Introduction to Relational Databases

- Bachelor Computer Science, Lille 1 University
- Oct 19th, 2011 (lecture 8/12)
- Today's lecturer: C. Kuttler
- Topic: Introduction to SQL as a query language
  - Subqueries with
    - Membership testing
    - Comparison of attribute with set
    - Existential quantification

Subqueries: set operations

- Compare one element to a set:
  - op ALL, op SOME
  - op can be =,<,<=,>,>=,<>
  - >SOME: ' greater than at least one'
  - >ALL: 'greater than all'
- Membership tests: [not] in
- Existence test
  - [NOT] EXISTS: test for existence of a tuple, with certain property

Subqueries: set comparison

- In the **`where`** clause, the some/all compares an attribute (or an expression with attributes) with the result of an SQL query (a set).
- Syntax:

  AttrExpr comp < **`some| all`** > *Subquery*

  - Comparison operator =, <>, <, <=, >, >=
  - *Subquery , or embedded query*
  - **`some`**: returns true *if at least one* line of the table returned by *Subsquery* satisfies the comparison. Synonym: **`any`**
  - **`all`**: returns true if *all* lines of the table returned by the *Subquery* fulfill the comparison

Test in ALL clause: formal definition

F *comp* **all *r***

⇔

$\forall\, t \in r :$ F *comp t*

- Where *comp* can be =,<,<=,>,>=,<>
- In words: the test F *comp* all r evaluates to true, if and only if, for all tuples t of the relation r, the test F *comp* t evaluates to true.

Examples of all Clause

- F *comp* **all** *r* ⇔ $\forall\, t \in r$: F *comp t*

(5 < all $\begin{array}{|c|}\hline 0\\\hline 5\\\hline 6\\\hline\end{array}$) = false

(5 < all $\begin{array}{|c|}\hline 6\\\hline 10\\\hline\end{array}$) = true

(5 = all $\begin{array}{|c|}\hline 4\\\hline 5\\\hline\end{array}$) = false

(5 ≠ all $\begin{array}{|c|}\hline 4\\\hline 6\\\hline\end{array}$) = true (since 5 ≠ 4 and 5 ≠ 6)

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

Test in some clause: definition

F comp **some** *r*

⇔

$\exists\, t \in r$: F *comp t*

- Where *comp* can be =,<,<=,>,>=,<>
- In words: the test **F *comp* some r** evaluates to true, if and only if, for some tuple t of the relation r, the test **F *comp* t** evaluates to true.
- Some: at least one

Examples of comparison with some

- F *comp* **some** *r* ⇔ $\exists\, t \in r$: F *comp t*

(5 < some $\begin{array}{|c|}\hline 0\\\hline 5\\\hline 6\\\hline\end{array}$) = true

(5 > some $\begin{array}{|c|}\hline 6\\\hline 10\\\hline\end{array}$) = false

(5 = some $\begin{array}{|c|}\hline 4\\\hline 5\\\hline\end{array}$) = true

(5 ≠ some $\begin{array}{|c|}\hline 4\\\hline 5\\\hline\end{array}$) = true (since 5 ≠ 4)

(= some) ≡ in

Queries with **some** / **all**

```
select Con_ID
from Contract
where Amount > some
         (select Amount
         from Contract)
```

```
select Con_ID
from Contract
where Amount >= all
        ( select Amount
         from Contract)
```

Con-ID	AMOUNT	SOME	ALL
1	50	F	F
2	300	T	T
3	90	T	F

Set comparison with some

- Extract the contracts for products with a price > 100.

```
 select Con_ID
from Detail
where Prod_ID = some(select Prod_ID
                     from Product
                     where Price > 100)
```

- Equivalent to :

```
select Con_ID
from Detail D, Product P
where D.Prod_ID = P.Prod_ID
  and Price > 100
```

Set comparison with some, 2

- Extract the products sold in contracts in which also the product with identifier ‘ABC’ was sold.
  - With an embedded query:

```
select Prod_ID
from Detail
where Con_ID = some
        (select Con_ID
         from Detail
         where Prod_ID = 'ABC')
```

  - Without sub-query:

```
select D1.Prod_ID
from Detail D1, Detail D2
where D1.Con_ID = D2.Con_ID and
      D2.Prod_ID = 'ABC'
```

Negation with subqueries

- Extract only those contracts that do not contain the product ‘ABC’:

```
select distinct Con_ID
from Contract
where Con_ID <> all (select Con_ID
                     from Detail
                     where Prod_ID = 'ABC')
```

- Alternative:

```
(select Con_ID from Contract)
   except
(select Con_ID from Detail where Prod_ID = 'ABC')
```

Subqueries: in / not in

- Allows for membership testing
- Syntax:

AttrExpr < **`in`** | **`not in`** > *Subquery*

- **`in`**: the predicate is true if *AttrExpr* appears in at least one line returned by *Subquery*
- **`not in`**: the predicate is true if *AttrExpr* does not appear anywhere in the result of the *Subquery*

Formal definition of in clause

F in r

$\Leftrightarrow$

$\mathbf{F} \in r$

- In words: the test **F *in* r** evaluates to true, if and only if, F is contained in the relation r.

