

DBDM

DataBases and Data Mining

E.Coquery, R.Thion, A. Bonifati, M. Plantevit, M. Kaytoue, C.
Robardet

`emmanuel.coquery@liris.cnrs.fr`

`http://liris.cnrs.fr/~ecoquery/dbdm/`

Course Overview - Database part

- DBMS
- Relational model
- SQL, Relational calculus
- Relational algebra, Query optimization
- Functional dependencies, Armstrong axioms, Armstrong relations
- Inclusion dependencies
- Data exchange, Chase algorithm, Query rewriting

Course Overview - Data Mining part

- Introduction to data mining
- Usual algorithms for set based patterns
- Constrained data mining
- Advanced pattern languages (FCA, sequences, dynamic graphs)
- (Bi-|Co-) Clustering

Outline

- 1 Introduction
 - Files
 - Database Management System
- 2 Relational model
- 3 SQL

Data(base)

A dataset is:

- Some objects
 - a name, e.g. Emmanuel
 - a course, e.g. DBDM
 - a date, e.g. January 1st, 2016
 - ...
- But also relations between such objects
 - Emmanuel teaches the “DBDM” course on January 1st, 2016

A Database (DB) is an application for storing, querying and updating a dataset.

Files

Files can be used for storing a dataset:

- Collection of applications where each of them defines and manages its own files.
- A file is a set of records containing related data.
 - One can use various libraries to ease reading/writing such files
 - record files in Pascal
 - serialization API in Java
 - JSON in Javascript (and others)
 - ...
- Require a tight coupling between program and files
 - File management is directly integrated into the program.

Example

Data on students in some university

- Student's address is used when he registers, for library access, etc.
- Each application must manage a set of data files and ensure they are up to date.
- Datafile format may vary
- Updates are done several times, which is error prone
 - e.g. address update: at the registration, at the library, etc.

Some problems when dealing with files

- Data access can be complex
 - In practice, complex data require to write a lot of code to be accessed.
 - Efficient access requires to right complex optimized code, even for simple applications.
- Separated files: redundancy both in definition and storage of data.
- Security problems: a breach in the program can compromise the whole file content (for confidentiality and integrity).
- No concurrency control: consistency problem may arise from simultaneous files access leading to data corruption.

Databases

Objective: avoid data access problem induced by direct file access

A database is a dataset:

- which is stored
- whose structure depends on the data, not on the application
- consistent
- minimally redundant
- accessible by several users concurrently

Who does what

The designer manages:

- logical structure
- non redundancy
- sharing (and distribution) of data

The Database Management System (DBMS) manages:

- storage
- data availability
- data access
- concurrency

DBMS

DBMS: Set of software tools allowing to create and use a database.

DBMS **functions**:

- Database definition
 - datatype specification
 - data organization
 - integrity constraints on stored data
- data querying
- data updates
- ensure data integrity
- manage concurrency
- security
 - manage data confidentiality

Database schema

- Centralized description of the database through a Data Description Language (DDL):
 - data organization
 - data types
 - integrity constraints

- Unique, shared between applications
 - ⇒ one application does not guide data organization

Manipulating data

- Tools and systems to enable communication between the database and the applications using data.
- Searching, creating, updating, deleting data.
- Data Manipulation Language (DML):
 - Declarative: describe what you want instead of how to get it.
- Data is independant from programs

Interacting with DBMSs

- Shells
- GUI
- Programmatically:
 - C, C++, Java, Python, PHP, OCaml,
(put your favorite practical language here)
 - libraries for sending (DML) queries to the DBMS.

Data integrity

- Integrity constraints, specified in DDL
 - enforced by the DBMS
 - with the possibility to be programmed for complex ones
- Execution safety and recovery
- Storage
 - Action logging
- Concurrency
 - Lock mechanisms (minimize performance impacts)
- Transactions: commits and rollbacks

Security and confidentiality

- Data sharing
- Authentication
- Autorisations
- Views for selective data access

Typical DBMS architecture

3 layers:

- External layer: user/application interaction
- Logical layer:
 - global control and data organization
- Internal layer:
 - data storage on physical devices,
 - management of persistence and access structures (files, indexes, etc)

Outline

- 1 Introduction
 - Files
 - Database Management System
- 2 Relational model
- 3 SQL

Data model

Defines a way to represent information

- A way to represent data (DDL)
- A way to represent data constraints (DDL)
- A set of operations to manipulate data (DML).

