Aggregates

.NET Cohort

Coding Bootcamp

SOFTWARE GUILD

Lesson Goals

- Learn the common keywords for rolling up detail rows into aggregate data
- Learn how to filter aggregated results

Aggregate?

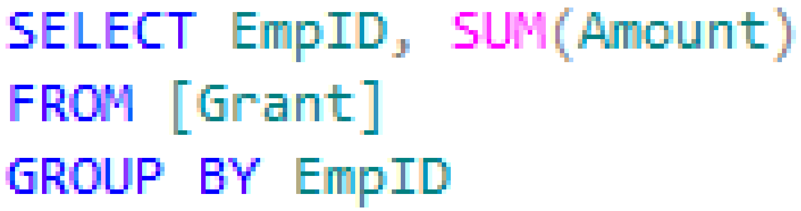
- Aggregating is a big scary word, but it really means “rolling up data into totals.”
- When we have millions of records and details, our management team is interested in these rollups and “big picture” items:
  - totals, subtotals; slicing data by key columns like date ranges, location, customer types, etc.
- Aggregate functions take raw details and turn them into actionable summaries.

Group By

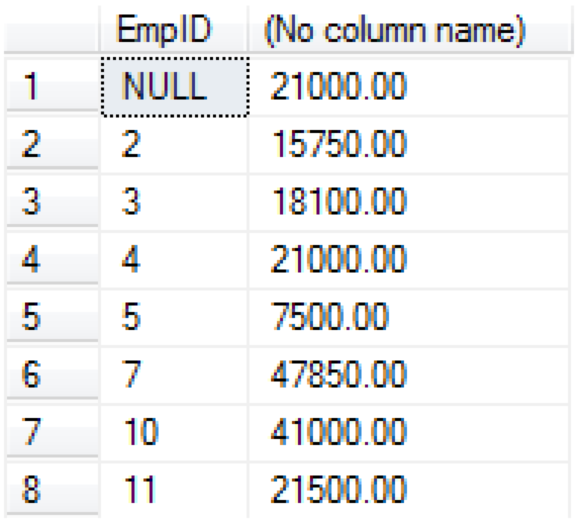
- The Group By statement does exactly what it says: it groups results of a query by <u>one or more</u> columns so that you can aggregate them.
- The most common aggregates are:
  - Count()
  - Sum()
  - Min()
  - Max()
  - AVG()
- Learning to use Group By effectively is critical to being a successful database developer.

Example: Group By with Sum()

Let's say we want to get the sum of all grant amounts by employee id:



```
SELECT EmpID, SUM(Amount)
FROM [Grant]
GROUP BY EmpID
```

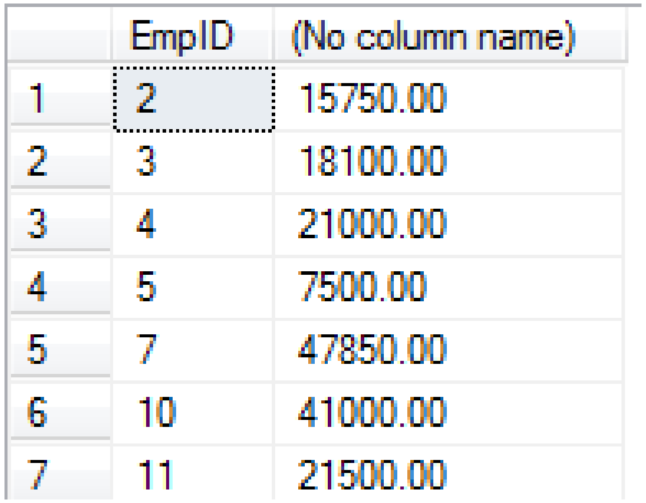


	EmpID	(No column name)
1	NULL	21000.00
2	2	15750.00
3	3	18100.00
4	4	21000.00
5	5	7500.00
6	7	47850.00
7	10	41000.00
8	11	21500.00

Oops, We Want Grants for Employees

Notice the null record: some grants aren’t assigned to employees. Let’s filter those out:

```
SELECT EmpID, SUM(Amount)
FROM [Grant]
WHERE EmpID IS NOT NULL
GROUP BY EmpID
```

