Query Processing
Connolly & Begg Chapter 19

Introduction
- In network and hierarchical DBMSs, low-level procedural query language is generally embedded in high-level programming language.
- Programmer’s responsibility to select most appropriate execution strategy.
- With declarative languages such as SQL, user specifies what data is required rather than how it is to be retrieved.
- Relieves user of knowing what constitutes good execution strategy.
- Also gives DBMS more control over system performance.
- Two main techniques for query optimization:
  - heuristic rules that order operations in a query.
  - comparing different strategies based on relative costs, and selecting one that minimizes resource usage.
- Disk access tends to be dominant cost in query processing for centralized DBMS.

Query Processing
- Activities involved in retrieving data from the database. Aims of QP:
  - transform query written in high-level language (e.g. SQL), into correct and efficient execution strategy expressed in low-level language (implementing RA);
  - execute the strategy to retrieve required data.
- Four main Phases: decomposition, optimization, code generation and execution.

Query Optimization
Activity of choosing an efficient execution strategy for processing query.
- As there are many equivalent transformations of same high-level query, aim of QO is to choose one that minimizes resource usage.
- Generally, reduce total execution time of query.
- May also reduce response time of query.
- Problem computationally intractable with large number of relations, so strategy adopted is reduced to finding near optimum solution.

Dynamic versus Static Optimization
- Two choices of decomposing and optimizing, when first three phases of QP can be carried out:
  - dynamically every time query is run.
  - Statically when query is parsed, validated and optimized once after submission.
- Advantages of dynamic QO arise from fact that information required to select an optimum strategy is up-to-date.
- Disadvantages are that performance of query is affected, time may limit finding optimum strategy.
- Advantages of static QO are removal of runtime overhead, and more time to find optimum strategy.
- Disadvantages arise from fact that chosen execution strategy may no longer be optimal when query is run.
- Could use a hybrid approach to overcome this.
Example 18.1 - Different Strategies

Find all Managers that work at a London branch.

SELECT Sno, Fname, Position, Bno
FROM staff s, branch b
WHERE s.bno = b.bno AND
(s.position = 'Manager' AND b.city = 'London');

Three equivalent RA queries are:

(1) \(\sigma\) (position='Manager') \(\land\) (city='London') \(\land\) (staff.bno=branch.bno) (Staff \(\times\) Branch)

(2) \(\sigma\) (position='Manager') \(\land\) (city='London') (Staff \(\bowtie\) staff.bno=branch.bno Branch)

(3) (\(\sigma\) position='Manager'(Staff)) \(\bowtie\) staff.bno=branch.bno (\(\sigma\) city='London' (Branch))

Assumptions:

- 1000 tuples in Staff; 50 tuples in Branch;
- 50 Managers; 5 London branches;
- No indexes or sort keys;
- Results of any intermediate operations stored on disk;
- Cost of the final write is ignored;
- Tuples are accessed one at a time.

Cost (in disk accesses) are:

(1) \((1000 + 50) + 2*(1000 \times 50) = 101 050\)

(2) \(2*1000 + (1000 + 50) = 3 050\)

(3) \(1000 + 2*50 + 5 + (50 + 5) = 1 160\)

Cartesian product and join operations are much more expensive than selection, and third option significantly reduces size of relations being joined together.
\( \Pi_{\text{Sno}, \text{Fname}, \text{Position}, \text{Bno}} \)

\[ \sigma (\text{position} = '\text{Manager}') \land (\text{city} = '\text{London}') \land (\text{staff.bno} = \text{branch.bno}) \]

RELATIONAL ALGEBRA TREE

\( \Pi_{\text{Sno}, \text{Fname}, \text{Position}, \text{Bno}} \)

\[ \sigma (\text{position} = '\text{Manager}') \land (\text{city} = '\text{London}') \]

\( \bowtie \text{staff.bno} = \text{branch.bno} \)

Staff \hspace{1cm} Branch
Phases of query Processing

QP has four main phases:
- decomposition (consisting of parsing and validation)
- optimization
- code generation
- execution.