Independent from physical data representation. Simplifies:

- administration
- optimization
- usage

Relational model

Set based

- Objects are simple, **atomic**:
 - integers, floats, strings, dates, ...
- No complex data structures:
 - No lists, tables, records, ...
- **Relations** are used to represent and manipulate data
 - seen as subsets of cartesian products
- Usual operations on sets
 - Union, intersection, difference
 - Cartesian product

Relational model advantages

Fundamentally simple

- easier to understand
- easier to optimize

but expressive enough

- ways to represent complex objects through relations

used in practice since the 80's in numerous DBMS nombreux
SGBD:

- Oracle, MySQL, PostgreSQL, DB2, SQL Server, Sqlite ...

Relational schema

Composed of

- a set of **attributes**
 - describes atomic data to manipulate
 - ex: title, year, genre
- a set of **relations** or **tables** on these attributes:
 - represent relationship between atomic data
 - can be used to represent complex objects (e.g. records)
 - ex: Movie(title, year, genre)
Vocabulary: title, year and genre are the attributes of relation Movie
- a **table** is a relation in a schema
 - whose content is extensional
 - as opposed to **views** whose content is intentional

Relational schema - 2

Constraints:

- Attribute **types**
 - `title: string, year: integer, genre: string`
 - Usually given with the relations definitions, e.g.
`Movie(title: string, year: integer, genre: string)`
- **Value domain:** set of instances of a given atomic type
 - e.g.: integers, reals, character strings, etc
- More complex constraints such as:
 - “In the relation `Movie`, there can only be one year and one genre for a given title.” (functional dependency)

Designing schemas

- The choice of relations is fundamental and usually complex:
 - it determines essential qualities of the database:
performance, accuracy, exhaustivity, availability of information

- Some methodologies can help:
 - ER-diagrams
 - UML

Instances, in theory

In theory

A **database instance** is a set of relation instances (one per relation of the database schema)

A **relation instance** of a relation $R(A_1, \dots, A_n)$ is a subset of the cartesian product of the domain of its attributes:

- If D_1 is the domain (of the type) of A_1 , \dots , D_n is the domain (of the type) of A_n
- any instance of R is included in $D_1 \times \dots \times D_n$

Consequences:

- order between elements is not important
- no duplication of tuples
- all possible values of attributes are known

Instances in practice

Real life more complex:

- bag semantics (possible duplication)
- order can be useful for the user
- user defined functions make values less predictable

Representing data using instances

Instances actually represent data:

| Movie | | |
|--------------|------|-----------------|
| title | year | genre |
| Alien | 1979 | Science-fiction |
| Vertigo | 1958 | Thriller |
| Face/Off | 1997 | Crime |
| Pulp fiction | 1995 | Crime |

Instance is a set of tuples:

$\{(Alien, 1979, Science-fiction), (Vertigo, 1958, Thriller), (Volte-face, 1997, Crime), (Pulp fiction, 1995, Crime)\}$

What is stored is the instances

Manipulating data

Data querying is done through relation manipulation

- Operations:
 - Input: one or several relations (more precisely relation instances)
 - that can be stored tables or not (e.g. dynamic views)
 - Output: one relation
- operation kinds:
 - selecting interesting tuples
 - Usual set-theoretic operations: union, intersection, difference, cartesian product

Updates: adding/deleting tuples in tables

Two approaches for DML languages

- Logical approach: relational calculus
- Algebraic approach: relational algebra

Same expressive power

Concrete language for users and developers: SQL

- Can be seen from both points of views

Outline

- 1 Introduction
 - Files
 - Database Management System
- 2 Relational model
- 3 SQL

SQL

- Concrete language, includes DDL and DML
- From IBM in the 70's
- Standards:
 - SQL-87: 1987 (ISO)
 - SQL-2: 1992 (ANSI)
 - SQL-3: 1999
 - SQL-2003
 - SQL-2006

Projection

```
SELECT att1, att2, ...  
FROM table_name;
```

- Obtain values in table *table_name*, while keeping only attributes *att*₁, *att*₂, ...

One can replace *att*₁, *att*₂, ... by * to get all attributes.

Example

Schema:

Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Give the name and position of each employee:

- `SELECT Nom,Fonction FROM Employe;`

Demo

Example 2

Schema:

Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Give available information for each employee:

- `SELECT * FROM Employe;`

Demo

DISTINCT

DISTINCT allows to remove duplicates (you have do it explicitly in practice)

Example:

Give the various positions the company:

- `SELECT DISTINCT Fonction FROM Employee;`

Demo

Selecting specific tuples

```
SELECT att1, att2, ...  
FROM table_name  
WHERE condition
```

- The **WHERE** clause specify *condition* for choosing tuples to keep.