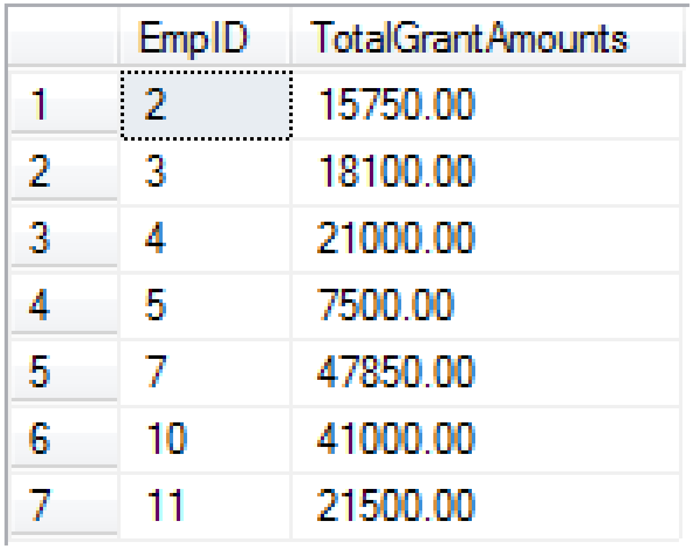


	EmpID	(No column name)
1	2	15750.00
2	3	18100.00
3	4	21000.00
4	5	7500.00
5	7	47850.00
6	10	41000.00
7	11	21500.00

Aggregates are Expressions

Notice the lack of a column name, let’s fix that too:

```sql
SELECT EmpID, SUM(Amount) AS TotalGrantAmounts
FROM [Grant]
WHERE EmpID IS NOT NULL
GROUP BY EmpID
```

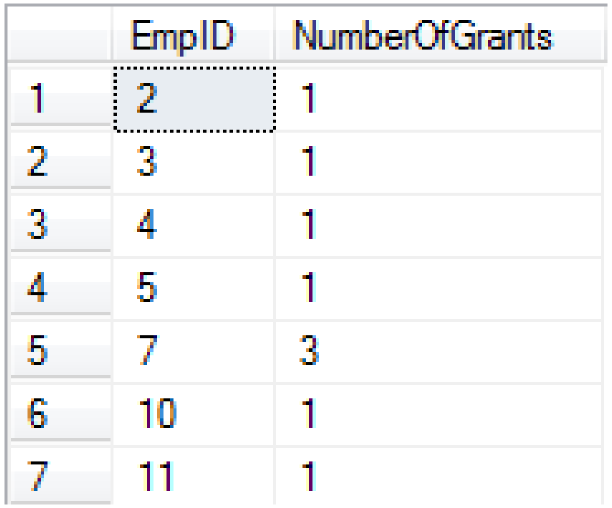


	EmpID	TotalGrantAmounts
1	2	15750.00
2	3	18100.00
3	4	21000.00
4	5	7500.00
5	7	47850.00
6	10	41000.00
7	11	21500.00

How About Count()?

Let's find the number of grants per employee:

```
SELECT EmpID, Count(Amount) AS NumberOfGrants
FROM [Grant]
WHERE EmpID IS NOT NULL
GROUP BY EmpID
```

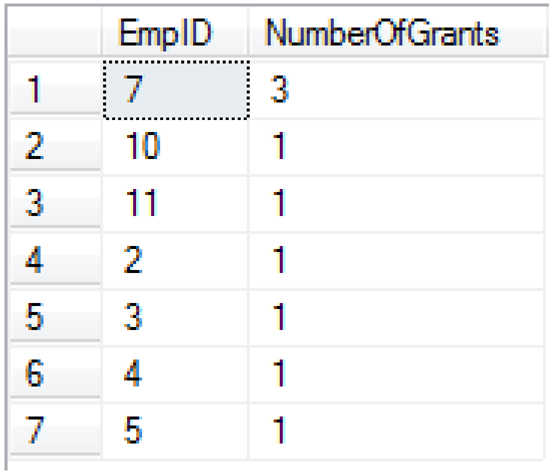


	EmpID	NumberOfGrants
1	2	1
2	3	1
3	4	1
4	5	1
5	7	3
6	10	1
7	11	1

Can We Order It?

