Lecture Notes: Relational Algebra

Det här är ett avsnitt i en webbkurs om databaser som finns fritt tillgänglig på adressen http://www.databasteknik.se/webbkursen/. Senaste ändring: 26 juni 2008.

Sök i webbkursen

Det finns inget kapitel om relationsalgebra i kursen. Jag hade först tänkt ha med ett, men relationsalgebra passar inte riktigt i en grundkurs som den här. I stället finns en kort förklaring i ordlistan, och för den som vill läsa mer finns dessutom dessa föreläsningsanteckningar på engelska.

What? Why?

- Similar to normal algebra (as in `2+3*x-y`), except we use relations as values instead of numbers, and the operations and operators are different.
- Not used as a query language in actual DBMSs. (SQL instead.)
- The inner, lower-level operations of a relational DBMS are, or are similar to, relational algebra operations. We need to know about relational algebra to understand query execution and optimization in a relational DBMS.
- Some advanced SQL queries requires explicit relational algebra operations, most commonly *outer join*.
- Relations are seen as *sets of tuples*, which means that no duplicates are allowed. SQL behaves differently in some cases. Remember the SQL keyword `distinct`.
- SQL is *declarative*, which means that you tell the DBMS *what* you want, but not *how* it is to be calculated. A C++ or Java program is *procedural*, which means that you have to state, step by step, exactly how the result should be calculated. Relational algebra is (more) procedural than SQL. (Actually, relational algebra is mathematical expressions.)

Set operations

Relations in relational algebra are seen as sets of tuples, so we can use basic set operations.

Review of concepts and operations from set theory

- set
- element
- no duplicate elements (but: multiset = bag)
- no order among the elements (but: ordered set)
- subset
- proper subset (with fewer elements)
- superset
- union
- intersection
- set difference
- cartesian product

Projection

Example: The table **E** (for **EMPLOYEE**)

nr	name	salary
1	John	100
5	Sarah	300
7	Tom	100

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr><td><code>select salary
from E</code></td><td><table><tr><th>salary</th></tr><tr><td>100</td></tr><tr><td>300</td></tr></table></td><td>PROJECT$_{salary}$(E)</td></tr>
<tr><td><code>select nr, salary
from E</code></td><td><table><tr><th>nr</th><th>salary</th></tr><tr><td>1</td><td>100</td></tr><tr><td>5</td><td>300</td></tr><tr><td>7</td><td>100</td></tr></table></td><td>PROJECT$_{nr,\ salary}$(E)</td></tr>
</table>

Note that there are no duplicate rows in the result.

Selection

The same table **E** (for **EMPLOYEE**) as above.

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr><td><code>select *
from E
where salary < 200</code></td><td><table><tr><th>nr</th><th>name</th><th>salary</th></tr><tr><td>1</td><td>John</td><td>100</td></tr><tr><td>7</td><td>Tom</td><td>100</td></tr></table></td><td>SELECT$_{salary\ <\ 200}$(E)</td></tr>
<tr><td><code>select *
from E
where salary < 200
and nr >= 7</code></td><td><table><tr><th>nr</th><th>name</th><th>salary</th></tr><tr><td>7</td><td>Tom</td><td>100</td></tr></table></td><td>SELECT$_{salary\ <\ 200\ and\ nr\ >=\ 7}$(E)</td></tr>
</table>

Note that the select operation in relational algebra has nothing to do with the SQL keyword `select`. Selection in relational algebra returns those tuples in a relation that fulfil a condition, while the SQL keyword `select` means "here comes an SQL statement".

Relational algebra expressions

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr><td><code>select name, salary
from E
where salary < 200</code></td><td><table><tr><th>name</th><th>salary</th></tr><tr><td>John</td><td>100</td></tr><tr><td>Tom</td><td>100</td></tr></table></td><td>PROJECT$_{name,\ salary}$ (SELECT$_{salary\ <\ 200}$(E))

<i>or, step by step, using an intermediate result</i>

Temp <- SELECT$_{salary\ <\ 200}$(E)
Result <- PROJECT$_{name,\ salary}$(Temp)</td></tr>
</table>

Notation

The operations have their own symbols. The symbols are hard to write in HTML that works with all browsers, so I'm writing **PROJECT** etc here. The real symbols:

