Aggregations in SQL

Enrico Franconi

franconi@inf.unibz.it
http://www.inf.unibz.it/~franconi

Faculty of Computer Science, Free University of Bozen-Bolzano
Aggregate Functions in SQL

Aggregation is an operation that computes a single value from all the values of an attribute.

SQL provides five functions that apply to an attribute of a relation and produce some aggregation of that column.

- **SUM**: Computes the sum of values in a attribute.
- **AVG**: Computes the average of values in a attribute.
- **MIN**: Computes the least value in a attribute.
- **MAX**: Computes the greatest value in a attribute.
- **COUNT**: Computes the number of values in a attribute (including duplicates unless they are explicitly eliminated with **DISTINCT**).
Example Database

DEPARTMENT (DNUMBER, DNAME)

EMPLOYEE (ENUMBER, NAME, SALARY, DNO)

DNO foreign key references DEPARTMENT

PROJECT (PNO, PNAME)

WORKS_ON (ENUMBER, PNUMBER)

ENUMBER foreign key references EMPLOYEE

PNUMBER foreign key references PROJECT

<table>
<thead>
<tr>
<th>ENUMBER</th>
<th>NAME</th>
<th>SALARY</th>
<th>DNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>“John”</td>
<td>45,000</td>
<td>5</td>
</tr>
<tr>
<td>id2</td>
<td>“Mary”</td>
<td>50,000</td>
<td>4</td>
</tr>
<tr>
<td>id3</td>
<td>“Nick”</td>
<td>42,000</td>
<td>4</td>
</tr>
<tr>
<td>id4</td>
<td>“Paul”</td>
<td>43,000</td>
<td>5</td>
</tr>
<tr>
<td>id5</td>
<td>“Laura”</td>
<td>55,000</td>
<td>1</td>
</tr>
<tr>
<td>id6</td>
<td>“Andrea”</td>
<td>31,000</td>
<td>5</td>
</tr>
<tr>
<td>id7</td>
<td>“Brian”</td>
<td>25,000</td>
<td>4</td>
</tr>
<tr>
<td>id8</td>
<td>“Alon”</td>
<td>26,000</td>
<td>5</td>
</tr>
</tbody>
</table>
Aggregate Functions in SQL (cont.)

**Query:** Find the sum of the salaries of all employees, the maximum salary, the minimum salary and the average salary.

This query can be expressed in SQL as follows:

```
SELECT SUM(SALARY), MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM EMPLOYEE;
```

This query will return the following relation:

<table>
<thead>
<tr>
<th>SUM(SALARY)</th>
<th>MAX(SALARY)</th>
<th>MIN(SALARY)</th>
<th>AVG(SALARY)</th>
</tr>
</thead>
<tbody>
<tr>
<td>317,000</td>
<td>55,000</td>
<td>25,000</td>
<td>39,625</td>
</tr>
</tbody>
</table>
Aggregate Functions in SQL (cont.)

Query: Find the sum as well as the maximum, minimum, and average salary of all employees working in the “Research” department.

This query can be expressed in SQL as follows:

```
SELECT SUM(SALARY), MAX(SALARY),
       MIN(SALARY), AVG(SALARY)
FROM    EMPLOYEE, DEPARTMENT
WHERE   DNO=DNUMBER AND DNAME='Research';
```
**Aggregate Functions in SQL (cont.)**

**Query:** Retrieve the total number of employees in the “Research” department.

This query can be expressed in SQL as follows:

```sql
SELECT COUNT(*)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO=DNUMBER AND DNAME='Research';
```

**Warning:** Only the aggregate function `COUNT` is allowed to apply to whole tuples. It does not make sense to apply any other aggregate functions to more than a single attribute.
**Aggregate Functions in SQL (cont.)**

**Query:** Count the number of distinct salary values in the database.

This query can be expressed in SQL as follows:

```sql
SELECT COUNT(DISTINCT SALARY)
FROM EMPLOYEE;
```

What would the effect of `COUNT(SALARY)` in the above query be?
The GROUP BY Clause

If we want to apply an aggregate function to subgroups of tuples then we can use the GROUP BY clause.

Each group corresponds to the value of one or more attributes.

The syntax of the GROUP BY clause is

```
GROUP BY <grouping attributes>
```

where `<grouping attributes>` specifies a list of attribute names.

**Note:** The SELECT clause must contain exactly the grouping attributes in addition with a possible aggregation function.
Query: For each department, retrieve the department number, the number of employees in the department and their average salary.