Operators **`in`** and **`not in`**

- The operator `in` is equivalent to `= some`

```
select Prod_ID
from Detail
where Con_ID in
        (select Con_ID
         from Detail
         where Prod_ID = 'ABC')
```

- The operator `not in` is equivalent to `<> all`

```
select distinct Con_ID
from Contract
where Con_ID not in (select Con_ID
                     from Detail
                     where Prod_ID = 'ABC')
```

Other example with “in”

- Extract the name of customers who have placed at least one order of an amount of over 10.000

```
select Name, Address
from Customer
where Cus_ID in
         (select Cus_ID
          from Contract
          where Amount > 10000)
```

Embedded queries with multiple levels

- Extract name and address of clients that have signed a contract containing the product “laser”

```
select Name, Address
from Customer
where Cus_ID in
       (select Cus_ID
       from Contract
       where Con_ID in
              (select Con_ID
              from Detail
              where Prod_ID in
                   (select Prod_ID
                   from Product
                   where Name = 'Laser')))
```

Equivalent queries

- The previous query is equivalent to:

```
select C.Name, Address
from Customer as C, Contract as O,
     Detail as D, Product as P
where C.Cus_ID = O.Cus_ID
  and O.Con_ID = D.Con_ID
  and D.Prod_ID = P.Prod_ID
  and P.Name = 'Laser'
```

`max` with embedded queries

- The aggregate functions `max` (and `min)` can be expressed through embedded queries
- Extract the contract with highest amount
  - With an embedded query, using `max`:

```
select Con_ID
from Contract
where Amount in (select max(Amount)
                 from Contract)
```

  - With an embedded query, using >= all:

```
select Con_ID
from Contract
where Amount >= all (select Amount
                     from Contract)
```

exists / not exists operators

- In the where clause, we can use existential quantification on the result of an SQL subquery
- Syntax:

<exists | not exists> *Subquery*

  - **`exists`**: true if the subquery returns something
  - **`not exists`**: true if the subquery doesn't return anything
- In the *Subquery*, it is advisable to always use **`select *`** because projection doesn't matter

Exist clause: definition

select … from T where

exists Subquery

⇔

Subquery ≠ Ø

- **exists** clause returns t**rue** if, and only if, the subquery's result is nonempty.
- The top level query returns those tuples from T for which the Subquery returns something.

Opposite case:

not exists *Subquery* ⇔ *Subquery = Ø*

Correlation variables and existential quantification

- The subquery typically uses a variable of the external query
- Extract all customers who have placed more than one order on the same day:

```
select Cus_ID
from Contract O
where exists (select *
              from Contract O1
              where O1.Cus_ID = O.Cus_ID
                and O1.Date = O.Date
                and O1.Con_ID <> O.Con_ID)
```

Interpretation

-
  ```
  select Cus_ID
  from Contract O
  where exists (select *
                from Contract O1
                where O1.Cus_ID = O.Cus_ID
                  and O1.Date = O.Date
                  and O1.Con_ID <> O.Con_ID)
  ```

- For each tuple O of Contract:
  - The subquery uses its Cus_ID, Date, Con_ID
  - The subquery is evaluated
  - If the subquery’s result isn’t empty, the Cus_ID for this tuple appears in the result of the outer query.

Subquery for emptiness test

- Extract all persons who do [not] have homonyms. :

```
select *
from Person P
where [not] exists
        (select *
         from Person P1
         where P1.Name = P.Name
           and P1.LastName = P.LastName
           and P1.NumSecu <> P.NumSecu)
```

Try yourselves !

- Trouvez l’article de notre boutique le moins cher
  - Deux sous requêtes simples
    - fonction d’aggrégation
    - *comp* ALL
  - Une sous requête corrélative
    - not exists

Our labwork example

1. articles non fournissables
2. couleurs, pour lesquelles un article n'est pas fournissable
3. articles offerts par au moins 2 fournisseurs
4. vendeurs offrant aussi bien des articles rouges que des verts
5. (**) vendeur offrant tous les articles
6. (**) fournisseur offrant tous les articles rouges
7. (**) les monopolistes, avec les articles (noms et aid) concernés.

Equivalence of expressive power

- IN, =ANY, EXISTS have the same expressive power, and can also be expressed through a join (except for duplicates)
- NOT IN, <>ALL, NOT EXISTS have the same expressive power, and can be expressed by a difference
- *comp* SOME, if there are no duplicates, can be rewritten as theta-joins (not as equi-joins)
- *comp* ALL can be rewritten by queries combining grouping and extraction of a minimum and maximum

Tuple construction

- The comparison with the embedded query can involve more than one attribute.
- The attributes must be enclosed by a pair of parentheses (tuple constructor)
- Our previous query can be rewritten as:

```
select *
from Person P
where (Name,LastName) in
        (select Name, LastName
         from Person P1
         where P1.NumSecu <> P.NumSecu)
```

Comments on subqueries

- Embedded queries can be 'less declarative', but are mostly easier to read
- Complex queries with variables can be hard to understand.
- The embedded queries can not contain set operations, mostly (take home lesson: "only do unions on top level"). This limitation is not significant, and not present in all DBMS.

Comments on subqueries

- The use of variables must respect rules of visibility
  - a variable can only be used in the query where it is introduced, or within subqueries embedded therein
  - If a variable name is ambiguous, the system assumes we are referring to the closer one

Visibility of variables

- Incorrect query:

```
select *
from Customer
where Cus_ID in
          (select Cus_ID
           from Contract O1
           where Con_ID = 'AZ1020')
   or Cus_ID in
          (select Cus_ID
           from Contract O2
           where O2.Date = O1.Date)
```

- The query is incorrect, because the variable `O1` is not visible within the second embedded query.

Subqueries in modification commands