Conditions in WHERE

Simple expressions:

- Comparisons (`=`, `!=`, `<`, `<=`, `>`, `>=`)
- between attributes or constants
- constants for each (atomic) data type
 - numbers: 1, 1980, 1.5
 - strings: 'Martin', 'directeur'
 - dates: '1980-06-18'
 - date formatting varies w.r.t. DBMS

Logical connectors that can be used: **AND**, **OR**

Example

Schema:

Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Who are the employees whose employment (embauche) date is before January 1st 1999 ?

- ```
SELECT Nom
FROM Employe
WHERE Embauche < '1999-01-01';
```

Demo

## Example 2

Schema:

Employe(Nom, Num, Fonction, Num\_sup, Embauche, Salaire, Num\_Dept)

Who are the employees whose employment (embauche) date is before January 1<sup>st</sup> 1999 and that are paid at least 30000 euros a year:

- ```
SELECT Nom
FROM Employe
WHERE Embauche < '1999-01-01'
AND Salaire >= 30000;
```

Demo

Other conditions

- Operator **IN**: as the set operation
 - Which employees are director ou engineer ?
SELECT Nom, Fonction
FROM Employe
WHERE Fonction **IN** ('ingenieur','directeur');
- Operator **BETWEEN ... AND** for specifying value intervals:
 - Employee that are paid between 25000 and 30000 euros ?
SELECT Nom, Salaire
FROM Employe
WHERE Salaire **BETWEEN** 25000 **AND** 30000;

Another example

```
SELECT Nom, Embauche, Fonction, Salaire
FROM Employe
WHERE Fonction IN ('ingenieur','directeur')
AND Embauche BETWEEN '1990-01-01' AND '1999-12-31'
AND Salaire < 32000;
```

condition, connector \wedge

Undefined values

Some values may be actually undefined in practice:

- represented by the keyword **NULL**.
- can be tested with **IS NULL** / **IS NOT NULL**

Schema: Batiment(Num_bat, Nom_bat, Ent_princ, Ent_Sec)

- Buildings with no secondary entrance have NULL as a “value” for attribute Ent_Sec.
- ```
SELECT *
FROM Batiment
WHERE Ent_sec IS NULL;
```

## Sorting query results

While the result of a query is unsorted, it is possible to sort it afterwards

```
SELECT att1, att2, ...
FROM table_name
WHERE condition
ORDER BY attj, attj, ...
```

- lexicographic order on the values specified by the ORDER BY clause
- In ORDER BY, it is possible to specify either **ascending** or **descending** order after the value using **ASC** or **DESC** after the value
  - defaults to ASC

## Example

Schema:

Employe(Nom, Num, Fonction, Num\_sup, Embauche, Salaire, Num\_Dept)

Give employees name in dept. 20, sorted by salary decreasing then by name in alphabetical order

```
SELECT Nom
FROM Employe
WHERE Num_dept=20
ORDER BY Salaire DESC, Nom;
```

## Querying several tables

```
SELECT att1, att2, ...
FROM table_name1, table_name2, ...
WHERE condition
ORDER BY atti, attj, ...
```

- Cartesian product
- If one attribute is in several tables, use table name for disambiguation using *table\_name.att*

# Joins

Join: special case of cartesian product with a filtering condition that allow to combine only related tuples.

“Natural” join: condition is equalities between attributes shared among two relations:

Schema  $R(A_1, A_2, B_1, B_2)$  and  $S(C_1, C_2, B_1, B_2)$

```
SELECT A1, A2, R.B1, S.B2, C1, C2
FROM R, S
WHERE R.B1=S.B1 AND R.B2=S.B2
```

## Example

Schema:

Batiment(Num\_bat, Nom\_bat, Ent\_princ, Ent\_Sec)

Departement(Num\_dept, Nom\_dept, Num\_bat)

Departments and their related buildings:

- ```
SELECT Num_dept, Nom_dept, Batiment.Num_bat,  
       Nom_bat, Ent_princ, Ent_sec  
FROM Departement, Batiment  
WHERE Departement.Num_bat = Batiment.Num_bat;
```

Renaming

When using a table several times, one needs to rename it:

```
SELECT att1, att2, ...  
FROM table_name1 new_name1,  
      table_name2 new_name2, ...  
WHERE condition  
ORDER BY atti, attj, ...
```

Values in SELECT can be renamed using AS.