This query can be expressed in SQL as follows:

```
SELECT DNO, COUNT(*), AVG(SALARY)
FROM EMPLOYEE
GROUP BY DNO;
```

The result of this query will be:

<table>
<thead>
<tr>
<th>DNO</th>
<th>COUNT(*)</th>
<th>AVG(SALARY)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4</td>
<td>36,250</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>39,000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>55,000</td>
</tr>
</tbody>
</table>
The GROUP BY Clause (cont.)

The following query shows how to use a GROUP BY in conjunction with JOIN.

**Query:** For each project, retrieve the project number, the project name and the number of employees who work on the project.

This query can be expressed in SQL as follows:

```
SELECT    PNUMBER, PNAME, COUNT(*)
FROM       PROJECT, WORKS_ON
WHERE      PNUMBER=PNO
GROUP BY   PNUMBER, PNAME
```

The grouping and aggregation are applied after joining the relations.
The result of this query is:

<table>
<thead>
<tr>
<th>PNUMBER</th>
<th>PNAME</th>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ProductX</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>ProductY</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>ProductZ</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>Computerization</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>Reorganization</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>Newbenefits</td>
<td>3</td>
</tr>
</tbody>
</table>
The GROUP BY Clause (cont.)

It is possible to use a GROUP BY clause in conjunction with a SELECT clause that does not use any aggregation function:

```
SELECT   SALARY
FROM     EMPLOYEE
GROUP BY SALARY
```

Has the same effect as:

```
SELECT DISTINCT SALARY
FROM    EMPLOYEE
```
The HAVING Clause

Sometimes we want to choose groups of tuples based on some aggregate property of the group itself. In this case we have to use the **HAVING** clause together with the GROUP BY clause.

The syntax of the **HAVING** clause is:

```
HAVING  < condition >
```

where `< condition >` is a Boolean expression formed by comparison conditions as in the WHERE clause.
The HAVING Clause (cont.)

**Query:** For each project on which *more than two employees work*, retrieve the project number, the project name and the number of employees who work on the project.

This query can be expressed in SQL as follows:

```sql
SELECT PNUMBER, PNAME, COUNT(*)
FROM PROJECT, WORKS_ON
WHERE PNUMBER=PNO
GROUP BY PNUMBER, PNAME
HAVING COUNT(*) > 2;
```
The result of this query is:

<table>
<thead>
<tr>
<th>PNUMBER</th>
<th>PNAME</th>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>ProductY</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>Computerization</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>Reorganization</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>Newbenefits</td>
<td>3</td>
</tr>
</tbody>
</table>
Interpreting SQL Queries

The **result** of an SQL query involving aggregate functions, GROUP BY and HAVING can be computed as follows:

1. Evaluate the relation \( R \) implied by the FROM and WHERE clauses. \( R \) is the Cartesian product of the relations specified in the FROM clause, to which the selection of the WHERE clause is applied.

2. Group the tuples of \( R \) according to the attributes in the GROUP BY clause.

3. Filter out the tuples of \( R \) not satisfying the condition of the HAVING clause to compute a new relation \( R' \).

4. Apply to \( R' \) the projections and aggregations specified in the SELECT clause to compute the final result.
The HAVING Clause (cont’d)

Be careful combining the conditions in a WHERE clause with the ones in the HAVING clause.

**Query:** For each department having more than 2 employees, retrieve the department name and the number of employees whose salary exceed £40,000.

An *incorrect* formulation of the query is:

```sql
SELECT DNAME, COUNT(*)
FROM DEPARTMENT, EMPLOYEE
WHERE DNUMBER=DNO AND SALARY > 40000
GROUP BY DNAME
HAVING COUNT(*) > 2;
```
The correct formulation of the query can be expressed in SQL as follows:

```
SELECT DNAME, COUNT(*)
FROM DEPARTMENT, EMPLOYEE
WHERE DNUMBER=DNO AND
  SALARY > 40000 AND
  DNO IN (SELECT DNO
            FROM EMPLOYEE
            GROUP BY DNO
           HAVING COUNT(*) > 2)
GROUP BY DNAME;
```
Query examples

RESTAURANT(NAME, PLACE, SEATS)

PARTY(CODE, COST, RESTNAME, OCCASION), RESTNAME foreign key references RESTAURANT

GUEST(NAME, PARTYCODE), PARTYCODE foreign key references PARTY

PRESENT(GUESTNAME, PARTYCODE, TYPE), GUESTNAME, PARTYCODE foreign key references GUEST

1. Select the names of the restaurants hosting a party with a number of guests greater than the number of seats of the restaurant.

2. Select the names of the most generous guest(s), i.e., the guest bringing the highest number of presents for a single party.

3. Select, for each party, the name of the most generous guest(s), i.e., the guest(s) bringing the highest number of presents for the party.