SQL Cheat Sheet: FUNCTIONS and Implicit JOIN

Command	Syntax	Description	Example
COUNT	`SELECT COUNT(column_name) FROM table_name WHERE condition;`	`COUNT` function returns the number of rows that match a specified criterion.	`SELECT COUNT(dep_id) FROM employees;`
AVG	`SELECT AVG(column_name) FROM table_name WHERE condition;`	`AVG` function returns the average value of a numeric column.	`SELECT AVG(salary) FROM employees;`
SUM	`SELECT SUM(column_name) FROM table_name WHERE condition;`	`SUM` function returns the total sum of a numeric column.	`SELECT SUM(salary) FROM employees;`
MIN	`SELECT MIN(column_name) FROM table_name WHERE condition;`	`MIN` function returns the smallest value of the SELECTED column.	`SELECT MIN(salary) FROM employees;`
MAX	`SELECT MAX(column_name) FROM table_name WHERE condition;`	`MAX` function returns the largest value of the SELECTED column.	`SELECT MAX(salary) FROM employees;`
ROUND	`SELECT ROUND(2number, decimals, operation) AS RoundValue;`	`ROUND` function rounds a number to a specified number of decimal places.	`SELECT ROUND(salary) FROM employees;`
LENGTH	`SELECT LENGTH(column_name) FROM table;`	`LENGTH` function returns the length of a string (in bytes).	`SELECT LENGTH(f_name) FROM employees;`
UCASE	`SELECT UCASE(column_name) FROM table;`	`UCASE` function displays the column name in each table in uppercase.	`SELECT UCASE(f_name) FROM employees;`
LCASE	`SELECT LCASE(column_name) FROM table;`	`LCASE` function displays the column name in each table in lowercase.	`SELECT LCASE(f_name) FROM employees;`
DISTINCT	`SELECT DISTINCT column_name FROM table;`	`DISTINCT` function is used to display data without duplicates.	`SELECT DISTINCT UCASE(f_name) FROM employees;`
DAY	`SELECT DAY(column_name) FROM table`	`DAY` function returns the day of the month for a given date.	`SELECT DAY(b_date) FROM employees where emp_id = 'E1002';`
CURRENT_DATE	`SELECT CURRENT_DATE;`	`CURRENT_DATE` is used to display the current date.	`SELECT CURRENT_DATE;`
DATEDIFF()	`SELECT DATEDIFF(date1, date2);`	`DATEDIFF()` is used to calculate the difference between two dates or time stamps. The default value generated is the difference in number of days.	`SELECT DATEDIFF(CURRENT_DATE, date_column) FROM table;`
FROM_DAYS()	`SELECT FROM_DAYS(number_of_days);`	`FROM_DAYS()` is used to convert a given number of days to YYYY-MM-DD format.	`SELECT FROM_DAYS(DATEDIFF(CURRENT_DATE, date_column)) FROM table;`
DATE_ADD()	`SELECT DATE_ADD(date, INTERVAL n type);`	`DATE_ADD()` is used to calculate the date after lapse of mentioned number of units of date type, i.e. if n=3 and type=DAY, the result is a date 3 days after what is mentioned in date column. The type valiable can also be months or years.	`SELECT DATE_ADD(date, INTERVAL 3 DAY);;`
DATE_SUB()	`SELECT DATE_SUB(date, INTERVAL n type);`	`DATE_SUB()` is used to calculate the date prior to the record date by mentioned number of units of date type, i.e. if n=3 and type=DAY, the result is a date 3 days before what is mentioned in date column. The type valiable can also be months or years.	`SELECT DATE_SUB(date, INTERVAL 3 DAY);;`

Subquery	`SELECT column_name [, column_name ] FROM table1 [, table2 ] WHERE column_name OPERATOR (SELECT column_name [, column_name ] FROM table1 [, table2 ] [WHERE])`	`Subquery` is a query within another SQL query and embedded within the WHERE clause.
		A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.
			`SELECT emp_id, fname, lname, salary FROM employees where salary < (SELECT AVG(salary) FROM employees);`
			
			`SELECT * FROM ( SELECT emp_id, f_name, l_name, dep_id FROM employees) AS emp4all;`
			
			`SELECT * FROM employees WHERE job_id IN (SELECT job_ident FROM jobs);`

Implicit Inner Join	`SELECT column_name(s) FROM table1, table2 WHERE table1.column_name = table2.column_name;`	`Implicit Inner Join` combines two or more records but displays only matching values in both tables. Inner join applies only the specified columns.	`SELECT * FROM employees, jobs where employees.job_id = jobs.job_ident;`
Implicit Cross Join	`SELECT column_name(s) FROM table1, table2;`	`Implicit Cross Join` is defined as a Cartesian product where the number of rows in the first table is multiplied by the number of rows in the second table.	`SELECT * FROM employees, jobs;`

Author(s)

Lakshmi Holla
Abhishek Gagneja

Skills Network

IBM

Changelog

Date	Version	Changed by	Change Description
2023-10-03	1.3	Steve Hord	QA pass with edits
2023-10-01	1.2	Abhishek Gagneja	Updated the document
2023-05-04	1.1	Benny Li	Formatting changes
2021-07-28	1.0	Lakshmi Holla	Initial Version