Week 6: Lecture Recap

CSC 343

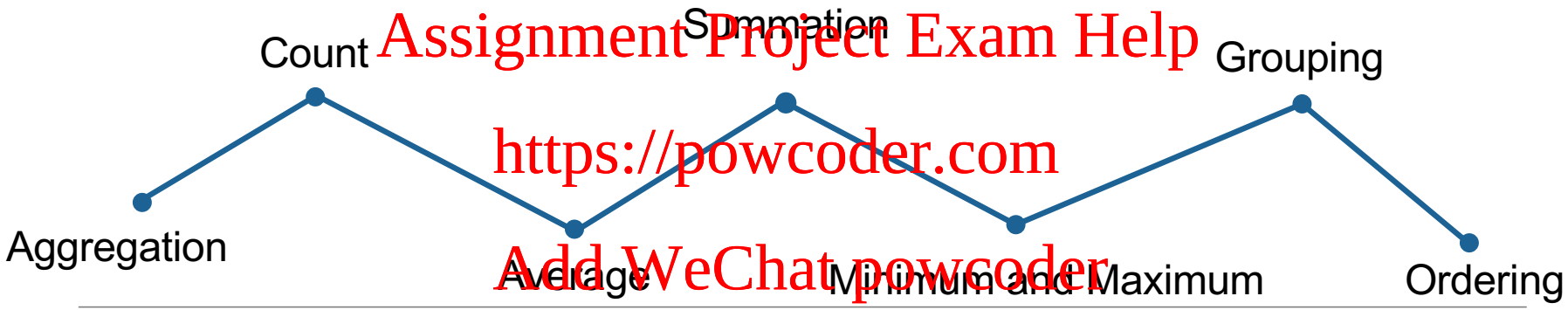
Winter 2021

MICHAEL LIUT (MICHAEL.LIUT@UTORONTO.CA)
ILIR DEMA (ILIR.DEMA@UTORONTO.CA)

DEPARTMENT OF MATHEMATICAL AND COMPUTATIONAL SCIENCES
UNIVERSITY OF TORONTO MISSISSAUGA

UNIVERSITY OF TORONTO MISSISSAUGA

Table of Contents



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

UNIVERSITY OF TORONTO MISSISSAUGA

Function Syntax	Function Usage
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.

https://www.postgresql.org/docs/13/functions-aggregate.html

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

UNIVERSITY OF TORONTO MISSISSAUGA

- Count() returns a single scalar value.
- 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

UNIVERSITY OF TORONTO MISSISSAUGA

- 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

UNIVERSITY OF TORONTO MISSISSAUGA

- 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

UNIVERSITY OF TORONTO MISSISSAUGA

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


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

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

Summation

UNIVERSITY OF TORONTO MISSISSAUGA

- 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

UNIVERSITY OF TORONTO MISSISSAUGA

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

UNIVERSITY OF TORONTO MISSISSAUGA

- 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

UNIVERSITY OF TORONTO MISSISSAUGA

- 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

UNIVERSITY OF TORONTO MISSISSAUGA

- 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 have 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

UNIVERSITY OF TORONTO MISSISSAUGA

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

UNIVERSITY OF TORONTO MISSISSAUGA

- 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

UNIVERSITY OF TORONTO MISSISSAUGA

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

UNIVERSITY OF TORONTO MISSISSAUGA

- The ORDER BY clause allows you to specify how rows in a result table are sorted.
  - PostgreSQL'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

UNIVERSITY OF TORONTO MISSISSAUGA

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).

Task (Week 4 Extension!)

Write a single statement to do the following:

- Return how many aliases citizen 4 has and name the column "Strange Nicknames"
- Return the average weight of Superheroes who are older than 50.
- Return the most vicious (in terms of people killed) Villain.
- For all citizens with 2 or more nicknames, show the their cid, and how many nicknames they have.
- Show the **names** of all the ExtraordinaryCitizen next to their total PowerGrid points

Note: You will have to have a clean/empty version of the tables from Lecture's Week 4 to do this. Use the sample solution DDL posted on the class site to create the tables for consistency in the solution.

