Introduction to Relational Databases

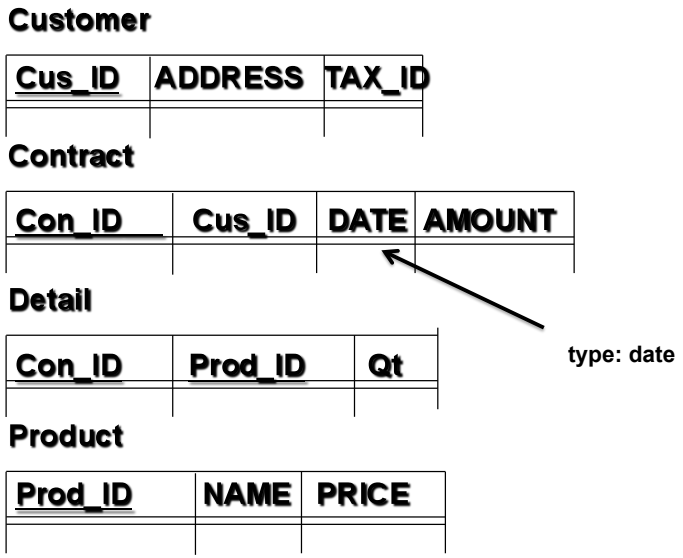
- Bachelor Computer Science, Lille 1 University
- Oct 12th, 2011 (lecture 7/12)
- Today's lecturer: C. Kuttler
- Topic: Introduction to SQL as a query language
  - Order and grouping

Classification of complex queries

- Query with ordering
- Query with aggregation
- Query with grouping
- Binary query (set operation)
- Embedded query

Queries with order and grouping

Example: contract management



Example of contracts

Contract

Con_ID	Cus_ID	DATE	AMOUNT
1	3	1-6-97	50.000.000
2	4	3-8-97	8.000.000
3	3	1-9-97	5.500.000
4	1	1-7-97	12.000.000
5	1	1-8-97	1.500.000
6	3	3-9-97	27.000.000

Ordering

- The `order by` clause, that 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
  - First, order by first attribute, then by second, etc

Query with order clause

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

Con_ID	Cus_ID	DATE	AMOUNT
1	3	1-6-97	50.000.000
4	1	1-7-97	12.000.000
5	1	1-8-97	1.500.000
2	4	3-8-97	8.000.000
3	3	1-9-97	1.500.000
6	3	3-9-97	5.500.000

`order by Cus_ID`

Con_ID	Cus_ID	DATE	AMOUNT
4	1	1-7-97	12.000.000
5	1	1-8-97	1.500.000
1	3	1-6-97	50.000.000
6	3	3-9-97	5.500.000
3	3	1-9-97	1.500.000
2	4	3-8-97	27.000.000

order by Cus_ID asc, Date desc

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

Con_ID	Cus_ID	DATE	AMOUNT
5	1	1-8-97	1.500.000
4	1	1-7-97	12.000.000
6	3	3-9-97	5.500.000
3	3	1-9-97	1.500.000
1	3	1-6-97	50.000.000
2	4	3-8-97	27.000.000

Aggregate functions

- Queries with aggregate functions can not be represented in relational algebra
- The result of a query with aggregate functions depends on the evaluation of the content of a set of lines
- SQL-2 offer five aggregate functions:
  - `count` cardinality
  - `sum` summation
  - `max` maximum
  - `min` minimum
  - `avg` average

Count Operator

- `count` 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 a non NULL value for the attribute Cus_ID:

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


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

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

Query with maximum

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

```
select max(Amount) as MaxAm
from Contract
```

MaxAm
50.000.000

Group by

Query with summation

- **What is the total value of all of customoer 1's contracts?**

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

SumAm
13.500.000

Aggregate function with join

- Extract the maximal contract's amount among those that contain the product with identifier 'ABC' :

```
select max(Amount) as MaxAmABC
from Contract, Detail
where Contract.Con_ID = Detail.Con_ID
      and
      Prod_ID = 'ABC'
```

Aggregate functions and target list

- Query with bug:

```
select Date, max(Amount)
from Contract, Detail
where Contract.Con_ID = Detail.Con_ID
      and
      Prod_ID = 'ABC'
```

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

Aggregate functions and target list

- Extract the maximal and minimal Amount of contracts:

```
select max(Amount) as MaxAm,
       min(Amount) as MinAm
from Contract
```

MaxAm	MinAm
50.000.000	1.500.000

Query with grouping

- In queries, we can use aggregation functions on sub-sets of lines
- By adding the clauses
  - **`group by`** (grouping)
  - **`having`** (group selection)

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


Query with grouping

- Extract the sum of amounts for contracts starting from 10-6-07 for those customers having at least 2 contracts, after that date

```
select Cus_ID, sum(Amount)
from Contract
where Date >= 10-6-07
group by Cus_ID
having count(*)  >= 2
```

Step 1: `where` evaluation

Con_ID	Cus_ID	Date	Amount
2	4	3-8-07	8.000.000
3	3	1-9-07	5.500.000
4	1	1-7-07	12.000.000
5	1	1-8-07	1.500.000
6	3	3-9-07	27.000.000

Eliminate tuples with `Date < 10-6-07`

Step 2 : grouping

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

Con_ID	Cus_ID	Date	Amount
4	1	1-7-07	12.000.000
5	1	1-8-07	1.500.000
3	3	1-9-07	1.500.000
6	3	3-9-07	5.500.000
2	4	3-8-07	8.000.000

Step 3 : computing the aggregates

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

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

Step 4 : extracting the groups

- Now, the having predicate **`count(Amount) >= 2`** is evaluated

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

Step 5 : producing the result

Cus_ID	sum (Amount)
1	13.500.000
3	7.000.000

Query with group by and target list

- Query with bug:

```
select Amount
from Contract
group by Cus_ID
```

- Query with bug:

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

- Correct query :

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


where or **having**?

- Only predicates that require the evaluation of aggregate functions may appear as an argument of the `having` clause!
- Extract the deparments 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 with grouping

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

Grouping and order

- Extract the sum of amounts of contracts after 10-6-07 for clients with at least two contracts, with orders decreasing in the sum of the amount.

```
select  Cus_ID, sum(Amount)
from Contract
where Date > 10-6-07
group by Cus_ID
having count(Amount) >= 2
order by sum(Amount) desc
```


Result after the order clause

Cus_ID	sum (Amount)
1	13.500.000
3	7.000.000

Grouping, 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 O, Detail as D
where  O.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	
3	2	2	1	30	group 3,1
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 Amount > 500
      union
select Con_ID
from Detail
where Amount > 1000
```

Union with all

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

```
select Con_ID
from Contract
where Amount > 500
      union all
select Con_ID
from Detail
where Amount > 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 Amount > 500
      except
select Con_ID
from Detail
where Qt > 1000
```


Difference with all

- Extract the identifiers of contracts that have n>=1 products ordered more than 10 times, and where no more than m>=n products were ordered over 1000 times.

```
select Con_ID
from Detail
where Qt > 10
      except all
select Con_ID
from Detail
where Qt > 1000
```


Intersection

- Extract the idenfiers 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 Amount > 500
      intersect
select Con_ID
from Detail
where Qt > 1000
```