Lic. Informatique, Univ Lille 1, 2010-11

- **Introduction aux bases de données relationnelles**
  - 5ème séance: **Introduction à SQL**
  - **Opérations ensemblistes, sous requêtes**
- **Enseignante**: C. Kuttler
- **Biblio**: chapitre 3 de *Database Systems Concepts* de Silberschatz et al, McGraw-Hill (6ème edition, 2010)
- Ces transparents sont une adaptation de ceux disponibles sur le site du livre: **www.db-book.com**

Chapter 3: Introduction to SQL (2/2)

Database System Concepts, 6th Ed.

Chapter 3: Introduction to SQL

- Overview of The SQL Query Language
- Data Definition
- Basic Query Structure
- Aggregate Functions
- Additional Basic Operations

- **Additional Basic Operations**
- **Set Operations**
- **Nested Subqueries**
- **Null Values**
- **Modification of the Database**

Where Clause Predicates: More Comparison operators

- SQL includes a **between** comparison operator
  - Example: Find the names of all instructors with salary between $90,000 and $100,000 (that is, ≥ $90,000 and ≤ $100,000)
  - **select** *name*
    from *instructor*
    where *salary* **between** 90000 **and** 100000
- **Tuple comparison**.
  - Example: Find the names of all instructors from the biology department, and the courses they teach.
  - **select** *name*, *course_id*
    from *instructor*, *teaches*
    where (*instructor.ID*, *dept_name*) = (*teaches.ID*, 'Biology');

Set Operations – require union compatibility

- Find courses that ran in Fall 2009 **or** in Spring 2010

  (**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
  union
  (**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

- Find courses that ran in Fall 2009 **and** in Spring 2010

  (**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
  intersect
  (**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

- Find courses that ran in Fall 2009 **but not** in Spring 2010

  (**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
  minus
  (**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

Note: Other DBMS systems, different from Oracle, follow the SQL standard definition: keyword **except** instead of **minus**.

Set Operations

- Set operations **union**, **intersect**, and **except**
  - Each of the above operations **automatically eliminates duplicates**
- To retain all duplicates use the corresponding multiset versions **union all**, **intersect all** and **except all**.

- Suppose a tuple occurs *m* times in *r* and *n* times in *s,* then, it occurs:
  - *m + n* times in *r* **union all** *s*
  - min(*m,n)* times in *r* **intersect all** *s*
  - max(0, *m – n)* times in *r* **except all** *s*

Chapter 3: Introduction to SQL

- Overview of The SQL Query Language
- Data Definition
- Basic Query Structure
- Aggregate Functions
- Additional Basic Operations

- **Additional Basic Operations**
- **Set Operations**
- **Nested Subqueries**
- **Null Values**
- **Modification of the Database**

Nested Subqueries

- SQL provides a mechanism for the nesting of subqueries.
- A **subquery** is a **select-from-where** expression that is nested within another query.

Subqueries: set operations

- **Membership tests: in, not in**
- **Compare one element to a set:**
  - **op ALL, op SOME**
  - **op can be = , < , <= , > , >= , <>**
  - **> SOME: "greater than at least one "**
  - **> ALL: "greater than all"**
- set **cardinality**
  - **EXISTS: test for empty relation.**
- Subset containment A ⊇ B
  - **NOT EXISTS (B minus A)**

Example: membership tests

- Find courses offered in Fall 2009 **and** in Spring 2010

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009
        and  course_id
                in (select course_id
                    from section
                    where semester = 'Spring' and year= 2010);
```

- Find courses offered in Fall 2009 **but not** in Spring 2010

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009
        and course_id
            not in (select course_id
                    from section
                    where semester = 'Spring' and year= 2010);
```

Example: tuple membership

- Find the total number of (distinct) students who have taken course sections taught by the instructor with *ID* 10101

```
select count (distinct ID)
from takes
where (course_id, sec_id, semester, year) in
                    (select course_id, sec_id, semester, year
                     from teaches
                     where teaches.ID= 10101);
```

- Note: Above query can be written in a much simpler manner. The formulation above is simply to illustrate SQL features.

Set Comparison

- Find names of instructors with salary greater than that of some (at least one) instructor in the Biology department.

```
select distinct T.name
from instructor T, instructor S
where T.salary > S.salary and S.dept name = 'Biology';
```

- Same query using > **some** clause

```
select name
from instructor
where salary > some
                (select salary
                from instructor
                where dept name = 'Biology');
```

Example Query

- Find the names of all instructors whose salary is greater than the salary of all instructors in the Biology department.

```
select name
from instructor
where salary > all (select salary
                    from instructor
                    where dept name = 'Biology');
```

Definition of all Clause

F <comp> **all** *r*

$\Leftrightarrow$

$\forall\ t \in r\ $ (F <comp> *t)*

Definition of all Clause

- F <comp> **all** *r* $\Leftrightarrow$ $\forall\ t \in r$ (F <comp> *t)*

(5 < **all** | 0 / 5 / 6 |) = false

(5 < **all** | 6 / 10 |) = true

(5 = **all** | 4 / 5 |) = false

(5 ≠ **all** | 4 / 6 |) = true (since 5 ≠ 4 and 5 ≠ 6)

