QUERY PROCESSING

Relational Algebra

University of Waterloo
List of Slides

1
2 How do we Execute Queries?
3 Relational Algebra
4 Examples
5 Projection
6 Selection
7 Product
8 Union
9 Difference
10 Calculus-to-Algebra Translation
11 Joins
12 Duplicate Operations
13 Example
14 Algebra Equivalences
15 Implementation of the Operators
16 Atomic Relations
17 Joins
18 Duplicates and Aggregates
19 The rest of the lot
20 Summary
How do we Execute Queries?

1. Parsing, typechecking, etc.

2. Relational Calculus (SQL) translated to Relational Algebra

3. Optimization:
   - generates an efficient query plan
   - uses statistics collected about the stored data

4. Plan execution:
   - access methods to access stored relations
   - physical relational operators to combine relations
Relational Algebra

Idea: “queries = functions over a universe of relations”.

\\{	heta : D, \theta \models \varphi \} \text{ is implemented as } F_\varphi(r_1, \ldots, r_k) \\}

- universe \(\mathcal{U}\): finite relations over DOM

- and relational operations:
  \[\Rightarrow\] Projection: \(\pi_V : \mathcal{U} \rightarrow \mathcal{U}\),
  \[\Rightarrow\] Selection: \(\sigma_\varphi : \mathcal{U} \rightarrow \mathcal{U}\)
  \[\Rightarrow\] Product: \(\times : \mathcal{U} \times \mathcal{U} \rightarrow \mathcal{U}\)
  \[\Rightarrow\] Union: \(\cup : \mathcal{U} \times \mathcal{U} \rightarrow \mathcal{U}\)
  \[\Rightarrow\] Difference: \(- : \mathcal{U} \times \mathcal{U} \rightarrow \mathcal{U}\)

operators are easy to implement and can be composed
Algebraic Approach:

- Boolean algebra vs. propositional logic
  \[ (U, \wedge, \vee, 0, 1, ') \]

- Tarski, 1931
  \[ \text{Cylindric Algebra} \]
  \[ (U, \wedge, \vee, 0, 1, ', c_i, d_{i,j}) \]

- Codd, 1970
  \[ \text{Relational Algebra} \]
  \[ \text{matches range-restricted queries} \]
  \[ \text{defined only for finite relations (no top)} \]
Examples

<table>
<thead>
<tr>
<th>Acnt#</th>
<th>Type</th>
<th>Balance</th>
<th>Bank</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>CHK</td>
<td>$1000</td>
<td>TD</td>
<td>1</td>
</tr>
<tr>
<td>1235</td>
<td>SAV</td>
<td>$20000</td>
<td>TD</td>
<td>2</td>
</tr>
<tr>
<td>1236</td>
<td>CHK</td>
<td>$2500</td>
<td>CIBC</td>
<td>1</td>
</tr>
<tr>
<td>1237</td>
<td>CHK</td>
<td>$2500</td>
<td>Royal</td>
<td>5</td>
</tr>
<tr>
<td>2000</td>
<td>BUS</td>
<td>$10000</td>
<td>Royal</td>
<td>5</td>
</tr>
<tr>
<td>2001</td>
<td>BUS</td>
<td>$10000</td>
<td>TD</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD</td>
<td>TD Centre</td>
</tr>
<tr>
<td>CIBC</td>
<td>CIBC Tower</td>
</tr>
</tbody>
</table>
Projection

Definition:

\[ \pi_V(R) = \{(x_{i_1}, \ldots, x_{i_k}) : (x_1, \ldots, x_n) \in R, i_j \in V\} \]

where \( V \) is a set of column \textit{numbers}.

Example:

\[ \pi_{\{\#1, \#2\}}(\text{Account}) = \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>CHK</td>
<td></td>
</tr>
<tr>
<td>1235</td>
<td>SAV</td>
<td></td>
</tr>
<tr>
<td>1236</td>
<td>CHK</td>
<td></td>
</tr>
<tr>
<td>1237</td>
<td>CHK</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>BUS</td>
<td></td>
</tr>
<tr>
<td>2001</td>
<td>BUS</td>
<td></td>
</tr>
</tbody>
</table>
Selection

Definition:

\[ \sigma_\varphi(R) = \{(x_1, \ldots, x_n) : (x_1, \ldots, x_n) \in R, \varphi(x_1, \ldots, x_n) \} \]

where \( \varphi \) is a \textit{built-in} selection condition.