Operation	My HTML	Symbol
Projection	**PROJECT**	π

Operation	My HTML	Symbol
Cartesian product	**X**	$\times$

Selection	**SELECT**	σ
Renaming	**RENAME**	ρ
Union	**UNION**	$\cup$
Intersection	**INTERSECTION**	$\cap$
Assignment	<-	$\leftarrow$

Join	**JOIN**	$\bowtie$
Left outer join	**LEFT OUTER JOIN**	⟕
Right outer join	**RIGHT OUTER JOIN**	⟖
Full outer join	**FULL OUTER JOIN**	⟗
Semijoin	**SEMIJOIN**	$\ltimes$

Example: The relational algebra expression which I would here write as

$\textbf{PROJECT}_{\textbf{Namn}}\ (\ \textbf{SELECT}_{\textbf{Medlemsnummer} < 3}\ (\ \textbf{Medlem}\ )\ )$

should actually be written

$$\pi_{\text{Namn}} \left(\sigma_{\text{Medlemsnummer} < 3} \ (\text{Medlem}) \right)$$

Cartesian product

The *cartesian product* of two tables combines each row in one table with each row in the other table.

Example: The table **E** (for **EMPLOYEE**)

enr	ename	dept
1	Bill	A
2	Sarah	C
3	John	A

Example: The table **D** (for **DEPARTMENT**)

dnr	dname
A	Marketing
B	Sales
C	Legal

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr><td><code>select * from E, D</code></td><td>
<table>
<tr><th>enr</th><th>ename</th><th>dept</th><th>dnr</th><th>dname</th></tr>
<tr><td>1</td><td>Bill</td><td>A</td><td>A</td><td>Marketing</td></tr>
<tr><td>1</td><td>Bill</td><td>A</td><td>B</td><td>Sales</td></tr>
<tr><td>1</td><td>Bill</td><td>A</td><td>C</td><td>Legal</td></tr>
<tr><td>2</td><td>Sarah</td><td>C</td><td>A</td><td>Marketing</td></tr>
<tr><td>2</td><td>Sarah</td><td>C</td><td>B</td><td>Sales</td></tr>
<tr><td>2</td><td>Sarah</td><td>C</td><td>C</td><td>Legal</td></tr>
<tr><td>3</td><td>John</td><td>A</td><td>A</td><td>Marketing</td></tr>
<tr><td>3</td><td>John</td><td>A</td><td>B</td><td>Sales</td></tr>
</table>
</td><td>E X D</td></tr>
</table>

	3	John	A	C	Legal	

- Seldom useful in practice.
- Usually an error.
- Can give a huge result.

Join (sometimes called "inner join")

The cartesian product example above combined each employee with each department. If we only keep those lines where the **dept** attribute for the employee is equal to the **dnr** (the department number) of the department, we get a nice list of the employees, and the department that each employee works for:

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr><td><pre>select *
from E, D
where dept = dnr</pre></td><td>
<table>
<tr><th>enr</th><th>ename</th><th>dept</th><th>dnr</th><th>dname</th></tr>
<tr><td>1</td><td>Bill</td><td>A</td><td>A</td><td>Marketing</td></tr>
<tr><td>2</td><td>Sarah</td><td>C</td><td>C</td><td>Legal</td></tr>
<tr><td>3</td><td>John</td><td>A</td><td>A</td><td>Marketing</td></tr>
</table>
</td><td>

SELECT$_{dept = dnr}$ (E **X** D)

or, using the equivalent join operation

E **JOIN**$_{dept = dnr}$ D

</td></tr>
</table>

- A very common and useful operation.
- Equivalent to a cartesian product followed by a select.
- Inside a relational DBMS, it is usually much more efficient to calculate a join directly, instead of calculating a cartesian product and then throwing away most of the lines.
- Note that the same SQL query can be translated to several different relational algebra expressions, which all give the same result.
- If we assume that these relational algebra expressions are executed, inside a relational DBMS which uses relational algebra operations as its lower-level internal operations, different relational algebra expressions can take very different time (and memory) to execute.

Natural join

A normal inner join, but using the join condition that columns with the same names should be equal. Duplicate columns are removed.

