Data Analysis with SQL

MySQL Cheat Sheet

Created By Ram Kedem, Shuki Molk, Dotan Entin, and Elad Peleg

Basic SQL Statements	
Select all	`SELECT * FROM table`
Select specific columns	`SELECT column1, column2 FROM table`
Arithmetic operations	`SELECT column + value FROM table`
String concatenation	`SELECT CONCAT(string_column,' ',string_column) FROM table`
Column alias	`SELECT column AS 'alias'`
Distinct values of a single column	`SELECT DISTINCT column FROM table`
Distinct values of multiple Columns	`SELECT DISTINCT column, column FROM table`
Quote column name in case it contains spaces, punctuation or conflicts with a reserved keyword	`SELECT `column_name` `

Filter the Dataset	
Specify a numeric value	`5`
Specify a string value	`'string'`
Specify a date value	`'2019-05-28'`
Basic operators	`WHERE column = value (or >, <, >=, <=, !=)`
IN	`WHERE column IN (value1, value2, value3)`
BETWEEN	`WHERE column BETWEEN value1 AND value2`
LIKE	`WHERE column LIKE 'pattern'`
IS NULL	`WHERE column IS NULL`
IS NOT NULL	`WHERE column IS NOT NULL`
AND	`WHERE condition1 AND condition2`
OR	`WHERE condition1 OR condition2`

Sort the Result Set	
ORDER BY a single column ascending	`ORDER BY column`
ORDER BY a single column descending	`ORDER BY column DESC`
ORDER BY multiple columns	`ORDER BY column1, column2 DESC ..`

Limit the Result Set	
Retrieves first N rows	`SELECT ... LIMIT N`
TOP N Analysis	`SELECT .. ORDER BY .. LIMIT N`

Common String Related Functions	
Returns the **right** part of a string	`SELECT RIGHT('hello' , 2) → 'lo'`
Returns the **left** side of a string	`SELECT LEFT('hello', 2) → 'he'`
Returns the **number** of characters in a string	`SELECT LENGTH('hello') → 5`
Replaces all occurrences of a given substring	`REPLACE('hello world' ,'l', '*') → 'he**o wor*d'`
Reverses a string	`REVERSE('hello') → 'olleh'`
Returns a **substring** of a string	`SUBSTRING('hello world' , 2, 3) → 'ell'`
Returns a string in **lower-case**	`LOWER('HELLO') → 'hello'`
Returns a string in **upper-case**	`UPPER('hello') → 'HELLO'`
Returns the **position** of a substring in a string	`POSITION('e' IN 'hello') → 2`

Common Numeric Functions	
Rounds the number	`ROUND(92.56, 1) → 92.6`
Rounds a number **downwards** the nearest integer	`FLOOR(92.56) → 92`
Rounds a number **upwards** the nearest integer	`CEIL(92.56) → 93`
Returns the **absolute** value of a number	`ABS(-28) → 28`
Returns the **square root** of a number	`SQRT(100) → 10`
Returns a number raised to the **power** of another	`POWER(10, 2) → 100`

Converting Values using CAST	
Convert a value to an int datatype	`CAST(5.25 as SIGNED) → 5`
Convert a value to a char datatype:	`CAST(5.25 as CHAR(3)) → '5.25'`
Convert a value to a date datatype	`CAST('2020-01-25' AS DATE)`

Common Date Related Functions	
Returns the **current** database date	`CURDATE()`
Adds a time/date interval to a date	`DATE_ADD ("2020-01-24", INTERVAL 1 YEAR) → 2021-01-24`
Return the **difference** between two date values	`TIMESTAMPDIFF (MONTH, '2020-01-24', '2020-04-24') → 3`
Returns the **year** of a specified date	`YEAR('2020-01-24') → 2020`
Returns the **month** of a specified date	`MONTH('2020-01-24') → 01`
Returns the **day** of a specified date	`DAY('2020-01-24') → 24`

Common Null Handling Functions	
Returns the specified value IF the expression is NULL, otherwise return the expression	`IFNULL(column, value_to_return_if_null)`

Conditional Expressions	

Goes through a series of conditions and returns a value when the first condition is met	`CASE`
	`  WHEN condition1 THEN result1`
	`  WHEN condition2 THEN result2`
	`  WHEN conditionN THEN resultN`
	`  ELSE result`
	`END;`