Example

Schema:

Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Give each employee's name and the name of his manager.

- ```
SELECT Employe.Nom, Employe.Fonction,
 Chef.Nom AS Superieur
FROM Employe, Employe Chef
WHERE Chef.Num = Employe.Num_sup;
```

## Example 2

Schema:

Employe(Nom, Num, Fonction, Num\_sup, Embauche, Salaire, Num\_Dept)

Who are the employees that are paid less than Bellot.

```
SELECT Employe.Nom, Employe.Salaire
FROM Employe, Employe bel
WHERE Employe.Salaire < bel.Salaire
AND bel.Nom = 'Bellot';
```

# Subqueries

Using the result of a query in another one

- Better expressivity through negation
- Subqueries can be used in:
  - WHERE
  - FROM (needs renaming)
  - SELECT (only if the subquery yields one atomic value for each tuple in the main query).
    - not checked statically
  
- Name clashes: natural scoping rules

## Example

If the subquery yields only one result

Schema:

Employe(Nom, Num, Fonction, Num\_sup, Embauche, Salaire, Num\_Dept)

Which employee have the same position as Jones ?

```
SELECT Nom
FROM Employe
WHERE Fonction =
 (SELECT Fonction
 FROM Employe
 WHERE Nom='Jones');
```

## Example: Subquery related to main query

Schema:

Employe(Nom, Num, Fonction, Num\_sup, Embauche, Salaire, Num\_Dept)

Which employees are in a different departement than the one of their manager?

```
SELECT Nom
FROM Employe Emp
WHERE Num_dept !=
 (SELECT Num_dept
 FROM Employe
 WHERE Emp.Num_sup = Num);
```

Exercice: rewrite without subquery

# Subqueries with more than one result

## Special operators

- $a$  **IN** (*subquery*)
  - true  $a$  is in the result of *subquery*.
- $a$   $\square$  **ANY** (*subquery*)  
where  $\square$  can be  $\{=, <, >, \leq, \geq\}$ 
  - true if there exists some  $b$  in the result of *subquery* such that  $a \square b$ .
- $a$   $\square$  **ALL** (*subquery*)  
where  $\square$  can be  $\{=, <, >, \leq, \geq\}$ 
  - true if for all values  $b$  in the result of *subquery*,  $a \square b$ .
- **EXISTS** (*subquery*)
  - true if the result of *subquery* is not empty

## Example

Schema:

Employe(Nom, Num, Fonction, Num\_sup, Embauche, Salaire, Num\_Dept)

```
SELECT Nom, Salaire
FROM Employe
WHERE Salaire > ALL (SELECT Salaire
 FROM Employe
 WHERE Num_dept = 20);
```

## Example 2

Schema:

Employe(Nom, Num, Fonction, Num\_sup, Embauche, Salaire, Num\_Dept)

```
SELECT Nom
FROM Employe Chef
WHERE EXISTS (SELECT Nom
 FROM Employe
 WHERE Employe.Num_sup = Chef.Num);
```



## Subqueries with several attributes in SELECT

Tuples  $(a, b, \dots)$  can be used to compare with query results having several values in SELECT

Schema:

Employe(Nom, Num, Fonction, Num\_sup, Embauche, Salaire, Num\_Dept)

```
SELECT Nom
FROM Employe
WHERE (Fonction, Num_sup) = (SELECT Fonction, Num_sup
 FROM Employe
 WHERE Nom='Bellot');
```

## Nested subqueries

Employe(Nom, Num, Fonction, Num\_sup, Embauche, Salaire, Num\_Dept)

```
SELECT Nom, Fonction
FROM Employe
WHERE Num_dept = 20
AND fonction IN
 (SELECT Fonction
 FROM Employe
 WHERE Num_dept = (SELECT Num_dept
 FROM Employe
 WHERE Nom = 'Dupont'));
```

## Set theoretic operators on relations

- Allow for combining several SELECT/FROM statements.
  - $\cup$ : UNION
  - $\cap$ : INTERSECTION
  - $\setminus$ : MINUS
- Set semantics (implicit DISTINCT).
- Each SELECT must have the same number of attributes
- Attribute names in the results are given by the first SELECT.
  - Matching between tuples is position wise (not name wise)
- The last SELECT can contain an ORDER BY for sorting the whole result

