# Databases

## Cap. 8. SQL Data Aggregation

Textbook: Ramakrishnan, Gehrke, “Database Management Systems”, McGraw Hill, 2003

2017 UPT Conf.Dr. Dan Pescaru

# Aggregate functions

1. Aggregate functions are functions that take a collection of values as input and return a single value
2. They are used to compute summaries of data in a table
3. Most aggregate functions (all except MIN, MAX, COUNT) work on a single column or expression of *numeric data*

# Behaviour of aggregate functions

1. Main characteristics of the aggregation functions:
   A. Operate - on a single column or expression from a group of rows
   B. Return - a single value for a group of rows
   C. Used only in the SELECT list and in the HAVING clause
   D. Anonymous result - use an alias to name the result column

# Projection list for aggregate functions

1. Projection list may contain only aggregation functions and distinct values per set:
   A. Columns with the single value per entire set of resulting rows

E.g.

SELECT ~~name,~~ COUNT(*) FROM Sailors;

- Note: it is not checked by MySQL. However, it has no meaning!

# Controlling the input set

1. To control the input set add:
   - DISTINCT: to consider only distinct values of the argument expression
   - ALL: to consider all values including all duplicates
2. E.g.:

```
SELECT COUNT( DISTINCT col) AS N
        FROM Table
```

# Type of aggregate functions

A. COUNT: returns the number of rows
B. SUM: returns the sum of the entries in a column
C. AVG: returns the average entry in a column
D. MIN, MAX: return the minimum and maximum entries in a column
E. VARIANCE: measures how far a set of numbers is spread out from the mean
F. STDDEV: returns the sample standard deviation of a set of numerical values

# COUNT()

1. Returns the number of values in the specified column
2. Input parameter
   - [DISTINCT/ALL] Column: do not consider Null values, and do not consider/consider duplicates
   - * : counts all the rows regardless of whether Nulls or the duplicate occur

```
SELECT   COUNT(DISTINCT CustomerID)
                                AS NrOfCustomers
         FROM Invoices;
```

# SUM()

1. Returns the sum of the values in a specified column
2. Ignores Null values (treated as 0). If all values are Null, return Null

- E.g.

```
SELECT SUM(Salary) AS SalaryBudget
        FROM Employees
        WHERE department_id=101;
```

# AVG()

1. Returns the average of the values in a specified column
2. Ignores Null values (they are not treated as 0). If all values are Null, return Null
3. Caution: using DISTINCT and ALL in AVG

- E.g.

```
SELECT SUM(age)/COUNT(*) AS sumAvg,
                            AVG(age)  AS theAVG
        FROM Sailors;
```

# MIN(), MAX()

1. MIN() returns the smallest value of a column
2. MAX() returns the largest value of a column
3. Ignores Null values (they are not treated as 0). If all values are Null, return Null
4. Characters: comparison is case sensitive/ insensitive in Oracle/MysQL
5. Note: DISTINCT has no effect

- E.g.

```
SELECT MIN(age) AS minAge, MAX(age)  AS maxAge
                FROM Sailors;
```

# VARIANCE()

1. Returns the variance of a set of expression values (measures how far a set of numbers is spread out around the mean). It express the average of the squared differences from the Mean
2. $\sigma = (\text{SUM}(\text{exp}^2) - \text{SUM}^2(\text{exp}) / \text{COUNT}(\text{exp})) / (\text{COUNT}(\text{exp}) - 1)$
3. Returns 0 if result contains just one row
4. Ignores Null values (they are not treated as 0). If all values are Null, return Null
5. Note: DISTINCT has effect. E.g.:

```
SELECT variance(age) AS ageVariance FROM Sailors;
```

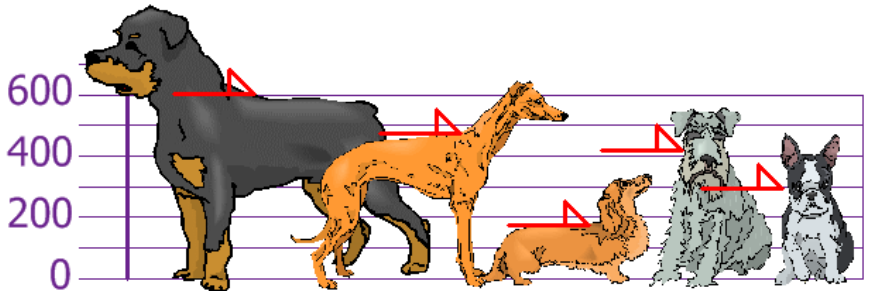
# STDDEV()

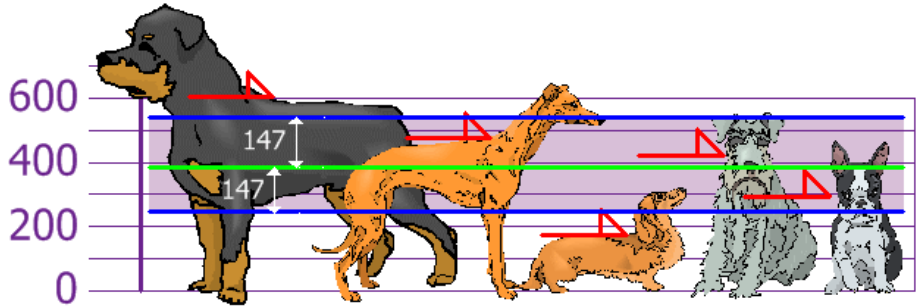
1. Returns the standard deviation of an set of expression values (defines de "normality" around the mean)
2. It is the square root of the Variance
3. Returns 0 if result contains just one row
4. Ignores Null values (they are not treated as 0). If all values are Null, return Null
5. Note: DISTINCT has effect