Common Group Operations	
Returns the **average**	`AVG()`
Returns the **minimum**	`MIN()`
Returns the **maximum**	`MAX()`
Returns the **sum**	`SUM()`
Counts the number of rows in a table	`COUNT(*)`
Counts the number of values in a column	`COUNT(column)`
Counts the number of distinct values in a column	`COUNT(DISTINCT column)`
Divides the query result into groups of rows	`GROUP BY column, column…`
Filter condition based on a group or aggregate	`HAVING <condition>`
Returns the aggregation result for each row in the table	`agg_function() OVER ()`
Returns the aggregated results for each partition, in each row (of the same partition)	`agg_function() OVER (PARTITION BY .. )`
Returns the cumulative aggregated results	`agg_function() OVER (ORDER BY.. )`
Returns the cumulative aggregated results in each partition	`agg_function() OVER (PARTITION BY.. ORDER BY..)`

Syntax vs Execution Order

Writing	Execution
SELECT	FROM (Joins included)
FROM (JOINs included)	WHERE
WHERE	GROUP BY
GROUP BY	HAVING
HAVING	SELECT
ORDER BY	ORDER BY
LIMIT	LIMIT

JOIN Operations

Inner	FROM table1 t1 INNER JOIN table2 t2 ON <condition>
Full outer	FROM table1 t1 LEFT OUTER JOIN table2 t2 ON <condition> *UNION* FROM table1 t1 RIGHT OUTER JOIN table2 t2 ON <condition>
Outer Left	FROM table1 t1 LEFT OUTER JOIN table2 t2 ON <condition>
Outer Right	FROM table1 t1 RIGHT OUTER JOIN table2 t2 ON <condition>

Subqueries in the WHERE Clause

Single row Subqueries	WHERE column = (INNER QUERY)
Comparing against multiple values	WHERE column IN (INNER QUERY)

CTE

A common table expression (CTE) is a named temporary result set that exists within the scope of a single statement and that can be referred to later within that statement, possibly multiple times

```
WITH expression_name [ ( column_name [,...n] ) ]
AS
( CTE_query_definition )
```

SET Operators

Combines the result set of two or more SELECT statements (allows duplicate values)	SELECT … FROM table_1 **UNION ALL** SELECT … FROM table_2
Combines the result set of two or more SELECT statements (only distinct values)	SELECT … FROM table_1 **UNION** SELECT … FROM table_2
Returns the intersection of two SELECT statements (Emulates INTERSECT using IN and subquery)	SELECT .. FROM table_1 WHERE *col* **IN (**SELECT *col* FROM table_2**)**
Returns any distinct values from the query left of the EXCEPT operator (Emulates EXCEPT using NOT IN and subquery)	SELECT .. FROM table_1 WHERE *col* **NOT IN** **(**SELECT *col* FROM table_2**)**

Ranking Functions	
Returns the rank of each row within the partition of a result set. The rank of a row is one plus the number of ranks that come before the row in question.	**RANK()** `OVER (PARTITION BY.. ORDER BY..)`
Returns the rank of each row within a result set partition. The rank of a specific row is one plus the number of distinct rank values that come before that specific row.	**DENSE_RANK()** `OVER (PARTITION BY.. ORDER BY..)`
Returns the sequential number of a row within a partition of a result set, starting at 1	**ROW_NUMBER()** `OVER (PARTITION BY.. ORDER BY..)`
Divides the result set produced by the FROM clause into partitions	**NTILE(n)** `OVER (PARTITION BY.. ORDER BY..)`

Analytic Functions	
Accesses data from a previous row in the same result	`LAG(column) OVER (PARTITION BY.. ORDER BY..)`
Accesses data from a subsequent row in the same result set	`LEAD(column) OVER (PARTITION BY.. ORDER BY..)`

Essential Data Types	
String Data Types	**Description**
CHAR(number)	A fixed number of characters
VARCHAR(number / MAX)	A variable number of characters
Numeric Data Types	**Description**
TINYINT	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255
SMALLINT	A small integer. Signed range is from -327,68 to 32,767. Unsigned range is from 0 to 65,535
INT	A medium integer. Signed range is from -21,474,83,648 to 2,147,483,647. Unsigned range is from 0 to 4,294,967,295.
BIGINT	A large integer. Signed range is from -9,223,372,036,854,775,808 to 9223372036854775807. Unsigned range is from 0 to 18,446,744,073,709,551,615.
DECIMAL(p,s)	An exact fixed-point number. p = total number of digits, s = number of decimal digits. I.e 123.4567 → p=7, s=4
BOOL / BOOLEAN	Zero is considered as false, nonzero values are considered as true.
Date Data Types	**Description**
DATETIME	Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.
DATE	Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'