## Example

Schema:

Employe1(Nom, Num, Fonction, NumSup, Embauche, Salaire, NumDept)

Employe2(Nom, Num, Fonction, Numsup, Embauche, Salaire, NumDept)

```
(SELECT NumDept FROM Employe1)
INTERSECT
(SELECT NumDept FROM Employe2);
```

# Expressions

Complex expressions are possible for values

- Arithmetic expressions
- String expressions
- Date expressions
- Conversion / cast functions

**Aggregation** functions can be used to handle a collection of values.

## Expressions - 2

Expressions are usable:

- In SELECT:
  - default name from expression, not really usable. Use AS to rename.
- In WHERE
- In ORDER BY

## Example

Schema:

Employe(Nom, Num, Fonction, Num\_sup, Embauche, Salaire, Num\_Dept)

```
SELECT Nom, (Salaire + Commission) Revenu
FROM Employe
WHERE Fonction = 'commercial';
```

## Example - 2

Schema:

Employe(Nom, Num, Fonction, Num\_sup, Embauche, Salaire, Num\_Dept)

```
SELECT Nom, (Commission/Salaire) Rapport
FROM Employe
WHERE Fonction = 'commercial'
ORDER BY Commission/Salaire;
```



## Example - 3

Schema:

Employe(Nom, Num, Fonction, Num\_sup, Embauche, Salaire, Num\_Dept)

```
SELECT Nom, ROUND(Salaire/(22*12), 2) SJournalier
FROM Employe
WHERE Commission <= Salaire * 0.5;
```

## Example - 4

Schema:

Employe(Nom, Num, Fonction, Num\_sup, Embauche, Salaire, Num\_Dept)

For each employee, the number of days since the employe was recruited.

```
SELECT Nom, DATEDIFF(SYSDATE(),Embauche) AS days
FROM Employe;
```

## Naïve operational semantics

```
SELECT att1, att2, ...
FROM table1, table2, ...
WHERE condition
ORDER BY atti, attj, ...
```

- Retrieve data specified in FROM  
→ cartesian product  $table_1 \times table_2 \times \dots$
- Filter tuples using condition in WHERE
- Sort tuples according to ORDER BY
- Compute SELECT for each tuple and output result

## Naïve operational semantics - 2

```
SELECT att1, att2, ...
FROM table1, table2, ...
WHERE condition
ORDER BY atti, attj, ...
```

- Subqueries in FROM are executed just before the cartesian product
- Subqueries in WHERE/ORDER BY/SELECT are executed for each tuple to be filtered/sorted/computed

Of course the DBMS optimizes the execution of queries

- e.g. subqueries in WHERE that do not depend on the main query are executed just once
- more on optimization on the next course

## Example

Schema:

Departement(Num\_dept, Nom\_dept, Num\_bat)

Batiment(Num\_bat, Nom\_bat, Ent\_princ, Ent\_Sec)

```
SELECT Nom_dept, Batiment.Nom_bat
FROM Departement, Batiment
WHERE Departement.Num_bat = Batiment.Num_bat
ORDER BY Nom_dept;
```

## Example - 2

| Departement |               |         | Batiment |            |           |         |
|-------------|---------------|---------|----------|------------|-----------|---------|
| Num_dept    | Nom_dept      | Num_bat | Num_bat  | Nom_bat    | Ent_princ | Ent_Sec |
| 10          | Marketing     | 1       | 1        | Turing     | Nord      | Ouest   |
| 20          | Developpement | 2       | 1        | Turing     | Nord      | Ouest   |
| 30          | Direction     | 3       | 1        | Turing     | Nord      | Ouest   |
| 10          | Marketing     | 1       | 2        | Einstein   | Ouest     | NULL    |
| 20          | Developpement | 2       | 2        | Einstein   | Ouest     | NULL    |
| 30          | Direction     | 3       | 2        | Einstein   | Ouest     | NULL    |
| 10          | Marketing     | 1       | 3        | Newton     | Sud       | Nord    |
| 20          | Developpement | 2       | 3        | Newton     | Sud       | Nord    |
| 30          | Direction     | 3       | 3        | Newton     | Sud       | Nord    |
| 10          | Marketing     | 1       | 4        | Pointcarre | Est       | NULL    |
| 20          | Developpement | 2       | 4        | Pointcarre | Est       | NULL    |
| 30          | Direction     | 3       | 4        | Pointcarre | Est       | NULL    |