Example:

\[ \sigma_{\#3>5000}(\text{Account}) = \]

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1235</td>
<td>SAV</td>
<td>$20000</td>
<td>TD</td>
<td>2</td>
</tr>
<tr>
<td>2000</td>
<td>BUS</td>
<td>$10000</td>
<td>Royal</td>
<td>5</td>
</tr>
<tr>
<td>2001</td>
<td>BUS</td>
<td>$10000</td>
<td>TD</td>
<td>3</td>
</tr>
</tbody>
</table>
Product

Definition:

\[ R \times S = \{ ((x_1, \ldots, x_n, y_1, \ldots, y_m) : (x_1, \ldots, x_n) \in R, (y_1, \ldots, y_n) \in S \} \]

Example: Account \times Bank =

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>CHK</td>
<td>$1000</td>
<td>TD</td>
<td>1</td>
<td>TD</td>
<td>TD Centre</td>
<td></td>
</tr>
<tr>
<td>1235</td>
<td>SAV</td>
<td>$20000</td>
<td>TD</td>
<td>2</td>
<td>TD</td>
<td>TD Centre</td>
<td></td>
</tr>
<tr>
<td>1236</td>
<td>CHK</td>
<td>$2500</td>
<td>CIBC</td>
<td>1</td>
<td>TD</td>
<td>TD Centre</td>
<td></td>
</tr>
<tr>
<td>1237</td>
<td>CHK</td>
<td>$2500</td>
<td>Royal</td>
<td>5</td>
<td>TD</td>
<td>TD Centre</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>BUS</td>
<td>$10000</td>
<td>Royal</td>
<td>5</td>
<td>TD</td>
<td>TD Centre</td>
<td></td>
</tr>
<tr>
<td>2001</td>
<td>BUS</td>
<td>$10000</td>
<td>TD</td>
<td>3</td>
<td>TD</td>
<td>TD Centre</td>
<td></td>
</tr>
<tr>
<td>1234</td>
<td>CHK</td>
<td>$1000</td>
<td>TD</td>
<td>1</td>
<td>CIBC</td>
<td>CIBC Tower</td>
<td></td>
</tr>
<tr>
<td>1235</td>
<td>SAV</td>
<td>$20000</td>
<td>TD</td>
<td>2</td>
<td>CIBC</td>
<td>CIBC Tower</td>
<td></td>
</tr>
<tr>
<td>1236</td>
<td>CHK</td>
<td>$2500</td>
<td>CIBC</td>
<td>1</td>
<td>CIBC</td>
<td>CIBC Tower</td>
<td></td>
</tr>
<tr>
<td>1237</td>
<td>CHK</td>
<td>$2500</td>
<td>Royal</td>
<td>5</td>
<td>CIBC</td>
<td>CIBC Tower</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>BUS</td>
<td>$10000</td>
<td>Royal</td>
<td>5</td>
<td>CIBC</td>
<td>CIBC Tower</td>
<td></td>
</tr>
<tr>
<td>2001</td>
<td>BUS</td>
<td>$10000</td>
<td>TD</td>
<td>3</td>
<td>CIBC</td>
<td>CIBC Tower</td>
<td></td>
</tr>
</tbody>
</table>
Union

Definition:

\[ R \cup S = \{ (x_1, \ldots, x_n) : (x_1, \ldots, x_n) \in R \]
\[ \quad \lor (x_1, \ldots, x_n) \in S \} \]

Example:

\[ \pi_{#1}(\sigma_{#2='CHK'}(\text{Account})) \cup \pi_{#1}(\sigma_{#2='SAV'}(\text{Account})) = \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>CHK</td>
<td></td>
</tr>
<tr>
<td>1236</td>
<td>CHK</td>
<td></td>
</tr>
<tr>
<td>1237</td>
<td>CHK</td>
<td></td>
</tr>
<tr>
<td>1235</td>
<td>SAV</td>
<td></td>
</tr>
</tbody>
</table>
### Difference

**Definition:**

\[
\{(x_1, \ldots, x_n) : (x_1, \ldots, x_n) \in R, \\
\quad \land (x_1, \ldots, x_n) \notin S\}
\]

**Example:**

*Is there an account without a bank?*

\[
\pi_{\#1,\#4}(\text{Account}) - \pi_{\#1,\#4}(\sigma_{\#4=\#6}(\text{Account} \times \text{Bank})) = \\
\begin{array}{cc}
1237 & \text{Royal} \\
2000 & \text{Royal}
\end{array}
\]
Calculus-to-Algebra Translation

The translation is a simple recursive procedure:

\[
\begin{align*}
\text{Trans}(R(x_{i_1}, \ldots, x_{i_k})) &= R \\
\text{Trans}(Q_1 \land Q_2) &= \text{Trans}(Q_1) \times \text{Trans}(Q_2) \\
\text{Trans}(Q \land x_i = x_j) &= \sigma_{x_i = x_j}(\text{Trans}(Q)) \\
\text{Trans}(\exists x_i.Q) &= \pi_{FV(Q) \setminus \{x_i\}}(\text{Trans}(Q)) \\
\text{Trans}(Q_2 \lor Q_3) &= \text{Trans}(Q_1) \cup \text{Trans}(Q_2) \\
\text{Trans}(Q_2 \land \neg Q_3) &= \text{Trans}(Q_1) \setminus \text{Trans}(Q_2)
\end{align*}
\]

where \( Q_1 \) and \( Q_2 \) have disjoint sets of free variables and \( Q_2 \) and \( Q_3 \) are union compatible.

