SQL – Aggregates

CS 4750
Database Systems

[A. Silberschatz, H. F. Korth, S. Sudarshan, Database System Concepts, Ch.3.7 and Ch. 5.5]
[C.M. Ricardo, S.D. Urban, Databases Illuminated, Ch. 5.4]

Aggregation Functions

Calculate a value across an entire set or across groups of rows within the set

SQL uses five aggregation operators:

- SUM – produces the sum of a column with numerical values
- AVG – produces the average of a column with numerical values
- MIN – applied to a column with numerical values, produces the smallest value
- MAX – applied to a column with numerical values, produces the largest value
- COUNT – produces the number of (not necessarily distinct) values in a column

Example: SUM

Given the `loan` schema

```
loan(loan_number, branch_name, amount)
```

loan_number	branch_name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

The sum of the amounts of all loans is expressed by

```
SELECT SUM(amount)
FROM loan
```

SUM(amount)
8700

```
SELECT SUM(amount) AS Total
FROM loan
```

Total
8700

Example: AVG

Given the `loan` schema

```
loan(loan_number, branch_name, amount)
```

loan_number	branch_name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

The average of the amounts of all loans is expressed by

```
SELECT AVG(amount)
FROM loan
```

AVG(amount)
1242.857142857143

Example: MIN

Given the `loan` schema

```
loan(loan_number, branch_name, amount)
```

loan_number	branch_name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

The smallest amount of loans is expressed by

```
SELECT MIN(amount)
FROM loan
```

min(amount)
500

Example: MAX

Given the `loan` schema

```
loan(loan_number, branch_name, amount)
```

loan_number	branch_name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

The largest amount of loans is expressed by

```
SELECT MAX(amount)
FROM loan
```

max(amount)
2000

Example: COUNT

Given the `loan` schema

```
loan(loan_number, branch_name, amount)
```

loan_number	branch_name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

Count the number of tuples in the `loan` table

```
SELECT count(*)
FROM loan
```

Count rows

count(*)
7

```
SELECT count(loan_number)
FROM loan
```

Count values of a specified column

Count(loan_number)
7

Example: COUNT .. DISTINCT

Given the `loan` schema

```
loan(loan_number, branch_name, amount)
```

loan_number	branch_name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

Count the number of values in the `branch_name` column

```
SELECT count(branch_name)
FROM loan
```

```
SELECT count(DISTINCT branch_name)
FROM loan
```

count(branch_name)
7

count(distinct branch_name)
5

Aggregation: Order of Actions

```
SELECT select_list
FROM table_source
[WHERE search_condition]
[GROUP BY group_by_expression]
[HAVING search_condition]
[ORDER BY order_expression [ASC | DESC] ]
```

Order matter

1. The FROM clause generates the data set
2. The WHERE clause filters the data set generated by the FROM clause
3. The GROUP BY clause aggregates the data set that was filtered by the WHERE clause (note: GROUP BY does not sort the result set)
4. The HAVING clause filters the data set that was aggregated by the GROUP BY clause
5. The SELECT clause transforms the filtered aggregated data set
6. The ORDER BY clause sorts the transformed data set

Grouping Requirement

Several DBMS requires that the columns appear in the SELECT clause that are not used in an aggregation function must appear in the GROUP BY clause

✓

```
SELECT column_A, column_B, some_aggregation_function
FROM table_source
GROUP BY column_A, column_B
```

✗

```
SELECT column_A, column_B, some_aggregation_function
FROM table_source
GROUP BY column_B
```

Example: SUM with GROUP BY

Given the `loan` schema

`loan(loan_number, branch_name, amount)`

loan_number	branch_name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

The sum of the amounts of all loans for each branch is expressed by

```
SELECT branch_name, SUM(amount)
FROM loan
GROUP BY branch_name;
```

branch_name	SUM(amount)
Downtown	2500
Mianus	500
Perryridge	2800
Redwood	2000
Round Hill	900

Let’s Try (1)

- Open your MySQL terminal (use phpMyAdmin or GCP shell)
- Use the bank database from the Basic SQL activity
  - Or run the script to set up your database http://www.cs.virginia.edu/~up3f/cs4750/inclass/alldbs.sql
- Consider table: `account`
- Write SQL statement to solve the following problem

Find the total amount each branch has in accounts

```
SELECT branch_name, SUM(balance) AS Total
FROM account
GROUP BY branch_name;
```

Let’s Try (2)

- Open your MySQL terminal (use phpMyAdmin or GCP shell)
- Use the bank database from the Basic SQL activity
  - Or run the script to set up your database http://www.cs.virginia.edu/~up3f/cs4750/inclass/alldbs.sql
- Consider table: `account`
- Write SQL statement to solve the following problem

Find the average amount each branch has in accounts

```
SELECT branch_name, AVG(balance) AS average
FROM account
GROUP BY branch_name;
```

Grouping, Aggregation, and Null

- The value NULL is ignored in any aggregation
  - Not contribute to a sum, average, or count of an attribute
  - Cannot be the minimum or maximum in its column
- Null is treated as an ordinary value when forming groups
  - Can have a group with NULL attribute(s)
- When performing any aggregation except count over an empty bag of values, the result is NULL
  - The count of an empty bag is 0

HAVING Clauses

- An aggregation in a HAVING clause applies only to the tuples of the group being tested – filter groups
- Any attributes of relations in the FROM clause may be aggregated in the HAVING clause
- But only those attributes that are in the GROUP BY list may appear unaggregated in the having clause

The sum of the amounts of all loans for each branch that has more than one loan is expressed by

```
SELECT branch_name, sum(amount)
FROM loan
GROUP BY branch_name
HAVING COUNT(branch_name) > 1;
```

branch_name	SUM(amount)
Downtown	2500
Perryridge	2800

Example 1

List the number of customers in each country. Only include countries with more than 10 customers

```
SELECT count(id), country
FROM Customer
GROUP BY country
HAVING COUNT(id) > 10;
```

count(id)	Country
11	France
11	Germany
13	USA

Example 2

List the number of customers in each country, except USA, sorted high to low. Only include countries with 9 or more customers

```
SELECT COUNT(id), country
FROM Customer
WHERE country <> “USA”
GROUP BY country
HAVING COUNT(id) >= 9
ORDER BY COUNT(id) DESC;
```

count(id)	Country
11	France
11	Germany
9	Brazil

Final Notes about Aggregation

#1: Keep the GROUP BY clause small and precise

- Several DBMSs require that all non-aggregated columns must be in the GROUP BY clause
- Excessive columns in GROUP BY can negatively impact the query's performance; make the query hard to read, understand, rewrite
- For queries that need both aggregations and details, do all aggregations in subqueries first, then join those to the tables to retrieve the details

Final Notes about Aggregation

#2: COUNT(*) and COUNT(<*column_name*>) are different

- COUNT(*) – count all rows, including ones with null values
- COUNT(<*column_name*>) – count only the rows where the column value is not NULL
- Sometimes, dividing a query into subqueies can be more efficient than using a GROUP BY (more about subqueries next week)

Final Notes about Aggregation

#3: Use DISTINCT to get distinct counts

- COUNT(*) – returns the number of rows in a group, including NULL value and duplicates
- COUNT(<*column_name*>) – returns the number of rows where the column value is not NULL
- COUNT(DISTINCT <*column_name*>) – returns the number of rows with unique, non-null values of the column

Wrap-Up

- Aggregation functions
- Order of actions matter when applying aggregation
- Aggregation helps make decisions and succinctly convey information

What's next?

- SQL – Joins
- Combine techniques (aggregates and joins) to solve complex questions