# SQL Aggregation Tutorial

*[Set up your tutorial in the same way as you have for the previous objectives if you are following along on your own DB.]*

Up to this point, all the queries you have written have involved identifying a set of **individual records** that you need to answer a particular question. “Which employees work in the ‘Customer Service’ department?” for example. Here you tell the database to return any record that matches the value ‘Customer Service’ in its ‘department’ attribute.

Some questions, however, can’t be answered just by matching criteria against the values contained in individual records. Sometimes you need to examine sets of records or even an entire table **together**, **collectively** to answer your questions. For example:

* What is the average of all of the salaries in the ‘Marketing’ department?
* Who earns the highest salary in the company?
* What is the total cost of all of the items ordered in order #100356?

In each of these three examples, you need to determine a value by examining a set of values together: which is the biggest or smallest, what is the average, or what is the sum. Examining individual values in isolation will never enable you to answer these types of questions. Luckily SQL’s aggregation functions enable you to address exactly these sort of questions.

## Basic SQL Aggregate Functions

The basic SQL aggregate functions include the following:

AVG() - Returns the average value of a set

COUNT() - Returns the number of rows in a set

MAX() - Returns the largest value in a set

MIN() - Returns the smallest value in a set

SUM() - Returns the sum of values in a set

## Applying the Aggregate Functions to an Entire Table

Let’s take a look at the ‘employees’ database to illustrate the use of aggregate functions.

1. Output the average of all of the salary values in the salary table.

SELECT AVG(salary) FROM salary;

1. Output the highest salary in the salary table.

SELECT MAX(salary) FROM salary;

1. Output the lowest salary.

SELECT MIN(salary) FROM salary;

1. How many salaries are between $50,000 and $75,000?

SELECT COUNT(salary) FROM salary WHERE salary <75000 AND salary >50000;

+---------------+

| COUNT(salary) |

+---------------+

| 6020 |

+---------------+

1 row in set (0.00 sec)

Note in this query that you can still add a ‘WHERE’ condition to filter which records are going to be included in your results.

## Aggregating According to Groups

Where things start to get really interesting with aggregate functions is when you add groupings using the GROUP BY clause. GROUP BY allows you to group the results of your aggregating into results sets according to the value of one or more columns. An example should help explain what this means.

Looking at the ‘title’ table, say we want to count the number of employees who have each of our 6 different titles. Using the COUNT aggregate function together with a GROUP BY statement, we can do exactly that:

1. How many people are there with each of the unique titles in the title table?

SELECT title, COUNT(title) FROM title GROUP BY title;

+--------------------+----------+

| TITLE | COUNT |

+--------------------+----------+

| ASSISTANT ENGINEER | 63 |

| ENGINEER | 464 |

| SENIOR ENGINEER | 407 |

| SENIOR STAFF | 372 |

| STAFF | 459 |

| TECHNIQUE LEADER | 75 |

+--------------------+----------+

6 rows in set (0.00 sec)

Grouping by title enables the counting to occur per value in the title attribute instead of across the entire table. If you sum the counts across all titles, you can confirm that it does include all 1,840 rows in the table.

The counts above include both past and present job positions. Suppose we wanted to include only current positions (where to\_date is NULL, indicating that they still currently hold the position)?

1. How many people are there that currently hold each of the unique titles in the title table?

SELECT title, COUNT(title) FROM title WHERE to\_date IS NULL GROUP BY title;

+--------------------+--------------+

| TITLE | COUNT(title) |

+--------------------+--------------+

| ASSISTANT ENGINEER | 15 |

| ENGINEER | 137 |

| SENIOR ENGINEER | 352 |

| SENIOR STAFF | 326 |

| STAFF | 112 |

| TECHNIQUE LEADER | 61 |

+--------------------+--------------+

6 rows in set (0.00 sec)

This query shows that we can still filter records using the WHERE clause when you are doing aggregation. Keep in mind that this filters the records BEFORE the aggregation happens.

You can also filter the aggregated results AFTER the aggregation is done by adding a HAVING clause. Suppose we were interested only in those titles currently held by a large number of people, defined as those having more than 200 people currently in them:

1. How many people are currently in each of titles that have more than 200 people in them?

SELECT title, COUNT(title) FROM title WHERE to\_date IS NULL GROUP BY title HAVING COUNT(title) > 200;

+-----------------+--------------+

| TITLE | COUNT(TITLE) |

+-----------------+--------------+

| SENIOR ENGINEER | 352 |

| SENIOR STAFF | 326 |

+-----------------+--------------+

2 rows in set (0.00 sec)

## Bringing it All Together

Now that you are familiar with the basic functions of aggregation and the SQL clauses they entail, let’s bring it all together with the rest of what you have learned to date to address a more complex, realistic query:

1. What is the average current salary for all of the employees in each of the job title categories and how many employees are currently in each? Order your results from highest to lowest total number of employees and exclude any job title that have fewer than 100 employees currently working in them.

SELECT title, ROUND(AVG(salary)), COUNT(salary.emp\_no) FROM salary JOIN title USING (emp\_no) WHERE title.to\_date IS NULL GROUP BY title HAVING count(salary.emp\_no) >= 100 ORDER BY COUNT(salary.emp\_no) DESC;

+------------------+--------------------+----------------------+

| TITLE | ROUND(AVG(SALARY)) | COUNT(SALARY.EMP\_NO) |

+------------------+--------------------+----------------------+

| SENIOR ENGINEER | 62048 | 4202 |

| SENIOR STAFF | 69945 | 3906 |

| ENGINEER | 56067 | 906 |

| STAFF | 62335 | 658 |

| TECHNIQUE LEADER | 61824 | 609 |

+------------------+--------------------+----------------------+

5 ROWS IN SET (0.02 SEC)

Finally, and unrelated to aggregation per se but especially useful nonetheless when doing aggregation, note that you can use aliases for categories to improve the readability of your SQL output and to save yourself some typing. To do that, the query above can be re-written as follows:

SELECT title, ROUND(AVG(salary)) AS AVERAGE, COUNT(salary.emp\_no) AS COUNT FROM salary JOIN title USING (emp\_no) WHERE title.to\_date IS NULL GROUP BY title HAVING COUNT >= 100 ORDER BY COUNT DESC;

+------------------+---------+-------+

| TITLE | AVERAGE | COUNT |

+------------------+---------+-------+

| SENIOR ENGINEER | 62048 | 4202 |

| SENIOR STAFF | 69945 | 3906 |

| ENGINEER | 56067 | 906 |

| STAFF | 62335 | 658 |

| TECHNIQUE LEADER | 61824 | 609 |

+------------------+---------+-------+

5 rows in set (0.02 sec)

Note that the only change from the previous query is that the titles for the last two columns have been replaced with the aliases, which improves readability.

There are lots of moving parts in a query like this, but if you are familiar with all the individual component parts and what they do, and you are systematic in working through it, you should at this point be able either to write such a query or to interpret one already written. Let me know if you are having trouble.