Renaming tables and columns

Example: The table **E** (for **EMPLOYEE**)

nr	name	dept
1	Bill	A
2	Sarah	C
3	John	A

Example: The table **D** (for **DEPARTMENT**)

nr	name
A	Marketing
B	Sales
C	Legal

We want to join these tables, but:

- Several columns in the result will have the same name (**nr** and **name**).
- How do we express the join condition, when there are two columns called **nr**?

Solutions:

- Rename the attributes, using the *rename* operator.
- Keep the names, and prefix them with the table name, as is done in SQL. (This is somewhat unorthodox.)

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr><td>

```
select *
from E as E(enr, ename, dept),
     D as D(dnr, dname)
where dept = dnr
```

</td><td>
<table>
<tr><th>enr</th><th>ename</th><th>dept</th><th>dnr</th><th>dname</th></tr>
<tr><td>1</td><td>Bill</td><td>A</td><td>A</td><td>Marketing</td></tr>
<tr><td>2</td><td>Sarah</td><td>C</td><td>C</td><td>Legal</td></tr>
<tr><td>3</td><td>John</td><td>A</td><td>A</td><td>Marketing</td></tr>
</table>
</td><td>$(\mathbf{RENAME}_{(enr,\ ename,\ dept)}(E))$ $\mathbf{JOIN}_{dept\ =\ dnr}$ $(\mathbf{RENAME}_{(dnr,\ dname)}(D))$</td></tr>
<tr><td>

```
select *
from E, D
where dept = D.nr
```

</td><td>
<table>
<tr><th>nr</th><th>name</th><th>dept</th><th>nr</th><th>name</th></tr>
<tr><td>1</td><td>Bill</td><td>A</td><td>A</td><td>Marketing</td></tr>
<tr><td>2</td><td>Sarah</td><td>C</td><td>C</td><td>Legal</td></tr>
<tr><td>3</td><td>John</td><td>A</td><td>A</td><td>Marketing</td></tr>
</table>
</td><td>$E\ \mathbf{JOIN}_{dept\ =\ D.nr}\ D$</td></tr>
</table>

You can use another variant of the renaming operator to change the name of a table, for example to change the name of **E** to **R**. This is necessary when joining a table with itself (see below).

$\mathbf{RENAME}_{R}(E)$

A third variant lets you rename both the table and the columns:

$\mathbf{RENAME}_{R(enr,\ ename,\ dept)}(E)$

Aggregate functions

Example: The table **E** (for **EMPLOYEE**)

nr	name	salary	dept
1	John	100	A
5	Sarah	300	C
7	Tom	100	A
12	Anne	**null**	C

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr><td>

```
select sum(salary)
from E
```

</td><td>
<table>
<tr><th>sum</th></tr>
<tr><td>500</td></tr>
</table>
</td><td>$F_{sum(salary)}(E)$</td></tr>
</table>

Note:

- Duplicates are not eliminated.
- **Null** values are ignored.

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr><td></td><td>Result:</td><td></td></tr>
</table>

`select count(salary)` `from E`	**count** 3	$F_{count(salary)}(E)$
`select count(distinct salary)` `from E`	Result: **count** 2	$F_{count(salary)}(\textbf{PROJECT}_{salary}(E))$

You can calculate aggregates "grouped by" something:

SQL	Result	Relational algebra

`select sum(salary)`
`from E`
`group by dept`
	<table><tr><th>dept</th><th>sum</th></tr><tr><td>A</td><td>200</td></tr><tr><td>C</td><td>300</td></tr></table>	${}_{dept}F_{sum(salary)}(E)$

Several aggregates simultaneously:

SQL	Result	Relational algebra

`select sum(salary), count(*)`
`from E`
`group by dept`
	<table><tr><th>dept</th><th>sum</th><th>count</th></tr><tr><td>A</td><td>200</td><td>2</td></tr><tr><td>C</td><td>300</td><td>1</td></tr></table>	${}_{dept}F_{sum(salary), count(*)}(E)$

Standard aggregate functions: sum, count, avg, min, max

Hierarchies

Example: The table **E** (for **EMPLOYEE**)

nr	name	mgr
1	Gretchen	**null**
2	Bob	1
5	Anne	2
6	John	2
3	Hulda	1
4	Hjalmar	1
7	Usama	4

Going up in the hierarchy *one level*: What's the name of John's boss?

