DATABASTEKNIK - 1DL116

Spring 2004

An introductory course on database systems

http://user.it.uu.se/~udbl/dbt-vt2004/

Kjell Orsborn
Uppsala Database Laboratory
Department of Information Technology, Uppsala University,
Uppsala, Sweden
Introduction to Relational Algebra

Elmasri/Navathe ch 6

Kjell Orsborn

Department of Information Technology
Uppsala University, Uppsala, Sweden
Query languages

• Languages where users can express what information to retrieve from the database.

• Categories of query languages:
  – Procedural
  – Non-procedural (declarative)

• Formal ("pure") languages:
  – Relational algebra
  – Relational calculus
    • Tuple-relational calculus
    • Domain-relational calculus
  – Formal languages form underlying basis of query languages that people use.
Relational algebra

• **Relational algebra** is a procedural language
• Operations in relational algebra takes two or more relations as arguments and return a new relation.
• Relational algebraic operations:
  – Operations from set theory:
    • Union, Intersection, Difference, Cartesian product
  – Operations specifically introduced for the relational data model:
    • Select, Project, Join
• It have been shown that the *select*, *project*, *union*, *difference*, and *cartesian product* operations form a complete set. That is any other relational algebra operation can be expressed in these.
Operations from set theory

• Relations are required to be union compatible to be able to take part in the union, intersection and difference operations.

• Two relations $R_1$ and $R_2$ is said to be union-compatible if:

$$R_1 \sqcup D_1 \times D_2 \times \ldots \times D_n \text{ and } R_2 \sqcup D_1 \times D_2 \times \ldots \times D_n$$

i.e. if they have the same degree and the same domains.
Union operation

- The **union** of two union-compatible relations $R$ and $S$ is the set of all tuples that either occur in $R$, $S$, or in both.
- Notation: $R \cup S$
- Defined as: $R \cup S = \{ t \mid t \in R \text{ or } t \in S \}$
- For example:

```
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
</tr>
</tbody>
</table>
```

\[ R \cup S = \begin{array}{ll}
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
</tr>
</tbody>
</table>
\end{array} \]
Difference operation

- The **difference** between two union-compatible sets $R$ and $S$ is the set of all tuples that occur in $R$ but not in $S$.
- Notation: $R \setminus S$
- Defined as: $R \setminus S = \{ t \mid t \in R \text{ and } t \notin S \}$
- For example:

![Diagram of sets R and S with examples of difference operation]
Intersection

- **The intersection** of two union-compatible sets \( R \) and \( S \), is the set of all tuples that occur in both \( R \) and \( S \).
- Notation: \( R \cap S \)
- Defined as: \( R \cap S = \{ t \mid t \in R \text{ and } t \in S \} \)
- For example:

\[
\begin{array}{c|c}
A & B \\
\hline
a & 1 \\
a & 2 \\
b & 1 \\
\end{array}
\cap
\begin{array}{c|c}
A & B \\
\hline
a & 2 \\
b & 3 \\
\end{array}
= \begin{array}{c|c}
A & B \\
\hline
a & 2 \\
\end{array}
\]
Cartesian product

- Let R and S be relations with k1 and k2 arities resp. The **cartesian product** of R and S is the set of all possible k₁+k₂ tuples where the first k₁ components constitute a tuple in R and the last k₂ components a tuple in S.
- Notation: R □ S
- Defined as: R □ S = \{t q | t \in R and q \in S\}
- Assume that attributes of r(R) and s(S) are disjoint. (i.e. R □ S = \emptyset). If attributes of r(R) and s(S) are not disjoint, then renaming must be used.
Cartesian product example

\[ A \times B = C \times D \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>5</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
</tr>
<tr>
<td>c</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>a</td>
<td>5</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>b</td>
<td>5</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>c</td>
<td>5</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>a</td>
<td>5</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>b</td>
<td>5</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>c</td>
<td>5</td>
</tr>
</tbody>
</table>
Selection operation

• The selection operator, \( \sigma \), selects a specific set of tuples from a relation according to a selection condition (or selection predicate) \( P \).