**Theorem [Codd]:**

For every (safe) relational calculus query there is an equivalent RA expression
Joins

An equality condition after product (common situation):

\[
\sigma_{#4=#6}
\]

we introduce a special \textit{composite} binary operator \textbf{join}:

\[
R \Join_C S = \sigma_C(R \times S)
\]

\(\Rightarrow\) absolutely necessary for performance.
Duplicate Operations

- Projection and duplicate elimination
  \[ \Rightarrow \text{(set) projection } (\pi) \text{ is usually split to} \]
  1. a duplicate preserving projection \((\pi)\)
  2. a duplicate elimination operator \((\varepsilon)\)

- Aggregates, counting, etc.
  \[ \Rightarrow \text{additional operator } \text{Agg}_{f_1, \ldots, f_k}(R) \]
  groups by columns in \(G\)
  applies aggregates \(f_i\) (new columns in result).

- Duplicate versions of standard operators
  \[ \Rightarrow \text{Product, Selection (always preserve duplication)} \]
  \[ \Rightarrow \text{Duplicate preserving Union and Difference} \]
Example

```
SELECT V.Vno, Vname, Count(*), Sum{Amount}
FROM Vendor V, Transaction T
WHERE V.Vno = T.Vno
AND V.Vno Between 1000 and 2000
GROUP BY V.Vno, Vname
HAVING sum(Amount) > 100
```
Algebra Equivalences

1. Selections can be “staged”:
   \[ \sigma_{\varphi_1 \land \varphi_2}(E) = \sigma_{\varphi_1}(\sigma_{\varphi_2}(E)) \]

2. Selections are commutative:
   \[ \sigma_{\varphi_1}(\sigma_{\varphi_2}(E)) = \sigma_{\varphi_2}(\sigma_{\varphi_1}(E)) \]

3. only the last projection counts:
   \[ \pi_V(\pi_U(E)) = \pi_V(E) \]

4. product can be replaced by join:
   \[ \sigma_{\varphi}(R \times S) = R \bowtie_{\varphi} S \]

5. joins are associative (\(\theta_2\) only over columns of \(E_2\)):
   \[ E_1 \bowtie_{\theta_1 \land \theta_3} (E_2 \bowtie_{\theta_2} E_3) = (E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \land \theta_3} E_3 \]

6. for \(\varphi\) involving columns of \(E_1\) only (and vice versa):
   \[ \sigma_{\varphi}(E_1 \bowtie_{\theta} E_2) = \sigma_{\varphi}(E_1) \bowtie_{\theta} E_2 \]

etc, etc, etc.
Implementation of the Operators

- every of the operators of Relational Algebra can be implemented in several ways
  ⇒ not always clear which is the best choice
  ⇒ we implement as many as possible (so we can pick depending on the particular query and database instance).

- the operators are composed using an *iterator protocol*.

- in practice, the operations are often decomposed to more primitive operations (e.g., retrieve a pointer to a record and extract a field from previously retrieved record).
Atomic Relations

We use the **Access Methods** (defined in last lecture) to gain access to the stored data:

- if an index $R_{index}(x)$ (where $x$ is the *search attribute*) is available we replace a subquery of the form
  \[ \sigma_{x=c}(R) \]
  with accessing $R_{index}(x)$ directly,
- Otherwise: check all file blocks holding tuples for $R$.

Even if an index is available, scanning the entire relation may be faster in certain circumstances:

- the relation is very small
- the relation is large, but we expect most of the tuples in the relation to satisfy the selection criteria
Joins

• THE most studied operation of relational algebra; There are many other ways to perform a join.

1. The *Nested Loop Join*

   \[
   \text{for } t \text{ in } R \text{ do for } u \text{ in } S \text{ do}
   \]
   \[
   \quad \text{if } C(t,u) \text{ then output } (tu)
   \]
   \[
   \Rightarrow \text{ with the optional use of indices on } S
   \]

2. The *Sort-Merge Join*

   sort the tuples of $R$ and of $R$ on the common values, then merge the sorted relations.

3. The *Hash Join*

   hash each tuple of $R$ and of $S$ to “buckets” by applying a hash function to columns involved in the join condition. Within each bucket, look for tuples with the matching values.

• the *cost* of the join depends on the chosen method
Duplicates and Aggregates

How do we eliminate duplicates in results of operations? How do we group tuples for aggregation?

Similar solution:

1. sort the result and then eliminate duplicates/aggregate
2. hash the result and do the same

⇒ often an index (e.g., a B+ tree) can be used to avoid the sorting/hashing phase
The rest of the lot

• we assume a natural implementation for selection, duplicate-preserving projection, and duplicate preserving union.

• set difference can be evaluated similarly to a join.

• additional operations:
  ⇒ sorts (used for Sort-Merge Join, Aggregation, and Duplicate Elimination). Uses an external sort algorithm (essentially a merge-sort adopted for disk)
  ⇒ temporary store (to avoid recomputation of subqueries; can be inserted anywhere in the query plan)
  ⇒ . . .
Summary

- Queries are translated to *relational algebra*
  - simple algebraic formalism
  - easy to manipulate
  - lot of equivalences of RA expressions
  - many implementations of basic operators
- a physical plan for a query is a relational algebra expression with choice of implementation for every operator
  - the choices leave room for query optimization