- E.g.

```
SELECT STDDEV(age) AS ageStandardDeviation
        FROM Sailors;
```

# Standard deviation




1. mean = 394 mm
2. variance = 21704
3. standard deviation = 147 mm
4. standard interval ("normality") = [247.. 541] mm
5. extra-large: 600 mm, extra small: 170 mm

(Ref: http://www.mathsisfun.com/)

# Grouping data (I)

1. Why: compute count(*) for each rank (how about for each age?)
2. GROUP BY: groups the data from the result and produce a single summary row for each group
3. E.g

```
SELECT   dep_id AS Department
         MAX(salary) AS maxDepSalary
  FROM Employees
  GROUP BY dep_id;
```

# Grouping data (II)

1. The projection list must contain only aggregation functions and expressions unique over each group
2. GROUP BY could contains more than one criteria (not in a hierarchal manner)
3. To increase result clarity, projection list may contain all GROUP BY expressions

```
SELECT age, rank, COUNT(*) AS nrOfSailors
    FROM Sailors
    GROUP BY rank, age;
```

# Grouping data: ordering groups

- ORDER BY used to control the hierarchy

```
SELECT rank, age, COUNT(*) AS nrOfSailors
    FROM Sailors
    GROUP BY rank, age ORDER BY rank;
```

vs.

```
SELECT rank, age, COUNT(*) AS nrOfSailors
    FROM Sailors
    GROUP BY rank, age ORDER BY age;
```

# Grouping data: adding more fields

- To add additional fields to projection list they must be included in GROUP BY clause

```
SELECT d.deptno, d.dname, AVG(e.sal)
    FROM (emp e INNER JOIN dept d
                ON e.deptno=d.deptno)
    GROUP BY d.deptno;
```

```
SELECT d.deptno, d.dname, AVG(e.sal)
    FROM (emp e INNER JOIN dept d
                ON e.deptno=d.deptno)
    GROUP BY d.deptno, d.dname;
```

# HAVING (I)

1. It is designed to be used with GROUP BY so that it can restrict the groups that appear in the final result table
2. Sometimes replaceable with WHERE

```
SELECT rank, COUNT(*) AS NrSail
    FROM Sailors GROUP BY rank HAVING rank>3;

SELECT rank, COUNT(*) AS NrSail
    FROM Sailors WHERE rank>3
    GROUP BY rank;
```

# HAVING (II)

1. Most of the time more efficient than WHERE
2. Not all the times replaceable by WHERE

```
SELECT rank, COUNT(*) AS NrSail
FROM Sailors
GROUP BY rank
HAVING COUNT(*)>1;
```

# Combining aggregate functions

1. Aggregate functions can be combined in arithmetical expressions
2. E.g

```
SELECT   dep_id AS Department
         (MAX(salary) - MIN(salary)) AS salRange
  FROM Employees
  GROUP BY dep_id;
```

# Nesting aggregate functions

1. Aggregate functions can be nested over groups of records (not supported in MySQL)
2. E.g

```
SELECT AVG(MAX(salary))
    FROM Employees
    GROUP BY dep_id;
```

- This calculation evaluates the inner aggregate (MAX(salary)) for each group defined by the GROUP BY clause (dep_id), and aggregates the results again

# Nesting aggregate functions

1. MySQL solution:subquery
2. E.g

```
SELECT AVG(s.maxs) FROM
    (SELECT MAX(salary) AS maxs
                FROM Employees
                GROUP BY dep_id) s;
```

# Aggregation functions and subqueries (I)

1. Aggregate functions can be used with SQL subqueries
2. E.g. Extract all sailors that have the age greater than average

```
SELECT *
    FROM Sailors
    WHERE age >
            (SELECT AVG(age) FROM Sailors);
```

# Aggregation functions and subqueries (II)

1. E.g. Extract only older sailors (we don't know the age of the older sailors)

```
SELECT *
    FROM Sailors
    WHERE age =
            (SELECT MAX(age) FROM Sailors);
```

# Examples (I)

1. Problem:

   - List all sailors with the total number of reserves made by each of them

# Examples (I)

Answer 1:

```
SELECT s.*,
    (SELECT count(*) FROM Reserves r
            WHERE r.sid=s.sid) AS nrReserves FROM
Sailors s;
```

Answer 2:

```
SELECT s.sid,s.name,COUNT(*) AS nrReserves
    FROM Sailors s INNER JOIN  Reserves r
                        ON s.sid=r.sid
    GROUP BY s.sid,s.name;
```

Note: how about 0 reserves (Why LEFT JOIN is wrong?
Solution: COUNT(r.date))

# Examples (II)

1. Problem:

   - Find the age of the youngest sailor with age > 18, for each rating with at least 2 sailors of any age

# Examples (II)

1. Answer

```
SELECT s1.rank, MIN(s1.age) AS youngAge
    FROM Sailors s1
    WHERE s1.age > 18
    GROUP BY  s1.rank
    HAVING 1  <  (SELECT COUNT(*)
                  FROM Sailors s2
                  WHERE s1.rank=s2.rank);
```

Note: How about HAVING COUNT(*)>1?