• Notation: \( \sigma_p(R) \)

• Defined as: \( \sigma_p(R) = \{ t \mid t \in R \text{ and } P(t) \} \) (i.e. the set of tuples \( t \) in \( R \) that fulfills the condition \( P \))

• Where \( P \) is a logical expression (*) consisting of terms connected by: \( \land \) (and), \( \lor \) (or), \( \neg \) (not)
  and each term is one of:
  <attribute> \( op \) <attribute> or <constant>
  where \( op \) is one of: =, ≠, >, ≥, <, ≤.

Example: \( \sigma_{\text{SALARY}>30000}(\text{EMPLOYEE}) \)

(*) a formula in propositional calculus
Selection example

\[ R = \begin{array}{cccc}
A & B & C & D \\
\text{a} & \text{a} & 1 & 7 \\
\text{a} & \text{b} & 5 & 7 \\
\text{b} & \text{b} & 2 & 3 \\
\text{b} & \text{b} & 4 & 9 \\
\end{array} \]

\[ \square A=B \square D > 5 (R) = \begin{array}{cccc}
A & B & C & D \\
\text{a} & \text{a} & 1 & 7 \\
\text{b} & \text{b} & 4 & 9 \\
\end{array} \]
Projection operation

• The projection operator, $\Pi$, picks out (or projects) listed columns from a relation and creates a new relation consisting of these columns.

• Notation: $\Pi_{A_1,A_2,\ldots,A_k}(R)$ where $A_1, A_2$ are attribute names and $R$ is a relation name.

• The result is a new relation of $k$ columns.

• Duplicate rows removed from result, since relations are sets.

Example: $\Pi_{LNAME,FNAME,SALARY}(EMPLOYEE)$
# Projection example

\[ R = \begin{array}{ccc}
A & B & C \\
a & 1 & 1 \\
a & 2 & 1 \\
b & 3 & 1 \\
b & 4 & 2 \\
\end{array} \]

\[ \Pi_{A,C}(R) = \begin{array}{cc}
A & C \\
a & 1 \\
a & 1 \\
b & 1 \\
b & 2 \\
\end{array} = \begin{array}{cc}
A & C \\
a & 1 \\
b & 1 \\
b & 2 \\
\end{array} \]
Join operator

• The **join** operator, (almost), creates a new relation by joining related tuples from two relations.

• Notation: $R \ C \ S$

  $C$ is the join condition which has the form $A_r \square A_s$, where $\square$ is one of $\{=, <, >, \leq, \geq, \neq\}$. Several terms can be connected as $C_1 \square C_2 \square \ldots C_k$.

• A join operation with this kind of general join condition is called “Theta join”.

Example Theta join

\[
\begin{array}{ccc}
A & B & C \\
1 & 2 & 3 \\
6 & 7 & 8 \\
9 & 7 & 8 \\
\end{array}
\quad \quad
\begin{array}{ccc}
B & C & D \\
2 & 3 & 4 \\
7 & 3 & 5 \\
7 & 8 & 9 \\
\end{array}
\quad \quad
\begin{array}{cccc}
A & B & C & D \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 5 \\
1 & 2 & 7 & 8 \\
9 & 7 & 8 & 9 \\
\end{array}
\]

\[ R_{A \leq D} \quad S \]

\[
\begin{array}{cccc}
A & B & C & D \\
1 & 2 & 3 & 2 \\
1 & 2 & 3 & 3 \\
1 & 2 & 7 & 8 \\
9 & 7 & 8 & 9 \\
\end{array}
\]

\[ = \]

\[
\begin{array}{cccc}
A & B & C & D \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 5 \\
1 & 2 & 7 & 8 \\
9 & 7 & 8 & 9 \\
\end{array}
\]

Equijoin

- The same as join but it is required that attribute \( A_r \) and attribute \( A_s \) should have the same value.
- Notation: \( R \ C \ S \)

\( C \) is the join condition which has the form \( A_r = A_s \). Several terms can be connected as \( C_1 \ C_2 \ ... \ C_k \).
Example Equijoin

\[ R \times_{B=C} S = R_{B=C} \times S \]
Natural join

• **Natural join** is equivalent with the application of join to \( R \) and \( S \) with the equality condition \( A_r = A_s \) (i.e. an equijoin) and then removing the redundant column \( A_s \) in the result.

