Query optimization

Stages:

1. Parsing, validation, view expansion.
2. Logical plan selection (using algebraic laws).
3. Physical plan selection (cost-based)

Textbook: chapter 14

Algebraic laws

Assumption: tuples are mappings from attribute names to domain values.

Join:

$$E_1 \bowtie E_2 = E_2 \bowtie E_1$$

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3).$$

Similarly for Cartesian product.

Cascading:

$$\pi_{A_1,\ldots,A_n}(\pi_{A_1,\ldots,A_n,\ldots,A_{n+k}}(E)) = \pi_{A_1,\ldots,A_n}(E)$$

$$\sigma_{F_1}(\sigma_{F_2}(E)) = \sigma_{F_1 \wedge F_2}(E)$$

Commuting projections

With **selection**:

$$\pi_{A_1,\ldots,A_n}(\sigma_F(E)) = \sigma_F(\pi_{A_1,\ldots,A_n}(E))$$

if F does not involve any attributes othet than $A_1, \ldots, A_n$.

With **Cartesian product**:

$$\pi_{A_1,\ldots,A_n,A_{n+1},\ldots,A_{n+k}}(E_1 \times E_2) =$$
$$\pi_{A_1,\ldots,A_n}(E_1) \times \pi_{A_{n+1},\ldots,A_{n+k}}(E_2)$$

where $A_1, \ldots, A_n$ come from E_1 and $A_{n+1}, \ldots, A_{n+k}$ come from E_2.

With **union**:

$$\pi_{A_1,\ldots,A_n}(E_1 \cup E_2) = \pi_{A_1,\ldots,A_n}(E_1) \cup \pi_{A_1,\ldots,A_n}(E_2).$$

Commuting selections

With **Cartesian product**:

$$\sigma_F(E_1 \times E_2) = \sigma_F(E_1) \times E_2$$

if F involves only the attributes of E_1.

Similarly for E_2.

With **union**:

$$\sigma_F(E_1 \cup E_2) = \sigma_F(E_1) \cup \sigma_F(E_2).$$

With **set difference**:

$$\sigma_F(E_1 - E_2) = \sigma_F(E_1) - \sigma_F(E_2).$$

Cost-based query optimization

Logical query plan:

- expression in relational algebra
- produced by rewrite rules that are obtained from the algebraic laws.

Physical query plan:

- *operators*: different implementations of relational algebra operators
- *table-scan*, *index-scan*, *sort-scan*, *sort*.

A **physical** query plan of **least estimated cost** is produced from a **logical** query plan.

Choices to be made

1. **order** and **grouping** of joins
2. choosing an **algorithm** for each operator occurrence
3. adding other operators like **sorting**
4. materialization vs. pipelining

Assumption: relation $\equiv$ file.

Cost estimation using size estimates

Size estimates should be:

- accurate
- easy to compute
- consistent

We use:

- $B(R)$: number of blocks in relation R
- $T(R)$: number of tuples in relation R
- $V(R, A)$: number of distinct values R has in attribute A (can be generalized to *sets* of attributes)

Results of RA operators

Projection:

- $T(\pi_X(R)) = T(R)$ on how much is projected out

Selection:

- $T(\sigma_{A=c}(R)) = T(R)/V(R, A)$
- $T(\sigma_{A<c}(R) = T(R)/3$
- $T(\sigma_{C_1 \wedge C_2}(R) = T(\sigma_{C_1}(\sigma_{C_2}(R)))$
- $T(\sigma_{C_1 \vee C_2}(R)) = \min(T(R), T(\sigma_{C_1}(R)) + T(\sigma_{C_2}(R)))$

Those estimates may be *inconsistent*.

Natural join $R(X, Y) \bowtie S(Y, Z)$

Assumptions:

- *Containment*: $V(R, Y) < V(S, Y)$ implies $\pi_Y(R) \subseteq \pi_Y(S)$
- *Preservation*: $V(R \bowtie S, A) = V(R, A)$.

Case 1: Y consists of one attribute A:

$$T(R \bowtie S) = \frac{T(R)T(S)}{\max(V(R, A), V(S, A))}.$$

Case 2: Y consists of n attributes $A_1, \ldots, A_n$:

$$T(R \bowtie S) = \frac{T(R)T(S)}{\prod_{j=1,\ldots,n} \max(V(R, A_j), V(S, A_j))}.$$

Multiway join $S = R_1 \bowtie \cdots \bowtie R_k$

Assume $A_1, \ldots A_n$ are the attributes that appear in more than one relation.

The A_j**-factor** $M(A_j)$ for an attribute A_j is the product of $V(R_i, A_j)$ for every relation R_i in which A_j appears, except for the smallest one.

$$T(R_1 \bowtie \cdots \bowtie R_k) = \frac{\prod_{i=1,\ldots,k} T(R_i)}{\prod_{j=1,\ldots,n} M(A_j)}.$$

Better estimates exist, for example based on **histograms**.

Enumerating physical plans for $R_1 \bowtie \cdots \bowtie R_k$

Bottom-up approach, based on **dynamic programming**.

Assumptions (usual):

- **left-deep** trees: more efficient use of space
- **cost = size of intermediate results**

Constructing a table *D(Size,LeastCost,BestPlan)*:

- Base cases:
  - One relation R: $D(T(R), 0, R)$
  - One join $R_i \bowtie R_j$: $D(T(R_i \bowtie R_j), 0, R_i \bowtie R_j)$ if $T(R_i) \leq T(R_j)$
- Inductive case $\underset{\mathcal{R}}{\bowtie}$:
  1. $\mathcal{R} = \{R_{i_1}, \ldots, R_{i_m}\}$ for $m < k$
  2. for every $j \in \{i_1, \ldots, i_m\}$, retrieve the tuple $D(T(\underset{i \neq j, R_i \in \mathcal{R}}{\bowtie} R_i), C_j, E_j)$
  3. find $p \in \{i_1, \ldots, i_m\}$ such that $C_p + T((\underset{i \neq p, R_i \in \mathcal{R}}{\bowtie} R_i))$ is minimal
  4. return

     $$D(T((\underset{i \neq p, R_i \in \mathcal{R}}{\bowtie} R_i) \bowtie R_p), C_p + T(\underset{i \neq p, R_i \in \mathcal{R}}{\bowtie} R_i), (\underset{i \neq p, R_i \in \mathcal{R}}{\bowtie} R_i) \bowtie R_p)$$

How many plans are considered?

Refinements

More sophisticated **cost measure**:

- calculating the cost of using different algorithms for the same operation
- keeping multiple plans, together with their costs, to account for *interesting* join orders.

Greedy algorithm:

1. start with a pair of relations whose estimated join size is minimal
2. keep adding further relations with minimal estimated join size

Completing the evaluation plan

Operations:

- `TableScan(Relation)`
- `SortScan(Relation, Attributes)`
- `IndexScan(Relation, Condition)`
- `Filter(Condition)`
- `Sort(Attributes)`
- different algorithms for the basic operations

Pipelining vs. materialization.