Introduction to Relational Databases

- Bachelor Computer Science, Lille 1 University
- Lecture 5/12
- Topic: Introduction to SQL as a query language, part 2

Today's 3 new clauses

```
select  …
from …
[ where … ]
[group by …]
[having …]
[order by …]
```

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

Example of contracts

Contract

Con_ID	Cus_ID	DATE	VALUE
1	3	1-6-12	50.000.000
2	4	3-8-12	8.000.000
3	3	1-9-12	5.500.000
4	1	1-7-12	12.000.000
5	1	1-8-12	1.500.000
6	3	3-9-12	27.000.000

Order by clause

- appears at the end of a query, orders the lines of the result
- Syntax:

`order by` *OrderAttribute* [`asc` | `desc`]
{, *OrderAttribute* [`asc` | `desc`] }

- The order conditions are applied sequentially :
  - order by first attribute, then by second, etc

Example

```
select  *
from Contract
where VALUE > 1.000.000
order by Date
```

Con_ID	Cus_ID	DATE	VALUE
1	3	1-6-12	50.000.000
4	1	1-7-12	12.000.000
5	1	1-8-12	1.500.000
2	4	3-8-12	8.000.000
3	3	1-9-12	1.500.000
6	3	3-9-12	5.500.000

```
select  *
from Contract
where VALUE > 1.000.000
order by Cus_ID
```

Con_ID	Cus_ID	DATE	VALUE
4	1	1-7-12	12.000.000
5	1	1-8-12	1.500.000
1	3	1-6-12	50.000.000
6	3	3-9-12	5.500.000
3	3	1-9-12	1.500.000
2	4	3-8-12	27.000.000

Several order criteria – sequential application

```
select  *
from Contract
where VALUE > 1.000.00
order by Cus_ID asc, Date desc
```

Con_ID	Cus_ID	DATE	VALUE
5	1	1-8-12	1.500.000
4	1	1-7-12	12.000.000
6	3	3-9-12	5.500.000
3	3	1-9-12	1.500.000
1	3	1-6-12	50.000.000
2	4	3-8-12	27.000.000

Aggregate functions

- Can not be represented in relational algebra
- Aggregate functions evaluate a set of lines
- SQL-2 offers five aggregate functions:

`count`	cardinality
`sum`	summation
`max`	maximum
`min`	minimum
`avg`	average

Count Operator

- returns the number of distinct lines or values;
- Syntax:

  `Count ( < * | [distinct|all]` ***AttributeList*** **>)**

- Extract the number of contracts:

```
select count(*)from Contract
```

- Extract the number of different values of the attribute Cus_ID for all lines of Contract:

```
select count(distinct Cus_ID)from Contract
```

- Extract the number of lines of Contract with Cus_ID not NULL:

```
select count(all Cus_ID)from Contract
```

`sum`, `max`, `min`, `avg`

Syntax:

- < `sum|max|min|avg` > ([`distinct|all`] *AttrExpr*)
- The `distinct` option takes into account each value only once
  - Only useful for `sum` and `avg`
- The option `all` considers all values different from *null*

Maximum query

- **Extract the highest VALUE from all contracts**

```
select max(VALUE) as MaxValue
from Contract
```

MaxValue
50.000.000

Summation query

- **What is the total value of customer 1’s contracts?**

```
select sum(VALUE) as SumAm
from Contract
where Cus_ID = 1
```

SumAm
13.500.000

Aggregate function with join

- Extract the maximal VALUE among those contracts that contain the product ‘ABC’ :

```
select max(VALUE) as MaxValABC
from Contract C join Detail D on
                  C.Con_ID = D.Con_ID
where
     D.Prod_ID = ’ABC’
```

Aggregate functions and target list

Want: the date of the contract with maximal value, and its date. Query **with bug**:

```
select Date, max(VALUE)
from Contract C join Detail D on
       C.Con_ID = D.Con_ID
 where
       D.Prod_ID = ’ABC’
```

The date of which contract, if there are several with same maximal VALUE?

Aggregate functions and target list

Extract the maximal and minimal VALUE of contracts:

```
select max(VALUE) as MaxValue,
       min(VALUE) as MinValue
from Contract
```

MaxValue	MinValue
50.000.000	1.500.000

Query with group selection

In queries, we can use aggregation functions as tests on groups of lines by adding another clause :

- **`group by`** (creation of groups)
- **`having`** (selection of groups)
-

```
select …
from …
where …
group by …
having …
```

Query with grouping

- Extract the sum of values for contracts starting from 10-6-12 for those customers having at least 2 contracts, after that date

```
select Cus_ID, sum(VALUE)
from Contract
where Date >= 10-6-12
group by Cus_ID
having count(*)  >= 2
```

Step 1: evaluate `where Date >...`

Con_ID	Cus_ID	Date	VALUE
2	4	3-8-12	8.000.000
3	3	1-9-12	5.500.000
4	1	1-7-12	12.000.000
5	1	1-8-12	1.500.000
6	3	3-9-12	27.000.000

Step1 eliminates one tuple with `Date < 10-6-12`

Step 2 : grouping

Next, the **`group by`** clause is evaluated

Con_ID	Cus_ID	Date	VALUE
4	1	1-7-12	12.000.000
5	1	1-8-12	1.500.000
3	3	1-9-12	1.500.000
6	3	3-9-12	5.500.000
2	4	3-8-12	8.000.000

Step 3 : computing the aggregates

Now, **`sum(VALUE)`** and **`count(VALUE)`** are calculated, separately for each group

Cus_ID	sum (VALUE)	count (VALUE)
1	13.500.000	2
3	7.000.000	2
4	8.000.000	1

Step 4 : extracting the groups

Next, we evaluate the predicate
having `count(VALUE) >= 2`

Cus_ID	sum (VALUE)	count (VALUE)
1	13.500.000	2
3	7.000.000	2
~~4~~	~~8.000.000~~	~~1~~

Step 5 : producing the result

Cus_ID	Sum (VALUE)
1	13.500.000
3	7.000.000

Avoiding mistakes

- Never write a HAVING without GROUP BY!
- In queries with a group by clause, the select clause (target list) can only contain:
  - grouping attributes
  - aggregate functions
- We *may* add extra grouping attributes for that reason.

