QUERY PROCESSING

Relational Algebra

University of Waterloo

List of Slides

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".

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

- universe $\mathcal{U}$: finite relations over DOM
- and **relational operations:**

⇒ Projection: $\pi_V : \mathcal{U} \to \mathcal{U}$,

⇒ Selection: $\sigma_\varphi : \mathcal{U} \to \mathcal{U}$

⇒ Product: $\times : \mathcal{U} \times \mathcal{U} \to \mathcal{U}$

⇒ Union: $\cup : \mathcal{U} \times \mathcal{U} \to \mathcal{U}$

⇒ Difference: $- : \mathcal{U} \times \mathcal{U} \to \mathcal{U}$

operators are easy to implement and **can be composed**

Algebraic Approach:

- Boolean algebra vs. propositional logic

  $\Rightarrow (U, \wedge, \vee, 0, 1, ')$

- Tarski, 1931

  $\Rightarrow$ Cylindric Algebra
  $\Rightarrow (U, \wedge, \vee, 0, 1, ', c_i, d_{i,j})$

- Codd, 1970

  $\Rightarrow$ *Relational Algebra*
  $\Rightarrow$ matches *range-restricted* queries
  $\Rightarrow$ defined only for *finite* relations (no top)

Examples

Account				
Acnt#	Type	Balance	Bank	Branch
1234	CHK	$1000	TD	1
1235	SAV	$20000	TD	2
1236	CHK	$2500	CIBC	1
1237	CHK	$2500	Royal	5
2000	BUS	$10000	Royal	5
2001	BUS	$10000	TD	3

Bank	
Name	Address
TD	TD Centre
CIBC	CIBC Tower

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 *numbers*.

Example:

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

1234	CHK
1235	SAV
1236	CHK
1237	CHK
2000	BUS
2001	BUS

Selection

Definition:

$$\sigma_\varphi(R) = \{(x_1, \ldots, x_n) : \quad (x_1, \ldots, x_n) \in R, \\ \wedge\, \varphi(x_1, \ldots, x_n)\}$$

where φ is a *built-in* selection condition.

Example:

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

1235	SAV	$20000	TD	2
2000	BUS	$10000	Royal	5
2001	BUS	$10000	TD	3

Product

Definition:

$$
\begin{aligned}
R \times S = \{((x_1, \ldots, x_n, y_1, \ldots, y_m) :\\
(x_1, \ldots, x_n) \in R,\\
(y_1, \ldots, y_n) \in S\}
\end{aligned}
$$

Example: $\mathrm{Account} \times \mathrm{Bank} =$

1234	CHK	$1000	TD	1	TD	TD Centre
1235	SAV	$20000	TD	2	TD	TD Centre
1236	CHK	$2500	CIBC	1	TD	TD Centre
1237	CHK	$2500	Royal	5	TD	TD Centre
2000	BUS	$10000	Royal	5	TD	TD Centre
2001	BUS	$10000	TD	3	TD	TD Centre
1234	CHK	$1000	TD	1	CIBC	CIBC Tower
1235	SAV	$20000	TD	2	CIBC	CIBC Tower
1236	CHK	$2500	CIBC	1	CIBC	CIBC Tower
1237	CHK	$2500	Royal	5	CIBC	CIBC Tower
2000	BUS	$10000	Royal	5	CIBC	CIBC Tower
2001	BUS	$10000	TD	3	CIBC	CIBC Tower

Union

Definition:

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

Example:

$$\pi_{\#1}(\sigma_{\#2='\mathrm{CHK}'}(\mathrm{Account})) \cup \pi_{\#1}(\sigma_{\#2='\mathrm{SAV}'}(\mathrm{Account})) =$$

1234	CHK
1236	CHK
1237	CHK
1235	SAV

Difference

Definition:

$$\{(x_1, \ldots, x_n) : \quad (x_1, \ldots, x_n) \in R, \\ \wedge (x_1, \ldots, x_n) \notin S\}$$

Example:

Is there an account without a bank?

$$\pi_{\#1,\#4}(\text{Account}) - \pi_{\#1,\#4}(\sigma_{\#4=\#6}(\text{Account} \times \text{Bank})) =$$

1237	Royal
2000	Royal

Calculus-to-Algebra Translation

The translation is a simple recursive procedure:

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

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):