SQL	Result	Relational algebra

`select b.name`
`from E p, E b`
`where p.mgr = b.nr`
`and p.name = "John"`
	name
	Bob
		$\textbf{PROJECT}_{bname}$ ([$\textbf{SELECT}_{pname = \text{"John"}}$($\textbf{RENAME}_{P(pnr, pname, pmgr)}$(E))] $\textbf{JOIN}_{pmgr = bnr}$ [$\textbf{RENAME}_{B(bnr, bname, bmgr)}$(E)])
		
		or, in a less wide-spread notation
		
		$\textbf{PROJECT}_{b.name}$ ([$\textbf{SELECT}_{name = \text{"John"}}$($\textbf{RENAME}_{P}$(E))] $\textbf{JOIN}_{p.mgr = b.nr}$ [$\textbf{RENAME}_{B}$(E)])
		
		or, step by step

		P <- **RENAME**$_{P(pnr,\ pname,\ pmgr)}$(E) B <- **RENAME**$_{B(bnr,\ bname,\ bmgr)}$(E) J <- **SELECT**$_{name\ =\ "John"}$(P) C <- J **JOIN**$_{pmgr\ =\ bnr}$ B R <- **PROJECT**$_{bname}$(C)

Notes about renaming:

- We are joining **E** with itself, both in the SQL query and in the relational algebra expression: it's like joining two tables with the same name and the same attribute names.
- Therefore, some renaming is required.
- **RENAME**$_P$(E) **JOIN**$_{...}$ **RENAME**$_B$(E) is a start, but then we still have the same attribute names.

Going up in the hierarchy *two levels*: What's the name of John's boss' boss?

SQL	Result	Relational algebra

`select ob.name`
`from E p, E b, E ob`
`where b.mgr = ob.nr`
`where p.mgr = b.nr`
`and p.name = "John"`
	name
	Gretchen
		PROJECT$_{ob.name}$ (([**SELECT**$_{name\ =\ "John"}$(**RENAME**$_P$(E))] **JOIN**$_{p.mgr\ =\ b.nr}$ [**RENAME**$_B$(E)]) **JOIN**$_{b.mgr\ =\ ob.nr}$ [**RENAME**$_{OB}$(E)])
		
		or, step by step
		
		P <- **RENAME**$_{P(pnr,\ pname,\ pmgr)}$(E)
		B <- **RENAME**$_{B(bnr,\ bname,\ bmgr)}$(E)
		OB <- **RENAME**$_{OB(obnr,\ obname,\ obmgr)}$(E)
		J <- **SELECT**$_{name\ =\ "John"}$(P)
		C1 <- J **JOIN**$_{pmgr\ =\ bnr}$ B
		C2 <- C1 **JOIN**$_{bmgr\ =\ bbnr}$ OB
		R <- **PROJECT**$_{obname}$(C2)

Recursive closure

Both one and two levels up: What's the name of John's boss, *and* of John's boss' boss?

SQL	Result	Relational algebra

`(select b.name ...)`
`union`
`(select ob.name ...)`
	name
	Bob
	Gretchen
		(...) **UNION** (...)

Recursively: What's the name of *all* John's bosses? (One, two, three, four or more levels.)

- Not possible in (conventional) relational algebra, but a special operation called **transitive closure** has been proposed.
- Not possible in (standard) SQL (SQL2), but in SQL3, and using SQL + a host language with loops or recursion.

Outer join

Example: The table **E** (for **EMPLOYEE**)

enr	ename	dept
1	Bill	A
2	Sarah	C
3	John	A

Example: The table **D** (for **DEPARTMENT**)

dnr	dname
A	Marketing
B	Sales
C	Legal

List each employee together with the department he or she works at:

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr>
<td><pre>select *
from E, D
where edept = dnr</pre>
or, using an explicit join
<pre>select *
from (E join D on edept = dnr)</pre></td>
<td><table>
<tr><th>enr</th><th>ename</th><th>dept</th><th>dnr</th><th>dname</th></tr>
<tr><td>1</td><td>Bill</td><td>A</td><td>A</td><td>Marketing</td></tr>
<tr><td>2</td><td>Sarah</td><td>C</td><td>C</td><td>Legal</td></tr>
<tr><td>3</td><td>John</td><td>A</td><td>A</td><td>Marketing</td></tr>
</table></td>
<td>$E\ \mathbf{JOIN}_{edept = dnr}\ D$</td>
</tr>
</table>

No employee works at department B, Sales, so it is not present in the result. This is probably not a problem in this case. But what if we want to know the number of employees at each department?