UNIVERSITY OF TORONTO MISSISSAUGA

Hmm..

- Show the **names** of the ExtraordinaryCitizen with the most PowerGrid points
- I can't do this, the DDL design doesn't allow me to do it!
- Let's consider the concept of joins.

UNIVERSITY OF TORONTO MISSISSAUGA

Cross product

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp Sci	4

course_id	prereq_id
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

course_id	title	dept_name	credits	course_id	prereq_id
BIO-301	Genetics	Biology	4	BIO-301	BIO-101
BIO-301	Genetics	Biology	4	CS-190	CS-101
BIO-301	Genetics	Biology	4	CS-347	CS-101
CS-190	Game Design	Comp Sci	4	BIO-301	BIO-101
CS-190	Game Design	Comp Sci	4	CS-190	CS-101
CS-190	Game Design	Comp Sci	4	CS-347	CS-101

Note: Taken f

Left outer join

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp Sci	4
CS-315	Robotics	Comp Sci	3

course_id	prereq_id
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp Sci	4	CS-101
CS-315	Robotics	Comp Sci	3	NULL

Note: Taken from lecture slides

UNIVERSITY OF TORONTO MISSISSAUGA

Right outer join

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp Sci	4
CS-315	Robotics	Comp Sci	3

course_id	prereq_id
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

course_id	prereq_id	title	dept_name	credits
BIO-301	BIO-101	Genetics	Biology	4
CS-190	CS-101	Game Design	Comp Sci	4
CS-347	CS-101	NULL	NULL	NULL

Full outer join

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp Sci	4
CS-315	Robotics	Comp Sci	3

course_id	prereq_id
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp Sci	4	CS-101
CS-315	Robotics	Comp Sci	3	NULL
CS-347	NULL	NULL	NULL	CS-101

Note: Taken from lecture slides

UNIVERSITY OF TORONTO MISSISSAUGA

Inner join

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp Sci	4
CS-315	Robotics	Comp Sci	3

course_id	prereq_id
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp Sci	4	CS-101

Note: Taken from lecture slides

UNIVERSITY OF TORONTO MISSISSAUGA

Let's try it again

- Show the **names** of all the ExtraordinaryCitizen next to their total PowerGrid points

Example

```
SELECT cid, Intelligence + EnergyProjection + Durability + Strength
       + Speed + FightingSkills AS SumPoints
FROM PowerGrid;
```


That gets us the cid and the point sum for each citizen, but how do we get the Names of each of those citizens?

Example

```
SELECT Name, SumPoints
    FROM ExtraordinaryCitizen EC INNER JOIN
    (SELECT
        cid,
        Intelligence + EnergyProjection + Durability + Strength
            + Speed + FightingSkills AS SumPoints
    FROM PowerGrid) SumPgTable
    ON SumPgTable.cid = EC.cid;
```

UNIVERSITY OF TORONTO MISSISSAUGA

Additional Trigger Exercise

Create a trigger that will automatically check when inserting a new villain. If this villain is in prison and kills 0 people with an age older than 100, release the new villain from the prison (i.e., set imprisonment to false).

Solution

```
CREATE OR REPLACE FUNCTION show_mercy()
   RETURNS TRIGGER AS $show_mercy_trigger$
   BEGIN
     IF (NEW.Imprisonment = true) AND (NEW.PeopleKilled = 0) AND (NEW.vid IN (SELECT cid
FROM ExtraordinaryCitizen WHERE age > 100))
       THEN NEW.Imprisonment = false;
     END IF;
     RETURN NEW;
   END;
$show_mercy_trigger$ LANGUAGE PLPGSQL;

CREATE TRIGGER show_mercy_trigger BEFORE INSERT ON villain
FOR EACH ROW EXECUTE PROCEDURE show_mercy();
```