Query Decomposition
Aims are to transform high-level query into RA query and check that query is syntactically and semantically correct.
- Typical stages are:
  - analysis,
  - normalization,
  - semantic analysis,
  - simplification,
  - query restructuring.

Analysis
- Analyze query lexically and syntactically using compiler techniques.
- Verify relations and attributes exist.
- Verify operations are appropriate for object type.

Example:
SELECT staff_no
FROM staff
WHERE position > 10;

- This query would be rejected on two grounds:
  - Staff.No is not defined for Staff relation (should be Sno).
  - Comparison ‘>10’ is incompatible with type Position, which is variable character string.

Finally, query transformed into some internal representation more suitable for processing.
- Some kind of query tree is typically chosen, constructed as follows:
  - Leaf node created for each base relation.
  - Non-leaf node created for each intermediate relation produced by RA operation.
  - Root of tree represents query result.
  - Sequence is directed from leaves to root.

Example 18.1 - R.A.T.
Normalization

- Converts query into a normalized form for easier manipulation.
- Predicate can be converted into one of two forms:
  - Conjunctive normal form:
    \[(\text{position} = 'Manager' \lor \text{salary} > 20000) \land (\text{bno} = 'B3')\]
  - Disjunctive normal form:
    \[(\text{position} = 'Manager' \land \text{bno} = 'B3') \lor (\text{salary} > 20000 \land \text{bno} = 'B3')\]

Semantic Analysis

- Rejects normalized queries that are incorrectly formulated or contradictory.
- Query is incorrectly formulated if components do not contribute to generation of result.
- Query is contradictory if its predicate cannot be satisfied by any tuple.
- Algorithms to determine correctness exist only for queries that do not contain disjunction and negation.
- For these queries, could construct:
  - Construct a relation connection graph.
  - Normalized attribute connection graph.

Relation connection graph
Create node for each relation and node for result. Create edges between two nodes that represent a join, and edges between nodes that represent projection. If not connected, query is incorrectly formulated.

Semantic Analysis - Normalized Attribute Connection Graph

- Create node for each reference to an attribute, or constant 0.
- Create directed edge between nodes that represent a join, and directed edge between attribute node and 0 node that represents selection.
- Weight edges \(a \rightarrow b\) with value \(c\), if it represents inequality condition \((a \leq b + c)\); weight edges \(0 \rightarrow a\) with \(-c\), if it represents inequality condition \((a \geq c)\).
- If graph has cycle for which valuation sum is negative, query is contradictory.
Example 18.2 - Checking Semantic Correctness

SELECT p.pno, p.street
FROM renter r, viewing v, property_for_rent p
WHERE r.rno = v.rno AND
  r.max_rent >= 500 AND
  r.pref_type = 'Flat' AND p.ono = 'CO93';

Relation Connection graph

Normalized attribute connection graph

- Create node for each reference to an attribute, or constant 0.
- Create directed edge between nodes that represent a join, and directed edge between attribute node and 0 node that represents selection.
- Weight edges $a \rightarrow b$ with value $c$, if it represents inequality condition $(a \leq b + c)$;
- Weight edges $0 \rightarrow a$ with $-c$, if it represents inequality condition $(a \geq c)$.
- If graph has cycle for which valuation sum is negative, query is contradictory.
- Normalized attribute connection graph has cycle between nodes R.Max_Rent and 0 with negative valuation sum, so query is contradictory.

Simplification
- Detects redundant qualifications,
- Eliminates common sub-expressions,
- Transforms query to semantically equivalent but more easily and efficiently computed form.

