a time

**STORED PROCEDURE**
A precompiled set of SQL statements stored in the database for repeated use.

# TRIGGERS AND EVENTS

Triggers automatically execute SQL code in response to specific changes in a table (e.g., inserts, updates, deletes), while Events are scheduled tasks that execute SQL code at a specified time or interval. Both are used to automate processes and enforce rules within the database.

**CREATE TRIGGER**
Defines a trigger that executes automatically in response to specified database events.

**AFTER INSERT ON**
Specifies the table and event (INSERT) that activates the trigger after the data is inserted.

**FOR EACH ROW**
Ensures the trigger executes for every affected row in the table.

**BEGIN/END**
Delimits the procedural logic inside the trigger or event.

**INSERT INTO**
Used within a trigger to insert data into another table.

**VALUES(NEW/OLD)**
Refers to the new values being inserted or the old values being modified in the trigger's execution.

**CREATE EVENT**
Defines a scheduled task to execute SQL code at a specific time or interval.

**ON SCHEDULE**
Specifies the timing for an event, such as a one-time or recurring schedule.

**DO**
Specifies the SQL action(s) to be executed as part of an event.

# CTEs

CTEs (Common Table Functions) simplify complex queries by defining temporary, reusable result sets that can be referenced within the same query. They improve readability and are particularly useful for breaking down large queries, performing hierarchical operations, and creating recursive logic.

### WITH

Defines a Common Table Expression (CTE) as a temporary, reusable result set within a query.

### RECURSIVE

Enables recursive CTEs, allowing repeated execution of the query to process hierarchical or iterative data structures until a termination condition is met.

## BASICS

### SELECT

Allows you to combine two or more tables if they have a common column.

## BASICS

### SELECT

Allows you to combine two or more tables if they have a common column.

### FROM

Allows you to combine two or more tables if they have a common column.

### WHERE

Allows you to combine two or more tables if they have a common column.

### GROUP BY

Allows you to combine two or more tables if they have a common column.

### ORDER BY

Allows you to combine two or more tables if they have a common column.

### HAVING

Allows you to combine two or more tables if they have a common column.

### AS

Allows you to combine two or more tables if they have a common column.

### LIMIT

Allows you to combine two or more tables if they have a common column.

## JOINS

### JOIN

Allows you to combine two or more tables if they have a common column.

### INNER JOIN

Allows you to combine two or more tables if they have a common column.

### OUTER JOIN

Allows you to combine two or more tables if they have a common column.

### LEFT JOIN

Allows you to combine two or more tables if they have a common column.

### RIGHT JOIN

Allows you to combine two or more tables if they have a common column.

### ON

Allows you to combine two or more tables if they have a common column.

### UNION

Allows you to combine two or more tables if they have a common column.

### INNER JOIN

Allows you to combine two or more tables if they have a common column.