FROM Departement, Batiment

## Example - 2

| Departement |               |         | Batiment |            |           |         |
|-------------|---------------|---------|----------|------------|-----------|---------|
| Num_dept    | Nom_dept      | Num_bat | Num_bat  | Nom_bat    | Ent_princ | Ent_Sec |
| 10          | Marketing     | 1       | 1        | Turing     | Nord      | Ouest   |
| 20          | Developpement | 2       | 1        | Turing     | Nord      | Ouest   |
| 30          | Direction     | 3       | 1        | Turing     | Nord      | Ouest   |
| 10          | Marketing     | 1       | 2        | Einstein   | Ouest     | NULL    |
| 20          | Developpement | 2       | 2        | Einstein   | Ouest     | NULL    |
| 30          | Direction     | 3       | 2        | Einstein   | Ouest     | NULL    |
| 10          | Marketing     | 1       | 3        | Newton     | Sud       | Nord    |
| 20          | Developpement | 2       | 3        | Newton     | Sud       | Nord    |
| 30          | Direction     | 3       | 3        | Newton     | Sud       | Nord    |
| 10          | Marketing     | 1       | 4        | Pointcarre | Est       | NULL    |
| 20          | Developpement | 2       | 4        | Pointcarre | Est       | NULL    |
| 30          | Direction     | 3       | 4        | Pointcarre | Est       | NULL    |

WHERE Departement.Num\_bat = Batiment.Num\_bat

## Example - 3

| Departement |               |         | Batiment |          |           |         |
|-------------|---------------|---------|----------|----------|-----------|---------|
| Num_dept    | Nom_dept      | Num_bat | Num_bat  | Nom_bat  | Ent_princ | Ent_Sec |
| 20          | Developpement | 2       | 2        | Einstein | Ouest     | NULL    |
| 30          | Direction     | 3       | 3        | Newton   | Sud       | Nord    |
| 10          | Marketing     | 1       | 1        | Turing   | Nord      | Ouest   |

ORDER BY Nom\_dept

| Nom_dept      | Num_bat  |
|---------------|----------|
| Developpement | Einstein |
| Direction     | Newton   |
| Marketing     | Turing   |

SELECT Nom\_dept, Batiment.Nom\_bat



# Grouping

```
SELECT att1, att2, ...
FROM table1, table2, ...
WHERE condition
GROUP BY attk, attl, ...
ORDER BY atti, attj, ...
```

- Grouping occurs just after the WHERE filter
- partitions the collection of tuples according to the values specified by GROUP BY
  - greatest groups such that two tuples in the same partition have the same value for *att*<sub>*k*</sub>, *att*<sub>*l*</sub>, ...
  - quotient by the equivalence relation consisting in having the same values for *att*<sub>*k*</sub>, *att*<sub>*l*</sub>, ...

## Grouping - 2

- A query produce one tuple per group.
- SELECT and ORDER BY can only directly use attributes/values specified in GROUP BY.
  - since these values are fixed in a group
  - Other attributes can not be used directly (as their value varies)

## Example

```
Schema: Employe(Nom, Num, Fonction, Salaire, Num_Dept)
SELECT Fonction, Num_Dept
FROM Employe
GROUP BY Fonction, Num_Dept
ORDER BY Num_Dept;
```

| Nom       | Num   | Fonction     | Salaire | Num_dept |
|-----------|-------|--------------|---------|----------|
| Bellot    | 13021 | ingenieur    | 25000   | 20       |
| Dupuis    | 14028 | commercial   | 20000   | 10       |
| LambertJr | 15630 | stagiaire    | 6000    | 20       |
| Martin    | 16712 | directeur    | 40000   | 30       |
| Dupont    | 17574 | gestionnaire | 30000   | 30       |
| Jones     | 19563 | ingenieur    | 20000   | 20       |
| Brown     | 20663 | ingenieur    | 20000   | 20       |
| Lambert   | 25012 | directeur    | 30000   | 20       |
| Fildou    | 25631 | commercial   | 20000   | 10       |
| Soule     | 28963 | directeur    | 25000   | 10       |