Target list for queries with group by

- Query with bug:

```
select VALUE
from Contract
   group by Cus_ID
```

- Query with bug:

```
   select Cu.Cus_ID, Cu.City, count(*)
    from Contract Co join Customer Cu
         on (Co.Cus_ID = Cu.Cus_ID)
     group by Co.Cus_ID
```

- Correct query :

```
    select Co.Cus_ID, Cu.City, count(*)
    from Contract Co join Customer Cu
    on (Co.Cus_ID = Cu.Cus_ID)
     group by Co.Cus_ID, Cu.City
```

http://docs.postgresql.fr/9.0/sql-select.html#sql-groupby

Quand GROUP BY est présent, les expressions du SELECT **ne peuvent faire référence qu'à des colonnes groupées,**

sauf à l'intérieur de fonctions d'agrégat,

la valeur de retour d'une colonne non-groupée n'étant pas unique.

http://docs.postgresql.fr/9.1/sql-select.html#sql-groupby

Quand GROUP BY est présent, les expressions du SELECT ne peuvent faire référence qu'à des colonnes groupées,
sauf à l'intérieur de fonctions d'agrégat,

ou bien si la colonne non groupée **dépend fonctionnellement des colonnes groupées.**

En effet, s'il en était autrement, il y aurait plus d'une valeur possible pour la colonne non groupée. Une dépendance fonctionnelle existe si les colonnes groupées (ou un sous-ensemble de ces dernières) sont la clé **primaire** de la table contenant les colonnes non groupées.

`where` or `having`?

- Only predicates that require the evaluation of aggregate functions may appear in the `having` clause!
-
- Extract the departments where the average incomes of employees working in office 20 exceeds 25:
-

```
select Depart
from Employee
where Office = '20'
group by Depart
having avg(Income) > 25
```

Query with grouping, and ordering

- We can order the result of queries, after grouping

```
select  …
from …
[ where …]
[group by …
[ having … ]]
[order by …]
```

Grouping and order

- Extract the sum of the values of contracts after 10-6-12, for clients with at least two such contracts. Display the result in decreasing order in the sum of the VALUE.

```
select  Cus_ID, sum(VALUE)
from Contract
where Date > 10-6-12
group by Cus_ID
having count(VALUE) >= 2
order by sum(VALUE) desc
```

Result after the order clause

Cus_ID	sum (VALUE)
1	13.500.000
3	7.000.000

Grouping by 2 attributes

- Extract per client, per product, how often this client has bought this product, provided the client has bought over 50 of this product.

```
select Cus_ID, Prod_ID, sum(Qt)
from Contract as C, Detail as D
where  C.Con_ID = D.Con_ID
group by Cus_ID, Prod_ID
having sum(Qt) > 50
```

Situation after group and joining

Contract		Detail			
Cus_ID	Contract. Con_ID	Detail. Con_ID	Prod_ID	Qt	
1	3	3	1	30	group 1,1
1	4	4	1	20	
1	3	3	2	30	group 1,2
1	5	5	2	10	
2	3	3	1	60	group 2,1
3	1	1	1	40	group 3,1
3	2	2	1	30	
3	6	6	1	25	

Extracting the result

- Now, we evaluate the aggregate function **`sum(Qt)`** and the predicate **`having,`** per group.

Cus_ID	Prod_ID	sum(Qt)
1	1	50
1	2	40
2	1	60
3	1	95

Set Queries

Set queries

- Built by combining two SQL queries through set operations
- Syntax:

SelectSQL { < **`union`** | **`intersect`** | **`except`** > [**`all`**]
SelectSQL}

▪ **`union`**	union
▪ **`intersect`**	intersection
▪ **`except`** (**`minus`**)	difference

▪

- Duplicates are eliminated, unless the option **`all`** is used

Union

- Extract the identifiers of those contracts whose value is over 500, or in which over 1000 pieces of some product were bought.

```
select Con_ID
from Contract
where VALUE > 500
      union
select Con_ID
from Detail
where VALUE > 1000
```

Union all

- Repeat the identifier as often as it appears in the tables Contract and Detail.

```
select Con_ID
from Contract
where VALUE > 500
      union all
select Con_ID
from Detail
where VALUE > 1000
```

Difference

- Extract the identifiers of those contracts whose value is over 500, but where no product was bought over 1000 times.

```
select Con_ID
from Contract
where VALUE > 500
      except
select Con_ID
from Detail
where Qt > 1000
```

Intersection

- Extract the identifiers of contracts in which the value is over 500 euro, and in which some product has been sold over 1000 times.

```
select Con_ID
from Contract
where VALUE > 500
      intersect
select Con_ID
from Detail
where Qt > 1000
```