• Notation: \( R \times_{A_r,A_s} S \)

\( A_r,A_s \) are attribute pairs that should fulfil the join condition which has the form \( A_r = A_s \). Several terms can be connected as \( C_1 \sqcap C_2 \sqcap \ldots \sqcap C_k \).
Example Natural join

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>4</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>9</td>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>

\[ R \quad \square_{B=C} \quad S \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>a</td>
<td>4</td>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>
Composition of operations

- Expressions can be built by composing multiple operations
- Example: $\square_{A=C} (R \square S)$

$R \square S = \begin{array}{|c|c|} \hline A & B \\ \hline a & 1 \\ b & 2 \\ \hline \end{array}$

$\square_{A=C} (R \square S) = \begin{array}{|c|c|} \hline A & B & C & D \\ \hline a & 1 & a & 5 \\ a & 1 & b & 5 \\ a & 1 & c & 5 \\ b & 2 & a & 5 \\ b & 2 & b & 5 \\ b & 2 & c & 5 \\ \hline \end{array}$
Additional relational operations

- Assignment and Rename
- Division
- Outer join and outer union
- Aggregate functions (presented together with SQL)
- Update operations (presented together with SQL)
  - (not part of pure query language)
Assignment operation

• The assignment operation (``) makes it possible to assign the result of an expression to a temporary relation variable.

• Example:

  • `temp` `dno = 5` `EMPLOYEE`
  • `result` `fname,lname,salary` `temp`

• The result to the right of the `=` is assigned to the relation variable on the left of the `=`.

• The variable may use variable in subsequent expressions.
Renaming relations and attribute

- The assignment operation can also be used to rename relations and attributes.

- Example:
  \[
  \text{NEWEMP} \leftarrow \text{dno} = 5(\text{EMPLOYEE}) \\
  R(\text{FIRSTNAME, LASTNAME, SALARY}) \leftarrow \Pi_{\text{fname, lname, salary}} (\text{NEWEMP})
  \]
Division operation

- Suited to queries that include the phrase “for all”.
- Let $R$ and $S$ be relations on schemas $R$ and $S$ respectively, where
  $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$
  $S = (B_1, \ldots, B_n)$
- The result of $R \div S$ is a relation on schema $R - S = (A_1, \ldots, A_m)$
  $R \div S = \{ t | t \in R - S \land \exists u \in S. (tu \in R) \}$
Example Division operation

\[
\begin{array}{c|c}
A & B \\
\hline
a & 1 \\
a & 2 \\
a & 3 \\
b & 1 \\
c & 1 \\
d & 3 \\
d & 4 \\
d & 6 \\
e & 1 \\
e & 2 \\
\end{array}
\quad \div \quad
\begin{array}{c|c}
B \\
\hline
1 & 1 \\
2 & 2 \\
\end{array}
\quad = \quad
\begin{array}{c|c}
A \\
\hline
a & a \\
e & e \\
\end{array}
\]
Outer join/union operation

- Extensions of the join/union operations that avoid loss of information.
- Computes the join/union and then adds tuples from one relation that do not match tuples in the other relation to the result of the join.
- Fills out with null values:
  - null signifies that the value is unknown or does not exist.
  - All comparisons involving null are false by definition.
Example Outer join

• Relation *loan*

<table>
<thead>
<tr>
<th>branch-name</th>
<th>loan-number</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>1-170</td>
<td>3000</td>
</tr>
<tr>
<td>Redwood</td>
<td>L-230</td>
<td>4000</td>
</tr>
<tr>
<td>Perryridge</td>
<td>L-260</td>
<td>1700</td>
</tr>
</tbody>
</table>

• Relation *borrower*

<table>
<thead>
<tr>
<th>customer-name</th>
<th>loan-number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>1-170</td>
</tr>
<tr>
<td>Smith</td>
<td>L-230</td>
</tr>
<tr>
<td>Hayes</td>
<td>L-155</td>
</tr>
</tbody>
</table>
Example Outer join cont...

