



UNIVERSITY OF  
**TORONTO**  
MISSISSAUGA

---

# Tutorial 5

CSC 343

Winter 2018

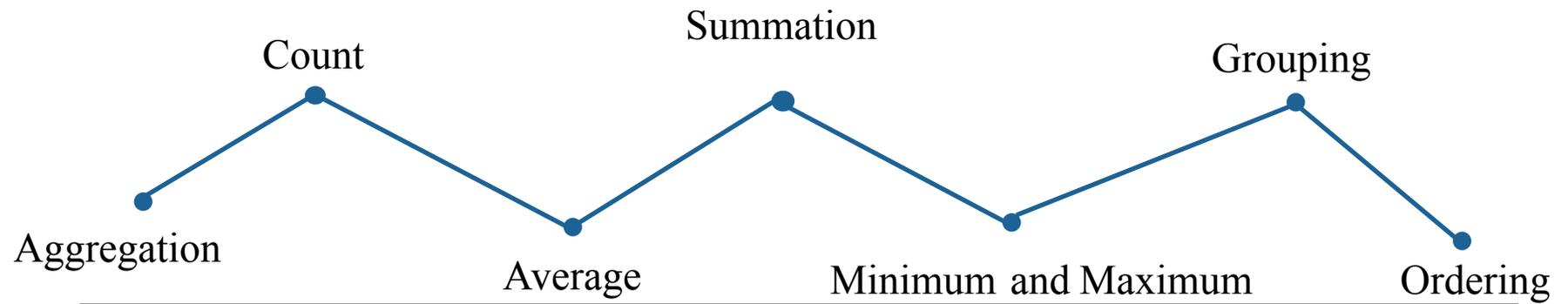
---

NAFIZ HOSSAIN ([NAFIZ.HOSSAIN@UTORONTO.CA](mailto:NAFIZ.HOSSAIN@UTORONTO.CA))

# Table of Contents



UNIVERSITY OF  
**TORONTO**  
MISSISSAUGA





## Aggregation

---

There are two rules you must follow when aggregating:

- 1 Aggregation functions can be used in both the SELECT and HAVING clauses (the HAVING clause will be discussed at another time).
  - 2 Aggregation functions cannot be used in a WHERE clause.
- 



RECALL: Aggregation is a column procedure, not a row operation!

# Aggregation Overview



<u>Function Syntax</u>	<u>Function Usage</u>
COUNT( [ALL   DISTINCT] expression )	The number of (distinct) non-NULL values in a column/expression.
COUNT(*)	The number of selected rows.
AVG( [ALL   DISTINCT] expression )	The average of the (distinct) values in a numeric column/expression.
SUM( [ALL   DISTINCT] expression )	The total of the (distinct) values in a numeric column/expression.
MAX(expression)	The largest value in a column/expression.
MIN(expression)	The smallest value in a column/ expression.

# Count



UNIVERSITY OF  
**TORONTO**  
MISSISSAUGA

- `Count(*)` will count all of the rows in a table.
- `Count(columnName)` will count a specified column.
  - Rows containing `NULL` (unknown) values are omitted.
  - An empty or zero value is not to be confused with `NULL`.

# Count



- Count(\*) returns a single scalar value.  
note: \* is the parameter used in the function.
- e.g. let's say a manager of a company needs to know the number of employees that work within the organization, count(\*) can produce this information.
  - Let's say that the HR department has a table of all "employee[s]".

```
SELECT COUNT (*)  
FROM employee;
```

```
COUNT (*)  
-----  
8
```

# Count



- The result is “8”, a single scalar value.
- Notice that the result table column heading, in this case “COUNT(\*)” for example, is not the most meaningful name.
- The output column name can be made more meaningful through query manipulation.
  - This is accomplished by using the “as” keyword.

```
SELECT COUNT(*)  
FROM employee;
```

```
COUNT(*)
```

```
-----
```

```
8
```

```
SELECT COUNT(*)  
AS numOfEmps  
FROM employee;
```

```
numOfEmps
```

```
-----
```

```
8
```

# Average



- `AVG(columnName)` returns the average for that column.
- e.g. let's say a manager of a company needs to know the average salary of their employees.

```
SELECT AVG(empSalary)
AS Average_Employee_Salary
FROM employee;

Average_Employee_Salary
-----
$79,575
```

# Summation



- SUM(columnName) returns the computed total summation for that column.
- e.g. let's say a manager of a company needs to know the average salary of their employees.

```
SELECT SUM(empSalary)
AS Total_Employee_Salary
FROM employee;
```

```
Total_Employee_Salary
-----
$1,875,750
```

# Summation



- Who or when would use SUM in a query?
  - e.g. let's say that you are a finance officer for the Faculty of Engineering and that your responsibility is to prepare budgets and expense reports for various departments. You were tasked with computing are required to specify what each department's salary expense is.

```
SELECT SUM(empSalary)
AS Total_CS_Employee_Salary
FROM employee
WHERE empDept = 'CS';
```

```
Total_CS_Employee_Salary
-----
$550,000
```

# Minimum and Maximum



UNIVERSITY OF  
**TORONTO**  
MISSISSAUGA

- The MIN function will return the smallest value stored in the search column.
- The MAX function will return the largest value stored in the search column.
- Unlike AVG and SUM, MIN and MAX work with both numeric and character data.
- e.g. let's say that you you want to know who from your employee table comes first alphabetically by last name and who comes last.