Sure can.

```sql
SELECT EmpID, Count(Amount) AS NumberOfGrants
FROM [Grant]
WHERE EmpID IS NOT NULL
GROUP BY EmpID
ORDER BY NumberOfGrants DESC
```

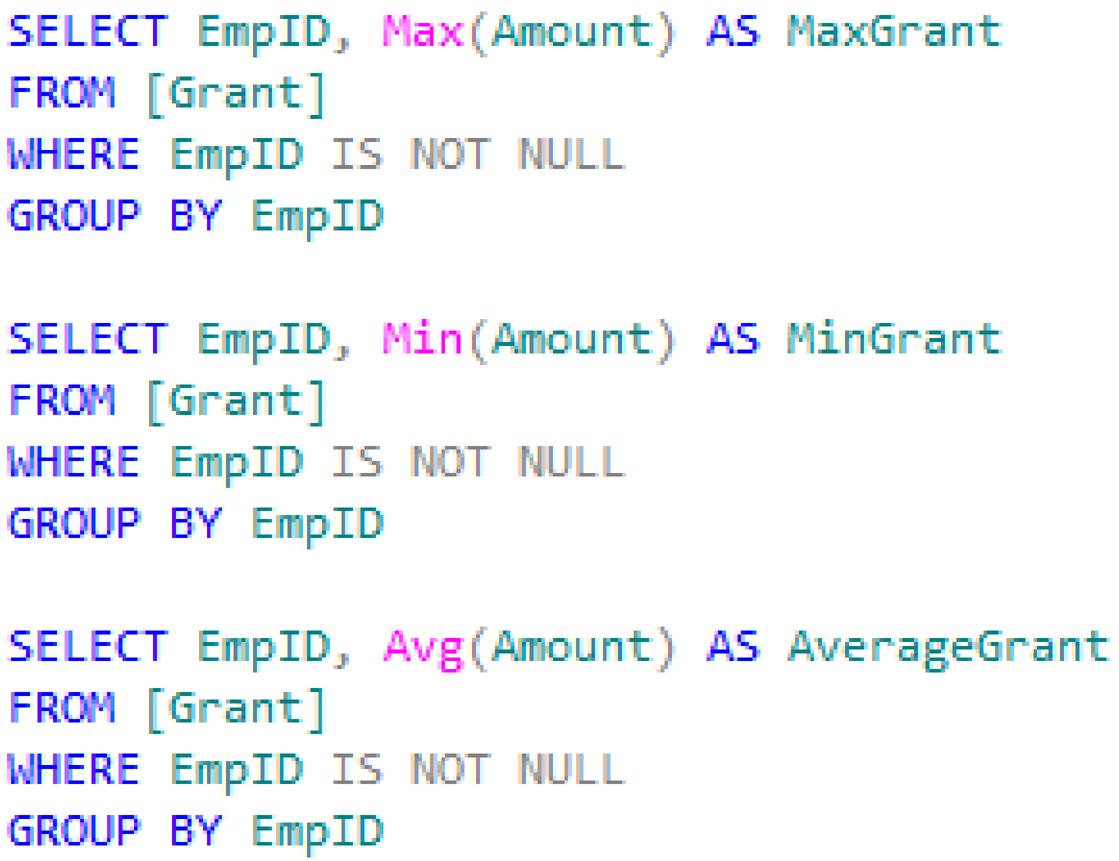


	EmpID	NumberOfGrants
1	7	3
2	10	1
3	11	1
4	2	1
5	3	1
6	4	1
7	5	1

Your Turn

- Find the Max, Min, and Average Grant amount by Employee ID

Hope you came up with this:



