Introduction to Relational Databases

- Bachelor Computer Science, Lille 1 University
- Lecture 6/12
-
- Topic: SQL as a query language:
  - Subqueries in the WHERE clause
-

Operators for subqueries

- *Subquery , or nested, or embedded query*

Compare an element to a set:

> SOME: ‘ greater than at least one’

> ALL: ‘greater than all’

Membership tests:

[NOT] IN

Existence test

[NOT] EXISTS: test for existence of a tuple

set comparison: all/some

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

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

  - comparison operators =, <> , < , <= , > , >=
  - **all**: returns true if *all* lines of the table returned by the *Subquery* fulfill the comparison
  - **some**: returns true *if at least one* line of the table returned by *Subquery* satisfies the comparison. Synonym: **any**

ALL: examples

t *comp* **all** ***Rel*** ⇔ ∀ *r* ∈ *Rel* : t *comp r*

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

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

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

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

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

ALL : definition

t comp ALL **Rel**

$\Leftrightarrow$

$\forall\, r \in \text{Rel} : \ t \text{ comp } r$

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

SOME: definition

t comp **some *Rel***

$\Leftrightarrow$

$\exists\, r \in Rel : t\ comp\ r$

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

SOME: examples

t *comp* **some *Rel*** $\Leftrightarrow$ $\exists\, r \in Rel$: t *comp r*

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

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

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

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

(= some) ≡ in

Example: contract management

Customer

Cus_ID	CITY	TAX_ID

Contract

Con_ID	Cus_ID	DATE	VALUE

Detail

Con_ID	Prod_ID	Qt

Product

Prod_ID	NAME	PRICE

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

```
select Con_ID                 select Con_ID
from Contract                 from Contract
where VALUE > some            where VALUE >= all
        (select VALUE                ( select VALUE
        from Contract)                 from Contract)
```

Con_ID	VALUE	>SOME	>=ALL
1	50	F	F
2	300	T	T
3	90	T	F

Set comparison with some

- Extract the contract IDs, for contracts containing at least one product 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 natural join Product
where
       Price > 100
```

Set comparison with some, 2

- Extract the products sold together with the product ‘ABC’ .
- 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 join Detail D2 using (Con_ID)
where
     D2.Prod_ID = 'ABC'
```

Negation with subqueries

- Get contracts that don’t 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: [not] in

- Tests membership of an element in a set
- Syntax:

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

- **`in`**: the predicate is true if *AttrExpr* appears in at least one line returned by the *Subquery*

- **`not in`**: the predicate is true if *AttrExpr* does not appear anywhere in the result of the *Subquery*

IN: definition

t in ***Rel***

⇔

t ∈ *Rel*

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

Equivalences of operators [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 names and addresses of customers with at least one contract of a VALUE over 10.000

```
select Name, Address
from Customer
where Cus_ID in
       (select Cus_ID
            from Contract
           where VALUE > 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 query

- 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

- `max` (and `min)` can be used in embedded queries, or replaced by embedded queries
- Extract the contract with highest VALUE

```
select Con_ID
from Contract
where VALUE in (select max(VALUE)
                  from Contract)
select Con_ID
from Contract
where VALUE >= all (select VALUE
                      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

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

- Subqueries with EXISTS typically use 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 C
where exists (select *
              from Contract C1
              where C1.Cus_ID =  C.Cus_ID
                and C1.Date   =  C.Date
                and C1.Con_ID  <> C.Con_ID)
```

⏎

Interpretation

```
select Cus_ID
from Contract C
where exists (select *
              from Contract C1
              where C1.Cus_ID = C.Cus_ID
                and C1.Date = C.Date
               and C1.Con_ID <> C.Con_ID)
```

For **each** tuple C of Contract:
the subquery is evaluated ,
the subquery uses C.Cus_ID, C.Date, C.Con_ID.
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)
```

Exos

- 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 fournisables
2.L'article le moins cher
3.articles offerts par au moins 2 fournisseurs
4.vendeurs offrant aussi bien des articles rouges que des verts
5.(**) les monopolistes, avec les articles (noms et aid) concernés.
6.(**) fournisseur offrant tous les articles rouges
7.(**) vendeur offrant tous les articles