- \textit{loan} * \textit{borrower} (natural join)

<table>
<thead>
<tr>
<th>branch-name</th>
<th>loan-number</th>
<th>amount</th>
<th>customer-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>1-170</td>
<td>3000</td>
<td>Jones</td>
</tr>
<tr>
<td>Redwood</td>
<td>L-230</td>
<td>4000</td>
<td>Smith</td>
</tr>
</tbody>
</table>

- \textit{loan} \textit{left} \textit{borrower} (left outer join)

<table>
<thead>
<tr>
<th>branch-name</th>
<th>loan-number</th>
<th>amount</th>
<th>customer-name</th>
<th>loan-number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>1-170</td>
<td>3000</td>
<td>Jones</td>
<td>1-170</td>
</tr>
<tr>
<td>Redwood</td>
<td>L-230</td>
<td>4000</td>
<td>Smith</td>
<td>L-230</td>
</tr>
<tr>
<td>Perryridge</td>
<td>L-260</td>
<td>1700</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>
Example Outer join cont...

- **loan** right **borrower** (natural right outer join)

<table>
<thead>
<tr>
<th>branch-name</th>
<th>loan-number</th>
<th>amount</th>
<th>customer-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>L-170</td>
<td>3000</td>
<td>Jones</td>
</tr>
<tr>
<td>Redwood</td>
<td>L-230</td>
<td>4000</td>
<td>Smith</td>
</tr>
<tr>
<td>null</td>
<td>L-155</td>
<td>null</td>
<td>Hayes</td>
</tr>
</tbody>
</table>

- **loan** full **borrower** (natural full outer join)

<table>
<thead>
<tr>
<th>branch-name</th>
<th>loan-number</th>
<th>amount</th>
<th>customer-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>L-170</td>
<td>3000</td>
<td>Jones</td>
</tr>
<tr>
<td>Redwood</td>
<td>L-230</td>
<td>4000</td>
<td>Smith</td>
</tr>
<tr>
<td>Perryridge</td>
<td>L-260</td>
<td>1700</td>
<td>null</td>
</tr>
<tr>
<td>null</td>
<td>L-155</td>
<td>null</td>
<td>Hayes</td>
</tr>
</tbody>
</table>
Aggregation operations

- Presented together with SQL later
- Examples of aggregation operations
  - avg
  - min
  - max
  - sum
  - count
Update operations

• Presented together with SQL later
• Operations for database updates are normally part of the DML
  – insert (of new tuples)
  – update (of attribute values)
  – delete (of tuples)
• Can be expressed by means of the assignment operator
Example DB schema

• In the following example we will use a database with the following relation schemas:
  • emps(ename, salary, dept)
  • depts(dname, dept#, mgr)
  • suppliers(sname, addr)
  • items(iname, item#, dept)
  • orders(o#, date, cust)
  • customers(cname, addr, balance)

  • supplies(sname, iname, price)
  • includes(o#, item, quantity)
Relation algebra as a query language

• Relational schema: supplies(sname, iname, price)
• “What is the names of the suppliers that supply cheese?”
  \[ \pi_{\text{sname}}(\pi_{\text{iname}}='\text{CHEESE'}(\text{SUPPLIES})) \]
• “What is the name and price of the items that cost less than 5 $ and that are supplied by WALMART”
  \[ \pi_{\text{iname}, \text{price}}(\pi_{\text{sname}}='\text{WALMART'} \ \land \ \pi_{\text{price}} < 5 \ (\text{SUPPLIES})) \]