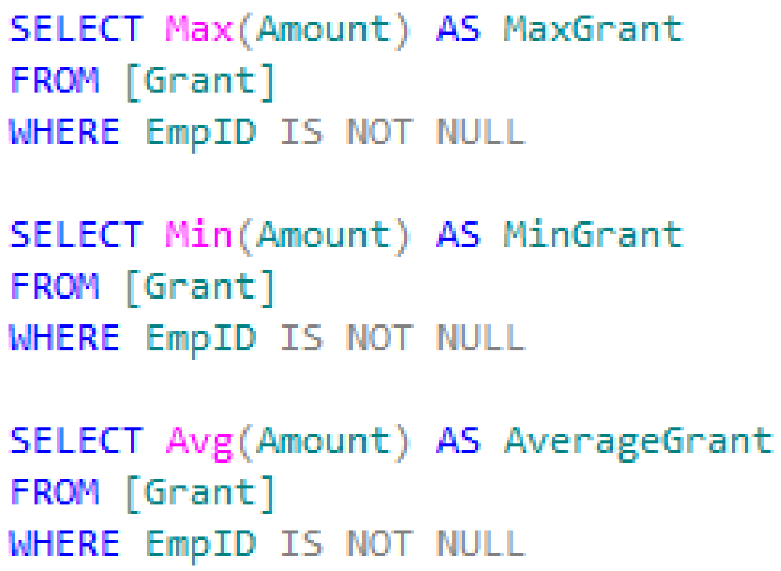
```sql
SELECT EmpID, Max(Amount) AS MaxGrant
FROM [Grant]
WHERE EmpID IS NOT NULL
GROUP BY EmpID

SELECT EmpID, Min(Amount) AS MinGrant
FROM [Grant]
WHERE EmpID IS NOT NULL
GROUP BY EmpID

SELECT EmpID, Avg(Amount) AS AverageGrant
FROM [Grant]
WHERE EmpID IS NOT NULL
GROUP BY EmpID
```

If You Want the Whole Table…

Then you don’t need Group By. For the max, min, and average of all employee grants:



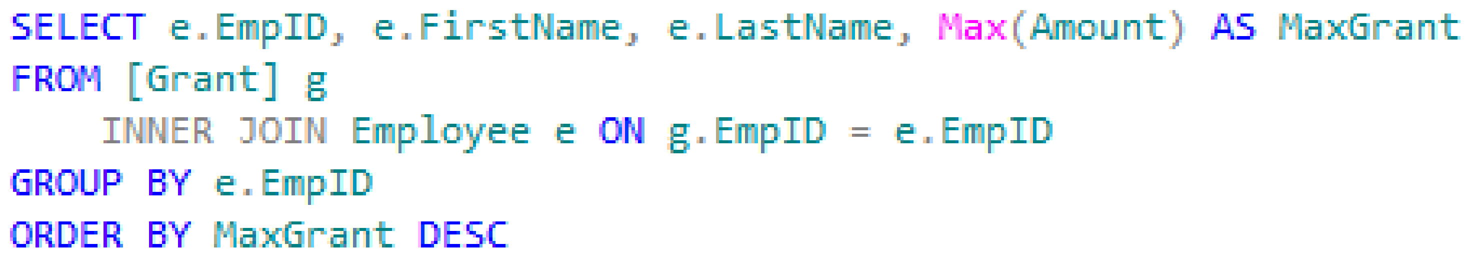
```
SELECT Max(Amount) AS MaxGrant
FROM [Grant]
WHERE EmpID IS NOT NULL

SELECT Min(Amount) AS MinGrant
FROM [Grant]
WHERE EmpID IS NOT NULL

SELECT Avg(Amount) AS AverageGrant
FROM [Grant]
WHERE EmpID IS NOT NULL
```

What About Joins?

Try this:

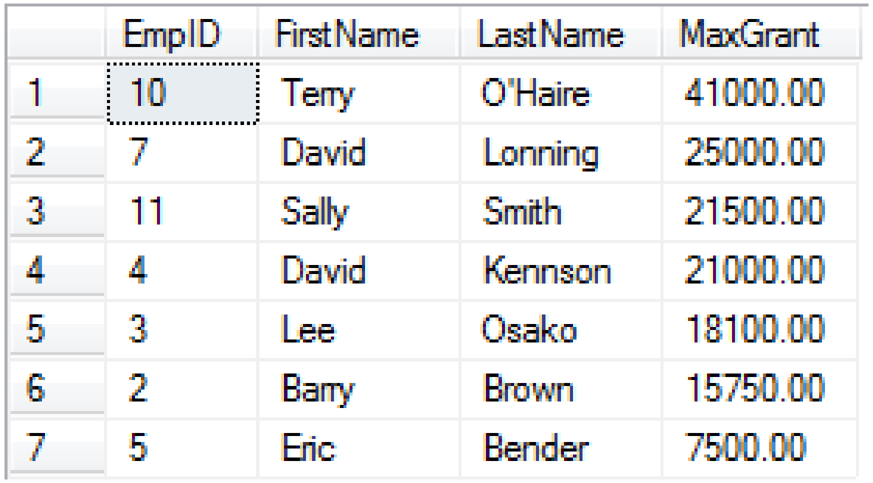


```
SELECT e.EmpID, e.FirstName, e.LastName, Max(Amount) AS MaxGrant
FROM [Grant] g
    INNER JOIN Employee e ON g.EmpID = e.EmpID
GROUP BY e.EmpID
ORDER BY MaxGrant DESC
```

What Went Wrong?

When you use an aggregate, you <u>must</u> list all non-aggregate columns in the Group By. Let's fix it:

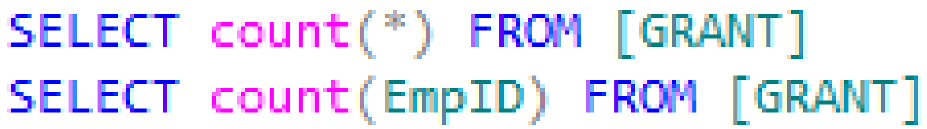
```
SELECT e.EmpID, FirstName, LastName, Max(Amount) AS MaxGrant
FROM [Grant] g
    INNER JOIN Employee e ON g.EmpID = e.EmpID
GROUP BY e.EmpID, FirstName, LastName
ORDER BY MaxGrant DESC
```



	EmpID	FirstName	LastName	MaxGrant
1	10	Terry	O'Haire	41000.00
2	7	David	Lonning	25000.00
3	11	Sally	Smith	21500.00
4	4	David	Kennson	21000.00
5	3	Lee	Osako	18100.00
6	2	Barry	Brown	15750.00
7	5	Eric	Bender	7500.00

A word about count()

- If you count(columnName), it will ignore null values in the column
- If you count(*), it will count rows regardless of null values.



```
SELECT count(*) FROM [GRANT]
SELECT count(EmpID) FROM [GRANT]
```

A word about Order By

When using aggregates, you can only order by fields in the SELECT statement. This is different than the normal behavior where you can order by hidden fields.

Lab Exercises! (Northwind)

1. Find the max unit price for each product by category.
2. We want to get some customer data. Create lists of customers with the following attributes:
    1. Most orders submitted
    2. Highest lifetime order total amount (hint: the Order table doesn't have the total, you will need to join to order details and calculate it)

Filtering Aggregates

Try this:

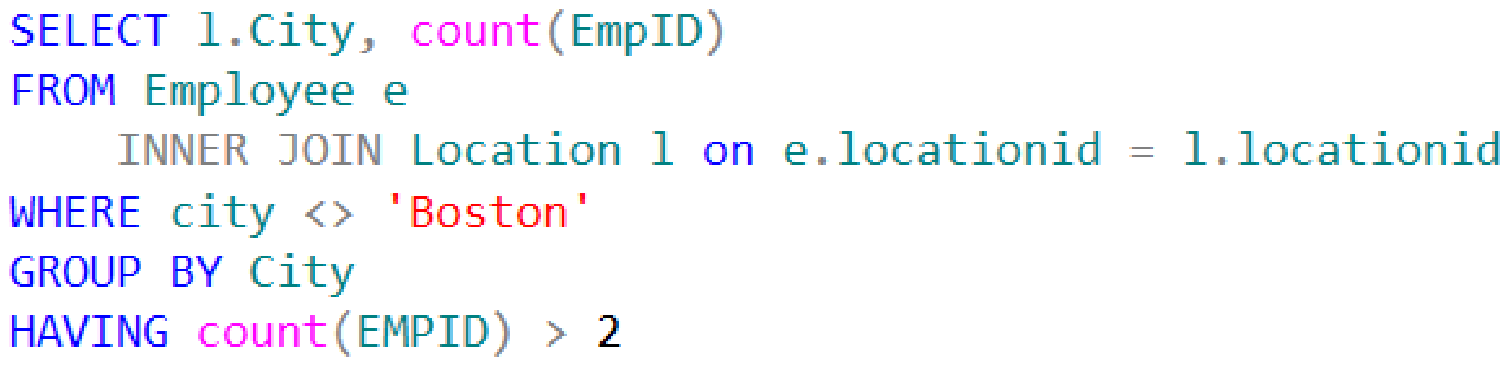
```
SELECT l.City, count(EmpID)
FROM Employee e
    INNER JOIN Location l on e.locationid = l.locationid
WHERE count(EmpID) > 3
GROUP BY City
```