1234	CHK	$1000	TD	1	TD	TD Centre
1235	SAV	$20000	TD	2	TD	TD Centre
1236	CHK	$2500	CIBC	1	TD	TD Centre
1237	CHK	$2500	Royal	5	TD	TD Centre
2000	BUS	$10000	Royal	5	TD	TD Centre
2001	BUS	$10000	TD	3	TD	TD Centre
1234	CHK	$1000	TD	1	CIBC	CIBC Tower
1235	SAV	$20000	TD	2	CIBC	CIBC Tower
1236	CHK	$2500	CIBC	1	CIBC	CIBC Tower
1237	CHK	$2500	Royal	5	CIBC	CIBC Tower
2000	BUS	$10000	Royal	5	CIBC	CIBC Tower
2001	BUS	$10000	TD	3	CIBC	CIBC Tower

$$\downarrow \sigma_{\#4=\#6}$$

1234	CHK	$1000	TD	1	TD	TD Centre
1235	SAV	$20000	TD	2	TD	TD Centre
2001	BUS	$10000	TD	3	TD	TD Centre
1236	CHK	$2500	CIBC	1	CIBC	CIBC Tower

we introduce a special *composite* binary operator **join**:

$$R \bowtie_C S = \sigma_C(R \times S)$$

$\Rightarrow$ absolutely necessary for performance.

Duplicate Operations

- Projection and duplicate elimination

  $\Rightarrow$ (set) projection (π) is usually split to

  1. a duplicate preserving projection (π)
  2. a duplicate elimination operator (ϵ)

- Aggregates, counting, etc.

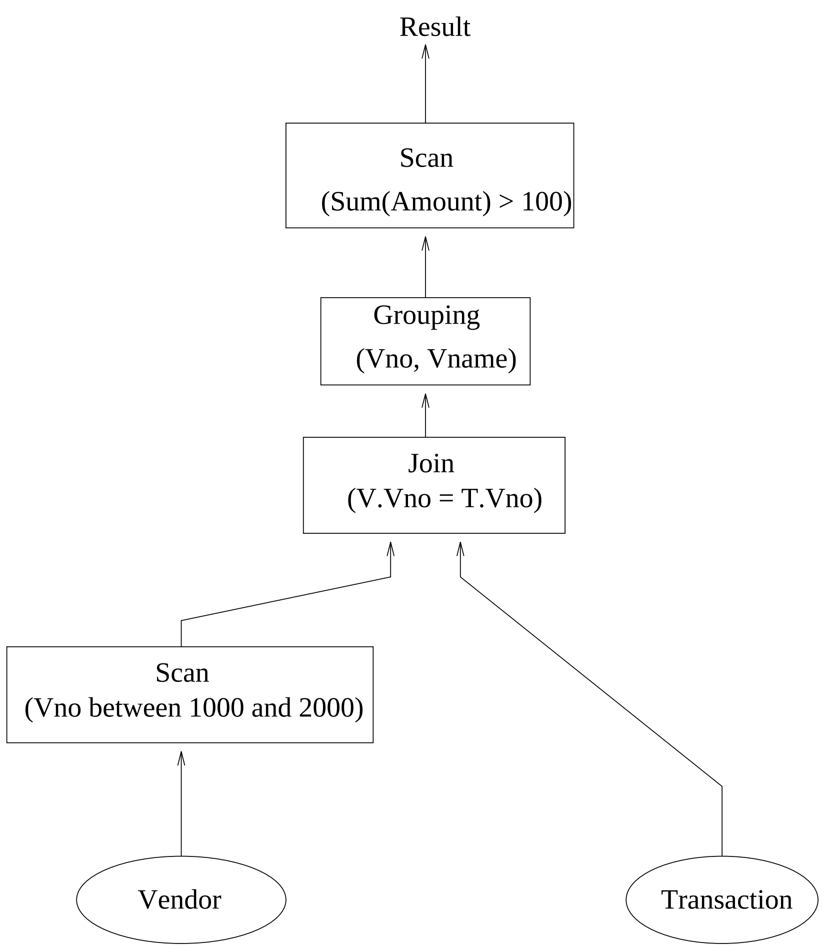
  $\Rightarrow$ additional operator $\mathrm{Agg}_G^{f_1,\ldots,f_k}(R)$
  groups by columns in G
  applies aggregates f_i (new columns in result).

- Duplicate versions of standard operators

  $\Rightarrow$ Product, Selection (always preserve duplication)
  $\Rightarrow$ 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":

$$\sigma_{\varphi_1 \wedge \varphi_2}(E) = \sigma_{\varphi_1}(\sigma_{\varphi_2}(E))$$

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 (θ_2 only over columns of E_2):

$$E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3) = (E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3$$

6. for φ 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
  - $\Rightarrow$ not always clear which is the best choice
  - $\Rightarrow$ 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

     ```
     for t in R  do for u in S do
                  if C(t,u) then output (tu)
     ```

     $\Rightarrow$ 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*
  - $\Rightarrow$ simple algebraic formalism
  - $\Rightarrow$ easy to manipulate
  - $\Rightarrow$ lot of equivalences of RA expressions
  - $\Rightarrow$ many implementations of basic operators
- a physical plan for a query is a relational algebra expression with choice of implementation for every operator

  $\Rightarrow$ the choices leave room for query optimization