# CS636 - Data Warehouse

## 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

$\mathtt{DEPARTMENT}(\underline{\mathtt{DNUMBER}}, \mathtt{DNAME})$

$\mathtt{EMPLOYEE}(\underline{\mathtt{ENUMBER}}, \mathtt{NAME}, \mathtt{SALARY}, \mathtt{DNO})$

*DNO foreign key references DEPARTMENT*

$\mathtt{PROJECT}(\underline{\mathtt{PNO}}, \mathtt{PNAME})$

$\mathtt{WORKS-ON}(\underline{\mathtt{ENUMBER}}, \underline{\mathtt{PNUMBER}})$

*ENUMBER foreign key references EMPLOYEE*

*PNUMBER foreign key references PROJECT*

EMPLOYEE

| *ENUMBER* | *NAME* | *SALARY* | *DNO* |
|---|---|---|---|
| id1 | “John” | 45,000 | 5 |
| id2 | “Mary” | 50,000 | 4 |
| id3 | “Nick” | 42,000 | 4 |
| id4 | “Paul” | 43,000 | 5 |
| id5 | “Laura” | 55,000 | 1 |
| id6 | “Andrea” | 31,000 | 5 |
| id7 | “Brian” | 25,000 | 4 |
| id8 | “Alon” | 26,000 | 5 |

# 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:

| SUM(SALARY) | MAX(SALARY) | MIN(SALARY) | AVG(SALARY) |
|---|---|---|---|
| 317,000 | 55,000 | 25,000 | 39,625 |

# 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:

```
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:

```
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

$$\text{GROUP BY} \ < \textit{grouping attributes} >$$

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.

# The GROUP BY Clause (cont.)

**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:

| DNO | COUNT(*) | AVG(SALARY) |
|---|---|---|
| 5 | 4 | 36,250 |
| 4 | 3 | 39,000 |
| 1 | 1 | 55,000 |

# 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 GROUP BY Clause (cont.)

The result of this query is:

| PNUMBER | PNAME | COUNT(*) |
|---|---|---|
| 1 | ProductX | 2 |
| 2 | ProductY | 3 |
| 3 | ProductZ | 2 |
| 10 | Computerization | 3 |
| 20 | Reorganization | 3 |
| 30 | Newbenefits | 3 |

# 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:

$$\textbf{HAVING} \ < condition >$$

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:

```
SELECT     PNUMBER, PNAME, COUNT(*)
FROM       PROJECT, WORKS_ON
WHERE      PNUMBER=PNO
GROUP BY   PNUMBER, PNAME
HAVING     COUNT(*) > 2;
```

# The HAVING Clause (cont.)

The result of this query is:

| PNUMBER | PNAME | COUNT(*) |
|---|---|---|
| 2 | ProductY | 3 |
| 10 | Computerization | 3 |
| 20 | Reorganization | 3 |
| 30 | Newbenefits | 3 |

# 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:

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

# The HAVING Clause (cont’d)

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

$\mathtt{RESTAURANT}(\underline{\mathtt{NAME}}, \mathtt{PLACE}, \mathtt{SEATS})$

$\mathtt{PARTY}(\underline{\mathtt{CODE}}, \mathtt{COST}, \mathtt{RESTNAME}, \mathtt{OCCASION})$, *RESTNAME foreign key references RESTAURANT*

$\mathtt{GUEST}(\underline{\mathtt{NAME}}, \underline{\mathtt{PARTYCODE}})$, *PARTYCODE foreign key references PARTY*

$\mathtt{PRESENT}(\underline{\mathtt{GUESTNAME}}, \underline{\mathtt{PARTYCODE}}, \underline{\mathtt{TYPE}})$, *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.