It doesn’t work because WHERE executes before Group By.

The HAVING Statement

Put an aggregate in a HAVING statement to filter on an aggregate column (or more than one column).

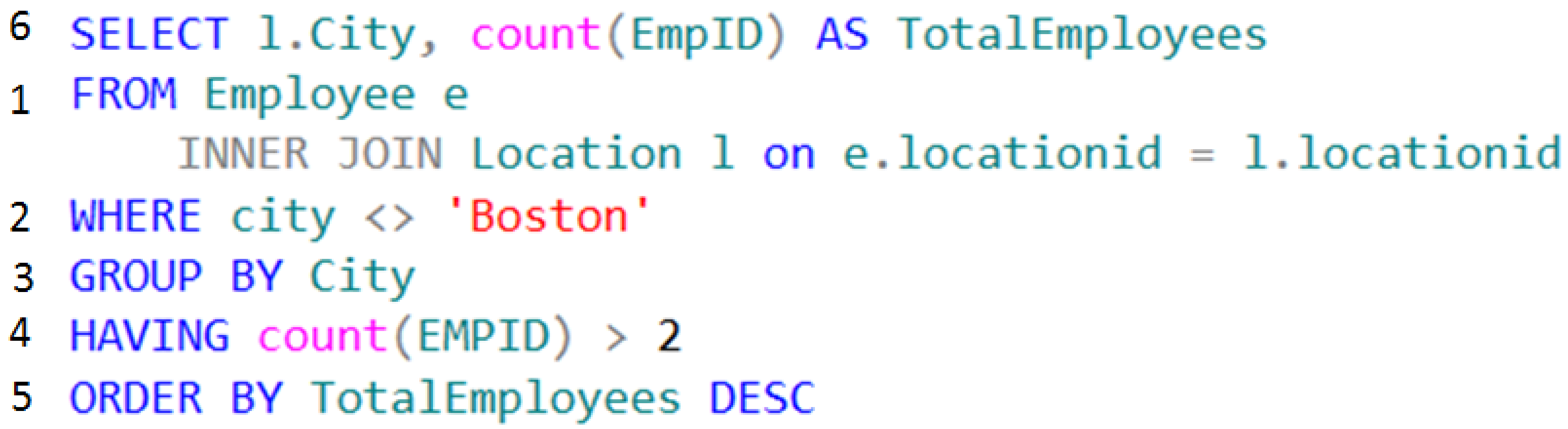
Think of it as a WHERE clause for your aggregate.



```sql
SELECT l.City, count(EmpID)
FROM Employee e
    INNER JOIN Location l on e.locationid = l.locationid
WHERE city <> 'Boston'
GROUP BY City
HAVING count(EMPID) > 2
```

Order of Execution

Where doesn't work for aggregates because Where affects results before Group By is applied. Here is the processing order of a SQL Query:



```
6  SELECT l.City, count(EmpID) AS TotalEmployees
1  FROM Employee e
       INNER JOIN Location l on e.locationid = l.locationid
2  WHERE city <> 'Boston'
3  GROUP BY City
4  HAVING count(EMPID) > 2
5  ORDER BY TotalEmployees DESC
```

Lab Exercises (Northwind)

1. Find customers with more than 10 orders.
2. Find employees that have more than 100 orders. Only the name and count should be outputted, so it should look like this:

```
EmployeeName                OrderCount
LastName, FirstName       #####
```

Fin

- Next up: Filtering Aggregates