Query Processing

SQL query

Parser

Initial logical query plan (algebraic expression)

Logical Plan Generator/Chooser

Candidate logical query plan

Optimization

Physical Plan Generator/Chooser

"Optimal" physical query plan

• algebraic expression
• implementation details
  • algorithm for operators
  • data management (pipelined, in memory, disk etc.)

Query Processor

• Basic task: turn user queries (and updates) into a sequence of operations (a plan) on the database
  =>$ from high level queries to low level commands

• Many choices:
  – Which of the algebraically equivalent forms of a query will lead to the most efficient algorithm?
  – For each algebraic operator which algorithm should we use to run the operator?
  – How should the operations pass data from one to the other? (eg, main memory buffers, disk buffers)

The Journey of a Query (Revisited)

SELECT Theater
FROM Movie, Schedule
WHERE Movie.Title=Schedule.Title
AND Actor="Winger"

π Movie Schedule
σ Movie.Title=Schedule.Title
σ Actor="Winger"

Logical Plan Generator
Applies Algebraic Rewrites

yet another logical query plan

JOIN

Another logical query plan

Logical Plan Generator

π Theater
σ Actor="Winger"
σ M.Title=S.Title

Optimization

Physical Plan Generator/Chooser

"Optimal" physical query plan

• algebraic expression
• implementation details
  • algorithm for operators
  • data management (pipelined, in memory, disk etc.)
4th logical query plan

- 4 logical query plans created
- algebraic rewritings were used to produce candidate logical query plans
- the last one wins
- multiple logical plans may "win" eventually (and be considered subsequently)

The Journey Continues at the Physical Plan Generator

Physical Plan Generators choose execution primitives and data passing

- More than one plan may be generated by choosing different primitives

More Than One Plan May be Generated and Considered for Evaluation
**Issues in Query Processing and Optimization**

- **Generate Plans**
  - systematically transform expressions
  - employ execution primitives for computing relational algebra operations
- **Estimate Cost of Generated Plans**
  - Statistics
  - “Smart” Search of the Space of Possible Plans
  - always do the “good” transformations (relational algebra optimization)
  - prune the space (e.g., System R)
- Often the above steps are mixed

---

**Bag Semantics of Operators**

- SQL semantics for collections of tuples:
  - often bags (aka multisets) instead of sets.
  - \( \text{Union}(R,S) \): a tuple \( t \) occurs in the result as many times as the sum of occurrences in each of \( R \) and \( S \)
  - \( \text{Intersection of } R \text{ and } S \): ... the minimum number of occurrences in \( R \) and \( S \)
  - \( \text{Difference of } R \text{ and } S \): ... occurrences in \( R \) minus occurrences in \( S \)
  - \( \delta(R) \) converts the bag \( R \) into a set
    - SQL’s \( R \cup S \) by default \( \delta(R \cup S) \)
- **Example:** \( R = \{A,B,B\} \) and \( S = \{C,A,B,C\} \).

---

**Extended Algebra Operators**

- **selection** \( \sigma_C(R) \): condition \( C \) over arithmetic-, string-, and comparison-operators
  - WHERE in SQL, but \( C \) depends only on individual tuples
- **projection** \( \pi_A(R) \): attribute list \( A \) may include
  - \( x \rightarrow y \) (rename \( x \) to \( y \) in the output schema)
  - \( a+b \rightarrow x \) (output schema contains \( x \) column with sum of \( a \) and \( b \) )
  - \( c|d \rightarrow y \) (concatenate \( c \) and \( d \) as \( y \) )
  - again: look at one tuple at a time
- alternatively: special purpose operators:
  - \( \text{MULT}_{a,b\rightarrow c}(R) \):
    - for each tuple of \( R \), multiply attribute \( A \) with attribute \( B \) and put the result in a new attribute named \( C \).
  - \( \text{PLUS}_{a,b\rightarrow c}(R), \text{CONCAT}_{a,b\rightarrow c}(R), \ldots \)
Product and Joins

- **Product** \((R \times S)\):
  - \(r \in n\) times in \(R\), \(s \in m\) times in \(S\), then \((r,s)\) is \(nm\) times in the product
  - (qualify attributes if necessary: \(R.a\) vs \(S.a\))

- **Natural Join** \(R \bowtie S = \pi_{\alpha} \sigma_C(R \times S)\) where
  - \(C\) equates all common attributes
  - \(A\) has all attributes from \(R\) and \(S\) (but only one copy per equated attribute)
  - (definition of join - simple rewriting rule)

- **Theta Join**
  - arbitrary condition involving attributes

Grouping and Aggregation

- **Grouping & Aggregation** in SQL:

  ```sql
  SELECT select-list FROM from-list WHERE...
  GROUP BY groupby-list
  HAVING group-qualification
  ```

- **Example**: given `movie(title, year, actor)` find for each actor having more than two movies, the first year of appearance

  ```sql
  SELECT actor, MIN(year) as firstYear FROM movie
  GROUP BY actor
  HAVING COUNT(title) > 2
  ```

  - Every attribute in the `select-list` must appear in the `groupby-list` (Why?)

  - `GROUP BY actor, MIN(year) as firstYear, COUNT(title) > 2 (movie)`

Algebraic Rewritings: Commutativity and Associativity

- **Cartesian Product**

- **Natural Join**

  - Do the above hold for both sets and bags?
  - Do commutativity and associativity hold for arbitrary Theta Joins?
Algebraic Rewritings: Commutativity and Associativity

<table>
<thead>
<tr>
<th>Union</th>
<th>Commutativity</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R S</td>
<td>R S</td>
</tr>
<tr>
<td></td>
<td>S R</td>
<td>S R</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Intersection</th>
<th>Commutativity</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>R S</td>
<td>R S</td>
<td>R S</td>
</tr>
<tr>
<td>S R</td>
<td>S R</td>
<td>S R</td>
</tr>
</tbody>
</table>

Commutativity: $R \cup S = S \cup R$
Associativity: $(R \cup S) \cup T = R \cup (S \cup T)$