- Typically, access restrictions, view definitions, and integrity constraints are considered.
- Assuming user has appropriate access privileges, first apply well-known idempotency rules of boolean algebra.
Example:

```
CREATE VIEW staff3
AS SELECT sno, fname, lname, salary
FROM Staff
WHERE bno = 'B3';
```

```
SELECT *
FROM staff3
WHERE (bno = 'B3' AND salary > 20000);
```

```
SELECT sno, fname, lname, salary
FROM Staff
WHERE (bno = 'B3' AND salary > 20000) AND bno = 'B3';
```

```
WHERE (bno = 'B3' AND salary > 20000);
```

**Integrity Constraints to simplify queries:**

```
CREATE ASSERTION only_manager_salary_high
CHECK ((position <> "Manager" AND salary <2000)
OR position = "Manager" AND salary > 20000));
```

```
SELECT *
FROM staff
WHERE (position = 'Manager' AND salary < 15000);
```

Predicate in the WHERE clause is a contradiction of the constraint

There can be no tuples that satisfies this predicate.
Transformation Rules for RA Operations

- Conjunctive selection operations can cascade into individual selection operations (and vice versa). Sometimes referred to as cascade of selection.

\[ \sigma p \land q \land r(R) = \sigma p(\sigma q(\sigma r(R))) \]

Example:

\[ \sigma \text{bno}=\text{B3} \land \text{salary}>15000(\text{Staff}) = \sigma \text{bno}=\text{B3}(\sigma \text{salary}>15000(\text{Staff})) \]

- Commutativity of selection.

\[ \sigma p(\sigma q(R)) = \sigma q(\sigma p(R)) \]

Example:

\[ \sigma \text{bno}=\text{B3}(\sigma \text{salary}>15000(\text{Staff})) = \sigma \text{salary}>15000(\sigma \text{bno}=\text{B3}(\text{Staff})) \]

- In a sequence of projection operations, only the last in the sequence is required.

\[ \Pi L \Pi M \ldots \Pi N (R) = \Pi L (R) \]

Example:

\[ \Pi \text{name} \Pi \text{bno}(\text{Staff}) = \Pi \text{name} (\text{Staff}) \]

- Commutativity of selection and projection.
  - If predicate p involves only attributes in projection list, selection and projection operations commute:

\[ \Pi \text{Ai}, \ldots, \text{Am}(\sigma p(R)) = \sigma p(\Pi \text{Ai}, \ldots, \text{Am}(R)) \quad \text{where } p \in \{ \text{A}_1, \text{A}_2, \ldots, \text{A}_m \} \]

Example:

\[ \Pi \text{name}=\text{Beech}(\sigma \text{name}=\text{Beech}(\Pi \text{name}(\text{Staff}))) = \sigma \text{name}=\text{Beech}(\Pi \text{name}(\text{Staff})) \]

- Commutativity of theta-join (and Cartesian product).

\[ R \bowtie p S = S \bowtie p R \]

\[ R \times S = S \times R \]

  - Rule also applies to equi-join and natural join.

  Example: \( \text{Staff} \bowtie \text{staff.bno=branch.bno} \text{Branch} = \text{Branch} \bowtie \text{staff.bno=branch.bno} \text{Staff} \)

- Commutativity of selection and theta-join (or Cartesian product).
  - If selection predicate involves only attributes of one of join relations, selection and join (or Cartesian product) operations commute:

\[ \sigma p(R \bowtie r S) = (\sigma p(R)) \bowtie r S \]

\[ \sigma p(R \times S) = (\sigma p(R)) \times S \quad \text{where } p \in \{ \text{A}_1, \text{A}_2, \ldots, \text{A}_n \} \]
If selection predicate is conjunctive predicate having form \((p \land q)\), where \(p\) only involves attributes of \(R\), and \(q\) only attributes of \(S\), selection and theta-join operations commute as:
\[
\sigma_{p \land q}(R \bowtie_r S) = (\sigma_p(R)) \bowtie_r (\sigma_q(S))
\]
\[
\sigma_{p \land q}(R \times S) = (\sigma_p(R)) \times (\sigma_q(S))
\]

Example: \(\sigma_{\text{position}='\text{Manager}' \land \text{city}='\text{London}'}(\text{Staff} \bowtie_{\text{staff.bno} = \text{branch.bno}} \text{Branch}) = (\sigma_{\text{position}='\text{Manager}'}(\text{Staff}) \bowtie_{\text{staff.bno} = \text{branch.bno}} (\sigma_{\text{city}='\text{London}'}(\text{Branch})))\)