## Example - 2

```
SELECT Fonction
FROM Employe
GROUP BY Fonction, Num_Dept
```

| Nom       | Num   | Fonction     | Salaire | Num_dept |
|-----------|-------|--------------|---------|----------|
| Bellot    | 13021 | ingenieur    | 25000   | 20       |
| Jones     | 19563 | ingenieur    | 20000   | 20       |
| Brown     | 20663 | ingenieur    | 20000   | 20       |
| Dupuis    | 14028 | commercial   | 20000   | 10       |
| Fildou    | 25631 | commercial   | 20000   | 10       |
| LambertJr | 15630 | stagiaire    | 6000    | 20       |
| Martin    | 16712 | directeur    | 40000   | 30       |
| Dupont    | 17574 | gestionnaire | 30000   | 30       |
| Lambert   | 25012 | directeur    | 30000   | 20       |
| Soule     | 28963 | directeur    | 25000   | 10       |

## Example - 3

ORDER BY Num\_Dept

| Nom       | Num   | Fonction     | Salaire | Num_dept |
|-----------|-------|--------------|---------|----------|
| Dupuis    | 14028 | commercial   | 20000   | 10       |
| Fildou    | 25631 | commercial   | 20000   | 10       |
| Soule     | 28963 | directeur    | 25000   | 10       |
| Bellot    | 13021 | ingenieur    | 25000   | 20       |
| Jones     | 19563 | ingenieur    | 20000   | 20       |
| Brown     | 20663 | ingenieur    | 20000   | 20       |
| LambertJr | 15630 | stagiaire    | 6000    | 20       |
| Lambert   | 25012 | directeur    | 30000   | 20       |
| Martin    | 16712 | directeur    | 40000   | 30       |
| Dupont    | 17574 | gestionnaire | 30000   | 30       |

## Example - 4

```
SELECT Fonction, Num_Dept
```

| Fonction     | Num_dept |
|--------------|----------|
| commercial   | 10       |
| directeur    | 10       |
| ingenieur    | 20       |
| stagiaire    | 20       |
| directeur    | 20       |
| directeur    | 30       |
| gestionnaire | 30       |

# Aggregation functions

- Operate on a set of atomic values.
- Usable in GROUP BY queries to compute a value from a set of values coming from the tuples in a group
- Used in SELECT and ORDER BY.
- *Not* in WHERE.  
(As the WHERE filter occurs *before* grouping.)
- For example,  $AVG(e)$  return the average of the values  $e$  of each tuple in the group.

## Example

Schema:                    Employe(Nom, Num, Fonction, Salaire, Num\_Dept)

Average salary for each position:

```
SELECT Fonction, AVG(Salaire) SalaireMoyen
FROM Employe
GROUP BY Fonction;
```



# Standard aggregation functions

- *COUNT*(*e*): bag semantics (a value can be counted more than one).
  - tuples for which *e* is NULL are not counted.
  - \* can replace *e* for counting tuples
- *MAX*(*e*)
- *MIN*(*e*)
- *SUM*(*e*)
- *AVG*(*e*)
- *STDDEV*(*e*)
- *VARIANCE*(*e*)

*e* can be preceded by DISTINCT for set semantics

- Important for COUNT, SUM, AVG, STDDEV and VARIANCE.

## Example

Schema:

Employe(Nom, Num, Fonction, Salaire, Num\_Dept)

Departement(Num\_dept, Nom\_dept, Num\_bat)

```
SELECT Nom_dept, COUNT(DISTINCT Fonction) NbFonctions
FROM Employe, Departement
WHERE Employe.Num_dept = Departement.Num_dept
GROUP BY Departement.Num_dept, Nom_dept;
```

## Example - 2

Schema:                    Employe(Nom, Num, Fonction, Salaire, Num\_Dept)

```
SELECT Num_dept, Nom, Salaire
FROM Employe
WHERE (Num_dept, Salaire) IN
 (SELECT Num_dept, MAX(Salaire)
 FROM Employe
 GROUP BY Num_dept);
```

## Filtering groups

```
SELECT att1, att2, ...
FROM table1, table2, ...
WHERE condition
GROUP BY attk, attl, ...
HAVING group_condition
ORDER BY atti, attj, ...
```

- WHERE can filter individual tuples, not groups
- HAVING is for filtering groups
  - same rules as SELECT and ORDER BY concerning usable values