# Minimum and Maximum



e.g. let's say that you you want to know who from your employee table comes first alphabetically by last name and who comes last.

- MIN will return the employee row that comes first alphabetical (aka is the smallest).
- MAX will return the employee row that comes last alphabetical (aka is the largest).

```
SELECT MIN(empSurname), MAX(empSurname)
FROM employee;
```

MIN (EMPSURNAME)	MAX (EMPSURNAME)
-----	-----
Amin	Zhu

# Minimum and Maximum



- The same idea can be said if we look at a numeric value.
  - e.g. let's say that you you want to know who from your employee table has the largest and smallest salary.

```
SELECT MIN(empSalary),MAX(empSalary)
AS Least_Paid, Most_Paid
FROM employee;
```

Least_Paid	Most_Paid
-----	-----
\$41,512	\$315,945

# Grouping



- Aggregation functions are more powerful when utilized with the GROUP BY clause.
- In fact, the GROUP BY clause is rarely used without an aggregation function.
  - I know what you're going to ask, "when would it be used without an aggregate"?
    - The schema construction that must be in existence for you to use this would have to be extremely poor, so poorly constructed that any results would likely lead to a confusing or misleading results table.
    - We will talk about the ORDER BY clause shortly!

# Grouping



- If any aggregation is used, then each element of the SELECT list must be:
  - 1 aggregated; or
  - 2 an attribute on the GROUP BY list.
- The name of the attribute used in GROUP BY does not have to be listed in the SELECT clause.
  - However, it must be a column from one of the tables in the FROM clause.



## Grouping

e.g. let's say you want to know what department numbers each employee you have belongs to.

```
SELECT COUNT(*)
AS Department_Count
FROM employee
GROUP BY empDeptNum;

Department_Count
-----
1
3
4
```

For simplicity, remember the following:

- 1 if you column names and aggregate functions in SELECT, then you must have a GROUP BY clause; and
- 2 the column names in SELECT must match the column names listed in GROUP BY.

# Grouping with WHERE



UNIVERSITY OF  
**TORONTO**  
MISSISSAUGA

- The WHERE clause eliminates rows prior to being grouped.
- Why is this important?
  - It cuts down on unnecessary computation time.
  - Less computations == less money and resources spent.
- Let's look at a query that produces the average number of hours an employee works per week, where the employee's SIN is greater than 999-500-000.

# Grouping with WHERE



Let's look at a query that produces the average number of hours an employee works per week, where the employee's SIN is greater than 999-500-000.

```
SELECT empSIN AS SIN,  
AVG(workHours) AS Average_Hours_Worked  
FROM employee  
WHERE empSIN > 999500000  
GROUP BY empSIN;
```

SIN	Average_Hours_Worked
999666666	39.5
999887777	40.5
999888888	41.5

# Grouping with HAVING



- The HAVING clause is used for aggregate functions. Much like the WHERE clause is used for column names and expressions.
- HAVING and WHERE in technicality do the same thing, just at different times.

i.e. both filter rows from inclusion in a result table based on some condition.

- 1 The WHERE clause filters rows **BEFORE** the grouping action.
- 2 The HAVING clause filters rows **AFTER** the grouping action.

# Grouping with HAVING



Let's look at the ordering example, this time using GROUP BY and HAVING.

```
SELECT empDeptNum AS Department,  
AVG(empSalary) as Average_Salary  
FROM employee  
GROUP BY empDeptNum  
HAVING AVG(empSalary) < 90000;
```

Department	Average_Salary
1	\$79,575
7	\$83,400

# Ordering



- The ORDER BY clause allows you to specify how rows in a result table are sorted.
  - DB2's default is ascending order (smallest to largest).

```
SELECT empDeptNum AS Department,  
AVG(empSalary) AS Average_Salary  
FROM employee  
GROUP BY empDeptNum  
ORDER BY AVG(empSalary) ;
```

Department	Average_Salary
1	\$79,575
7	\$83,400
3	\$93,660

# Grouping with WHERE and HAVING



Let's look at the combination of the two clauses.

```
SELECT empDeptNum AS Department,  
       AVG(empSalary) as Average_Salary  
FROM employee  
WHERE empDeptNum <> 1  
GROUP BY empDeptNum  
HAVING AVG(empSalary) < 90000;
```

Department	Average_Salary
7	\$83,400

# SQL step-by-step

```
SELECT empDeptNum AS Department,  
AVG(empSalary) as  
Average_Salary  
FROM employee  
WHERE empDeptNum <> 1  
GROUP BY empDeptNum  
HAVING AVG(empSalary) < 90000;
```

- Conceptually, SQL performs the following steps for the query:
  - 1 The WHERE clause filters the empDeptNum not equal to 1.
  - 2 The GROUP BY clause collects the remaining rows into one or more groups for each unique empDeptNum.
  - 3 The aggregate function calculates the average salary for each empDeptNum grouping.
  - 4 The HAVING clause filters the rows from the result table which fail to meet the condition (i.e. the average salary is less than \$90,000).



# Any Questions?

---

- Do you have any questions?
- If you have any content that you would like to be added in a Tutorial, please let me know by Friday!
  - Email request to me: [nafiz.hossain@utoronto.ca](mailto:nafiz.hossain@utoronto.ca)