Data Analysis with SQL

PostgreSQL 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 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"`

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	`RIGHT('hello' , 2) → 'lo'`
Returns the **left** side of a string	`LEFT('hello', 2) → 'he'`
Returns the **number** of characters in a string	`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 & Operations	
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	`CEILING(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`
If an integer *dividend* is divided by an integer *divisor*, the result is an integer	`5/2 → 2`
Return a Decimal output from dividing two integers	`5/(CAST 2 AS DECIMAL) → 2.5`

Converting Values using CAST	
Convert a value to an int datatype	`CAST(5.25 as INT) → 5`
Convert a value to a varchar datatype:	`CAST(5.25 as VARCHAR) → '5.25'`
Convert a value to a date datatype	`CAST('2020-01-25' AS DATE)`
Convert a value to a decimal datatype	`CAST(5 AS DECIMAL)`

Common Date Related Functions	
Returns the **current** database date	`CURRENT_DATE`
Adds a time/date interval to a date	`CURRENT_DATE + INTERVAL '1 DAY'`
Return the **difference** between two date values	Depends on the exact diff calculation, you can use various expressions or UDFs
Returns the **year** of a specified date	`DATE_PART('year', CURRENT_DATE)`
Returns the **month** of a specified date	`DATE_PART('month', CURRENT_DATE)`
Returns the **day** of a specified date	`DATE_PART('day', CURRENT_DATE)`

Common Null Handling Functions	
Returns the specified value IF the expression is NULL, otherwise return the expression	`COALESCE(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

Subqueries in the WHERE Clause

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

JOIN Operations

Inner	`FROM table1 t1 INNER JOIN table2 t2 ON <condition>`
Full outer	`FROM table1 t1 FULL 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>`

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	SELECT … FROM table_1 **INTERSECT** SELECT … FROM table_2
Returns any distinct values from the query left of the EXCEPT operator	SELECT … FROM table_1 **EXCEPT** SELECT … 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..)`

PIVOT

PIVOT rotates a table-valued expression by turning the unique values from one column in the expression into multiple columns in the output

```
SELECT ..
FROM (SELECT query that produces the data for axis) AS alias
PIVOT
     (aggregate_function (column)
      FOR x_axis_column IN (list of values)
     ) AS alias
```

UNPIVOT

UNPIVOT carries out the opposite operation to PIVOT by rotating columns of a table-valued expression into column values

```
SELECT ..
FROM  (SELECT columns participating in the process) AS alias
UNPIVOT
      (column_representing_z_values
       FOR
       column_representing_x_values IN (list of values..
      ) AS alias
```

Essential Data Types	
String Data Types	Description
CHAR(number)	A fixed number of characters
VARCHAR(number)	A variable number of characters
Numeric Data Types	Description
SMALLINT	-32768 to +32767
INTEGER	-2147483648 to +2147483647
BIGINT	Integers between (-9,223,372,036,854,775,808) and 9,223,372,036,854,775,807
DECIMAL(p,s)	Numbers from (-10^38 +1) to (10^38 –1) p = total number of digits, s = number of decimal digits. I.e 123.4567 → p=7, s=4
NUMERIC(p,s)	numeric is functionally identical to decimal
Date Data Types	Description
TIMESTAMP	With / without time zone, accuracy of 1 microsecond
DATE	Accuracy of 1 day