(≠ **all**) ≡ **not in.** However, (= **all**) ≡ **in**

Definition of some Clause

F <comp> **some** *r*

$\Leftrightarrow$

$\exists\ t \in r$ (F <comp> *t)*

Test for Empty Relations

- The **exists** construct returns the value **true** if the argument subquery is nonempty.
- **exists** $r \Leftrightarrow r \neq \emptyset$
- **not exists** $r \Leftrightarrow r = \emptyset$

Correlation Variables

- Yet another way of specifying the query "Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester"

```
select course_id
from section as S
where semester = 'Fall' and year= 2009 and
        exists (select *
                from section as T
                where semester = 'Spring' and year= 2010
                      and S.course_id= T.course_id);
```

- **Correlated subquery**
- **Correlation name** or **correlation variable: S**

Not Exists

- Find all students who have taken all courses offered in the Biology department.

```
select distinct S.ID, S.name
from student  S
where not exists ( (select course_id
                    from course
                    where dept_name = 'Biology')
                   minus
                    (select T.course_id
                     from takes  T
                     where S.ID = T.ID)
                   );
```

  - Note that $X - Y = \emptyset \Leftrightarrow X \subseteq Y$
  - *Note:* Cannot write this query using = **all** and its variants

Exo1 INFOTOUR

- Trouvez le tour d'INFOTOUR le moins cher
  - Deux sous requêtes simples
    - fonction d'aggrégation
    - <= ALL
  - Une sous requête corrélative
    - not exists

Exo: le tour le moins cher (sol 1/3)

```
select t.numtour
from tours t
where t.prix = (select min(prix) from tours)
```

Exo: le tour le moins cher (sol 2/3)

```
select t.numtour
from tours t
where t.prix = (select min(prix) from tours)
```

```
select t.numtour
from tours t
where t.prix <= ALL (select prix from tours)
```

Exo: le tour le moins cher (sol 3/3)

```
select t.numtour
from tours t
where t.prix = (select min(prix) from tours)
```

```
select t.numtour
from tours t
where t.prix <= ALL (select prix from tours)
```

```
select t.numtour
from tours t
where not exists (select t2.numtour from tours t2
   where t2.prix < t.prix)
```

Exo 2 INFOTOUR

- Trouver des ids de directeurs d'hotels, qui ne sont pas des ids de personnes

select directeur
from hotels
where directeur **not in**
(**select** numper **from** personnes)

- (**select** directeur **from** hotels) **minus** (**select** numper **from** personnes)

Test for Absence of Duplicate Tuples

- The **unique** construct tests whether a subquery has any duplicate tuples in its result.
- Not yet widely implemented in DBMS
- Find all courses that were offered at most once in 2009

```
select T.course_id
from course T
where unique (select S.course_id
              from section S
              where T.course_id= S.course_id
                    and S.year = 2009);
```

Exercice 3: rewrite with **count** in subquery!

Subquery in from clause

- SQL allows a subquery expression to be used in the **from** clause
- Not yet supported by all DBMS.
- **Example:** Find the average instructors' salaries of those departments where the average salary is greater than $42,000."

```
select dept_name, avg_salary
from (select dept_name, avg (salary) as avg_salary
        from instructor
        group by dept_name)
where avg_salary > 42000;
```

- The result can also be obtained with having (ex: try!)
- Another way to write above query (**not yet implemented in Oracle**)

```
select dept_name, avg_salary
from (select dept_name, avg (salary)
        from instructor
        group by dept_name) as dept_avg (dept_name, avg_salary)
where avg_salary > 42000;
```

Derived Relations: lateral clause

- Nested subqueries in from clause can not use correlation variables from other relations in the from clause.
- New in SQL:2003 **lateral** clause. Supported in IBM's DB2 system

```
select name, salary, avg_salary
from instructor I1, lateral
                        (select avg(salary) as avg_salary
                    from instructor I2
                    where I2.dept_name= I1.dept_name);
```

With Clause

- Defines a temporary view whose definition is available only to the query in which the **with** clause occurs.
- Introduced in SQL:1999, and supported by many DBMS
- Example: find all departments with the maximum budget

```
with max_budget (value) as
    (select max(budget)
      from department)
select budget
from department, max_budget
where department.budget = max_budget.value;
```

Complex Queries using With Clause

- Find all departments where the total salary is greater than the average of the total salary at all departments

```
with
dept _total (dept_name, value) as
        (select dept_name, sum(salary)
         from instructor
         group by dept_name),
dept_total_avg(value) as
       (select avg(value)
       from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value >= dept_total_avg.value;
```

Scalar Subquery

- returns only one tuple with one attribute
- Can be nested wherever an expression returning a value is permitted
- Example: departments with their numbers of instructors

```
select dept_name,
        (select count(*)
           from instructor
          where department.dept_name = instructor.dept_name)
        as num_instructors
 from department;
```

Chapter 3: Introduction to SQL

- Overview of The SQL Query Language
- Data Definition
- Basic Query Structure
- Aggregate Functions
- Additional Basic Operations

- **Additional Basic Operations**
- **Set Operations**
- **Nested Subqueries**
- **Null Values**
- **Modification of the Database**