Introduction to Relational Databases

- Bachelor Computer Science, Lille 1 University
- Oct 21st, 2015 (lecture 8/12)

- Topic: Introduction to SQL
  - Other definitions of data in SQL
    - Views

Views

- Offer the "view" of virtual tables (external schemas)
- Classified into:
  - simple (selection and projection from only one table)
  - complex

- Syntax:

Not in Postgres!

```
create view ViewName [ (AttributeList) ]
    as Subquery
[ with [ local | cascaded ] check option ]
```

Views

- Their definition may contain other views, that were previously defined, but without mutual dependency (recursion was introduced in SQL:1999)

- Can be used to write complex queries
  - Query decomposition
- Are needed to express certain queries
  - Combine and embed several aggregate operations

Example: contract management

Customer

Cus_ID	ADDRESS	TAX_ID

Contract

Con_ID	Cus_ID	DATE	VALUE

Detail

Con_ID	Prod_ID	Qt

Product

Prod_ID	NAME	PRICE

Composition of views and queries

- View creation:

```
create view MainContracts as
    select *
    from Contract
    where VALUE > 10000
```

- Query:

```
select Cus_ID
from MainContracts
```

Composition of both:

```
select Cus_ID
from Contract
where VALUE > 10000
```

Views and queries

- Extract the customer with the highest total bill (without view):

```
select Cus_ID
from Contract
group by Cus_ID
having sum(VALUE) >= all
          (select sum(VALUE)
           from Contract
          group by Cus_ID)
```

- Works with Postgresql, but not accepted by all SQL systems.

Views and queries

- Extract the customer with the highest bill (via view):

```
create view CustomerBill(Cus_ID,TotalBill)
as
   select Cus_ID, sum(VALUE)
   from Contract
   group by Cus_ID;

select Cus_ID
from CustomerBill
where TotalBill = (select max(TotalBill)
                   from CustomerBill);
```

Views and queries

- Extract the average number of contracts per customer:
  - Wrong query (aggregate functions can not be nested):

```
select avg(count(*))
from Contract
group by Cus_ID
```

  - **Correct query (with a view):**

```
create view CustomerStat(Cus_ID,ConNumber) as
select Cus_ID, count(*)
from Contract
group by Cus_ID;

select avg(ConNumber)from CustomerStat;
```

Example of simple view

- Contracts with VALUE over 10.000

```
create view MainContracts as
       select *
       from Contract
       where VALUE > 10000
```

VIEW:
Main contracts

Contract

CON_ID	Cus_ID	DATE	VALUE
1	3	1-6-96	50.000
4	1	1-7-12	12.000
6	3	3-9-12	27.000

Modifications through SIMPLE views

- View:

```
    create view MainContracts as
      select *  from Contract
      where VALUE > 10000
```

- Modification:

```
   update MainContracts
     set VALUE = VALUE * 1.05
     where Cus_ID = '45'
```

Not in Postgres!

- Composition of both:

```
    update Contract
      set VALUE = VALUE * 1.05
      where Cus_ID = '45' and VALUE > 10000
```

Simple views in a cascade

```
   create view Administrators
    (Sid,Name,LastName,Income) as
select Sid, Name, LastName, Income
from Employee
where Department = 'Administration'


   create view JuniorAdministrators as
select *
from Administrators
where Income < 50
with check option
```

Check option: updating views

- The **`check option`** acts when the content of a view is modified.
  - **Pre-condition**: inserted/ updated tuple must be part of the view.
  - **Post-condition**: the tuple must remain in the view
- If the conditions aren't satisfied, the modification is refused.

- **`local:`** control only with respect to *this* view
- **`cascaded:`** the control is recursive.

Not in Postgres!

Check option: example

```
create view MainContracts70 as
select *
from MainContracts
where Cus_ID = 70
with local check option
```

Not in Postgres!

- **Dependencies:**
  - `MainContracts: Contracts with VALUE>10000`
  - `MainContracts70: MainContracts for Cus_ID=70`

Check option

```
update MainContracts70
set Cus_ID = 71
where Con_ID = 754
```

- `is refused with` check option **`local`** and **`cascaded`**

```
update MainContracts70
set VALUE = 5000
where Con_ID = 754
```

Not in Postgres!

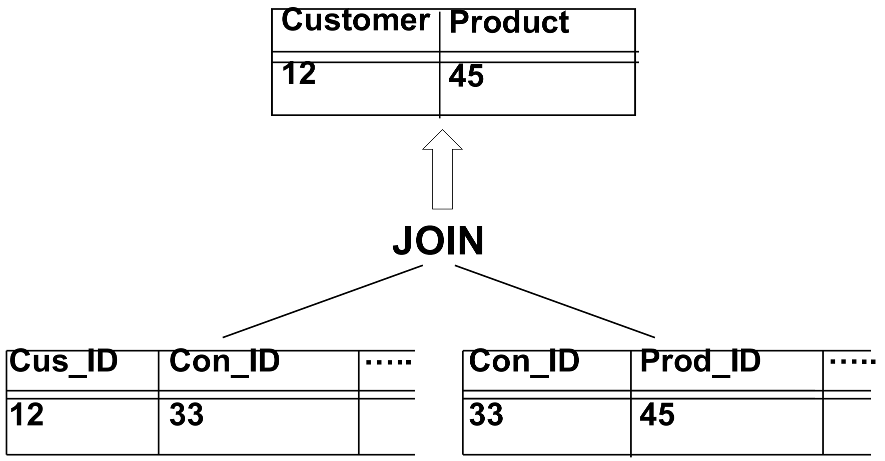
- is accepted with **`local`**, but refused with **`cascaded`**

Complex view

What else is possible, beyond selection and projection?

```
create view CusPro(Customer,Product) as
   select Cus_ID, Prod_ID
   from Contract join Detail
     on Contract.Con_ID = Detail.Con_ID
```

Complex view (JOIN)



Query on complex view

 Query:

```
select Customer
from CusPro
where Product = 45
```

Combining both:

```
select Cus_ID
from Contract join Detail
  on Contract.Con_ID = Detail.Con_ID
where Prod_ID = 45
```

Modifications of the complex view

 It is impossible to modify the original table through the view, because the interpretation is ambiguous:

Ex.:
```
update CusPro
   set Product = 42
    where Customer = 12
```

Not in Postgres!

 Ambiguity for the modification of the original tables
- The customer has changed his contract
- The product’s identifier has changed

Exo 9 du TP

 Le nombre d’articles offerts par le fournisseur avec le plus grand choix, et l’identifiant de ce fournisseur.

```
create view ChoixParFournisseur(fid,nbarticles)
as   select fid, count (*)
      from catalogue  group by fid;

select fid , nbarticles
from ChoixParFournisseur
where nbarticle =
       (select max(nbarticles) from
ChoixParFournisseur);
```