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr>
<td><pre>select dnr, dname, count(*)
from E, D
where edept = dnr
group by dnr, dname</pre>
or, using an explicit join
<pre>select dnr, dname, count(*)
from (E join D on edept = dnr)
group by dnr, dname</pre></td>
<td><table>
<tr><th>dnr</th><th>dname</th><th>count</th></tr>
<tr><td>A</td><td>Marketing</td><td>2</td></tr>
<tr><td>C</td><td>Legal</td><td>1</td></tr>
</table></td>
<td>${}_{dnr, dname}\mathbf{F}_{count(*)}(E\ \mathbf{JOIN}_{edept = dnr}\ D)$</td>
</tr>
</table>

No employee works at department B, Sales, so it is not present in the result. It disappeared already in the join, so the aggregate function never sees it. But what if we want it in the result, with the right number of employees (zero)?

Use a *right outer join*, which keeps all the rows from the right table. If a row can't be connected to any of the rows from the left table according to the join condition, **null** values are used:

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr>
<td><pre>select *
from (E right outer join D on edept = dnr)</pre></td>
<td><table>
<tr><th>enr</th><th>ename</th><th>dept</th><th>dnr</th><th>dname</th></tr>
<tr><td>1</td><td>Bill</td><td>A</td><td>A</td><td>Marketing</td></tr>
<tr><td>2</td><td>Sarah</td><td>C</td><td>C</td><td>Legal</td></tr>
<tr><td>3</td><td>John</td><td>A</td><td>A</td><td>Marketing</td></tr>
</table></td>
<td>$E\ \mathbf{RIGHT\ OUTER\ JOIN}_{edept = dnr}\ D$</td>
</tr>
</table>

<table>
<tr><td></td><td><table><tr><td>null</td><td>null</td><td>null</td><td>B</td><td>Sales</td></tr></table></td><td></td></tr>
<tr><td><pre>select dnr, dname, count(*)
from (E right outer join D on edept = dnr)
group by dnr, dname</pre></td><td><table><tr><th>dnr</th><th>dname</th><th>count</th></tr><tr><td>A</td><td>Marketing</td><td>2</td></tr><tr><td>B</td><td>Sales</td><td>1</td></tr><tr><td>C</td><td>Legal</td><td>1</td></tr></table></td><td>${}_{dnr,\ dname}\mathbf{F}_{count(*)}$ (E RIGHT OUTER JOIN$_{edept\ =\ dnr}$ D)</td></tr>
<tr><td><pre>select dnr, dname, count(enr)
from (E right outer join D on edept = dnr)
group by dnr, dname</pre></td><td><table><tr><th>dnr</th><th>dname</th><th>count</th></tr><tr><td>A</td><td>Marketing</td><td>2</td></tr><tr><td>B</td><td>Sales</td><td>0</td></tr><tr><td>C</td><td>Legal</td><td>1</td></tr></table></td><td>${}_{dnr,\ dname}\mathbf{F}_{count(enr)}$(E RIGHT OUTER JOIN$_{edept\ =\ dnr}$ D)</td></tr>
</table>

Join types:

- **JOIN** = "normal" join = inner join
- **LEFT OUTER JOIN** = left outer join
- **RIGHT OUTER JOIN** = right outer join
- **FULL OUTER JOIN** = full outer join

Outer union

Outer union can be used to calculate the union of two relations that are *partially union compatible*. Not very common.

Example: The table **R**

A	B
1	2
3	4

Example: The table **S**

B	C
4	5
6	7

The result of an outer union between R and S:

A	B	C
1	2	**null**
3	4	**5**
null	6	**7**

Division

Who works on (at least) *all* the projects that Bob works on?

Semijoin

A join where the result only contains the columns from one of the joined tables. Useful in distributed databases, so we don't have to send as much data over the network.

Update

To update a named relation, just give the variable a new value. To add all the rows in relation **N** to the relation **R**:

R <- R **UNION** N