## Example

```
SELECT Num_Dept, COUNT(DISTINCT Fonction) NbFonctions
FROM Employe
WHERE Salaire > 15000
GROUP BY Num_Dept
HAVING COUNT(*) > 2;
```

| Nom       | Num   | Fonction     | Salaire | Num_dept |
|-----------|-------|--------------|---------|----------|
| Bellot    | 13021 | ingenieur    | 25000   | 20       |
| Dupuis    | 14028 | commercial   | 20000   | 10       |
| LambertJr | 15630 | stagiaire    | 6000    | 20       |
| Martin    | 16712 | directeur    | 40000   | 30       |
| Dupont    | 17574 | gestionnaire | 30000   | 30       |
| Jones     | 19563 | ingenieur    | 20000   | 20       |
| Brown     | 20663 | ingenieur    | 20000   | 20       |
| Lambert   | 25012 | directeur    | 30000   | 20       |
| Fildou    | 25631 | commercial   | 20000   | 10       |
| Soule     | 28963 | directeur    | 25000   | 10       |

## Example - 2

```
FROM Employe WHERE Salaire > 15000
```

| Nom       | Num   | Fonction     | Salaire | Num_dept |
|-----------|-------|--------------|---------|----------|
| Bellot    | 13021 | ingenieur    | 25000   | 20       |
| Dupuis    | 14028 | commercial   | 20000   | 10       |
| LambertJr | 15630 | stagiaire    | 6000    | 20       |
| Martin    | 16712 | directeur    | 40000   | 30       |
| Dupont    | 17574 | gestionnaire | 30000   | 30       |
| Jones     | 19563 | ingenieur    | 20000   | 20       |
| Brown     | 20663 | ingenieur    | 20000   | 20       |
| Lambert   | 25012 | directeur    | 30000   | 20       |
| Fildou    | 25631 | commercial   | 20000   | 10       |
| Soule     | 28963 | directeur    | 25000   | 10       |

## Example - 3

GROUP BY Num\_Dept

| Nom     | Num   | Fonction     | Salaire | Num_dept |
|---------|-------|--------------|---------|----------|
| Bellot  | 13021 | ingenieur    | 25000   | 20       |
| Jones   | 19563 | ingenieur    | 20000   | 20       |
| Brown   | 20663 | ingenieur    | 20000   | 20       |
| Lambert | 25012 | directeur    | 30000   | 20       |
| Martin  | 16712 | directeur    | 40000   | 30       |
| Dupont  | 17574 | gestionnaire | 30000   | 30       |
| Dupuis  | 14028 | commercial   | 20000   | 10       |
| Fildou  | 25631 | commercial   | 20000   | 10       |
| Soule   | 28963 | directeur    | 25000   | 10       |

## Example - 4

HAVING COUNT(\*) &gt; 2

| Nom     | Num   | Fonction     | Salaire | Num_dept |
|---------|-------|--------------|---------|----------|
| Bellot  | 13021 | ingenieur    | 25000   | 20       |
| Jones   | 19563 | ingenieur    | 20000   | 20       |
| Brown   | 20663 | ingenieur    | 20000   | 20       |
| Lambert | 25012 | directeur    | 30000   | 20       |
| Martin  | 16712 | directeur    | 40000   | 30       |
| Dupont  | 17574 | gestionnaire | 30000   | 30       |
| Dupuis  | 14028 | commercial   | 20000   | 10       |
| Fildou  | 25631 | commercial   | 20000   | 10       |
| Soule   | 28963 | directeur    | 25000   | 10       |



## Example - 5

```
SELECT Num_Dept, COUNT(DISTINCT Fonction) NbFonctions
```

| Num_dept | NbFonctions |
|----------|-------------|
| 10       | 2           |
| 20       | 2           |

# Global grouping

Using aggregation function without GROUP BY:

- Implicit grouping with only one group
- SELECT can then only contain aggregation functions

# Example

Schema:

Employe(Nom, Num, Fonction, Salaire, Num\_Dept)

```
SELECT SUM(Salaire)
FROM Employe
WHERE Num_dept = 10;
```

## Double grouping

Nested use of aggregation functions in SELECT

- Possible only in a GROUP BY query.
- triggers two grouping:
  - The first normal one corresponding to the GROUP BY statement
  - A second implicit one because of the englobing aggregation function.
    - work as a global grouping

Remark: not always implemented as is but by can be recoded using a subquery in FROM

## Example

Schema:

Employe(Nom, Num, Fonction, Salaire, Num\_Dept)

Size of the largest departement for the number of employees

```
SELECT MAX(COUNT(*))
FROM Employe
GROUP BY Num_dept;
```

```
SELECT MAX(NbEmp)
FROM (SELECT COUNT(*) AS NbEmp
 FROM Employe
 GROUP BY Num_dept)
 CountEmp;
```