Commutativity of projection and theta-join (or Cartesian product).
- If projection list is of form \(L = L_1 \cup L_2\), where \(L_1\) only involves attributes of \(R\), and \(L_2\) only involves attributes of \(S\), provided join condition only contains attributes of \(L\), projection and theta-join operations commute as:
\[
\Pi_{L_1 \cup L_2}(R \bowtie_r S) = (\Pi_{L_1}(R)) \bowtie_r (\Pi_{L_2}(S))
\]

- If join condition contains additional attributes not in \(L\), say attributes \(M = M_1 \cup M_2\) where \(M_1\) only involves attributes of \(R\), and \(M_2\) only involves attributes of \(S\), a final projection operation is required:
\[
\Pi_{L_1 \cup L_2}(R \bowtie_r S) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup M_1}(R)) \bowtie_r (\Pi_{L_2 \cup M_2}(S)))
\]

Example:
\[
\Pi_{\text{position, city, bno}}(\text{Staff} \bowtie_{\text{staff.bno} = \text{branch.bno}} \text{Branch}) =
(\Pi_{\text{position, bno}}(\text{Staff}) \bowtie_{\text{staff.bno} = \text{branch.bno}} (\Pi_{\text{city, bno}}(\text{Branch})))
\]

and using the latter rule:
\[
\Pi_{\text{position, city}}(\text{Staff} \bowtie_{\text{staff.bno} = \text{branch.bno}} \text{Branch}) =
\Pi_{\text{position, city}}((\Pi_{\text{position, bno}}(\text{Staff}) \bowtie_{\text{staff.bno} = \text{branch.bno}} (\Pi_{\text{city, bno}}(\text{Branch}))))
\]

Commutativity of union and intersection (but not set difference).
\[
R \cup S = S \cup R
\]
\[
R \cap S = S \cap R
\]

Commutativity of selection and set operations (union, intersection, and set difference).
\[
\sigma_p(R \cup S) = \sigma_p(S) \cup \sigma_p(R)
\]
\[
\sigma_p(R \cap S) = \sigma_p(S) \cap \sigma_p(R)
\]
\[
\sigma_p(R - S) = \sigma_p(S) - \sigma_p(R)
\]
Commutativity of projection and union.
\[ \Pi_L(R \cup S) = \Pi_L(S) \cup \Pi_L(R) \]

Associativity of union and intersection (but not set difference).
\[ (R \cup S) \cup T = S \cup (R \cup T) \]
\[ (R \cap S) \cap T = S \cap (R \cap T) \]

Associativity of theta-join (and Cartesian product).
- Cartesian product and natural join are always associative:
\[ (R \bowtie S) \bowtie T = R \bowtie (S \bowtie T) \]
\[ (R \times S) \times T = R \times (S \times T) \]
- If join condition \( q \) involves attributes only from \( S \) and \( T \), then theta-join is associative as follows:
\[ (R \bowtie_p S) \bowtie_q T = R \bowtie_p (S \bowtie_q T) \]

Example:
\[ (\text{Staff} \bowtie_{\text{staff.sno}=\text{property_for_rent.sno}} \text{Property_for_Rent}) \bowtie_{\text{ono}=\text{owner.ono}} \bowtie_{\text{staff.lname}=\text{owner.lname}} \text{Owner} = \]
\[ \text{Staff} \bowtie_{\text{staff.sno}=\text{property_for_rent.sno} \land \text{staff.lname}=\text{lname}} \text{Property_for_Rent} \bowtie_{\text{ono}} \text{Owner} \]

Heuristical Processing Strategies

- Perform selection operations as early as possible.
  - Keep predicates on same relation together.
- Combine Cartesian product with subsequent selection whose predicate represents join condition into a join operation.
  \[ \sigma_{R.a \theta S.b} (R \times S) = R \bowtie_{R.a \theta S.b} S \]
- Use associativity of binary operations to rearrange leaf nodes so leaf nodes with most restrictive selection operations executed first.
- Perform projection as early as possible.
  - Keep projection attributes on same relation together.
- Compute common expressions once.
  - If common expression appears more than once, and result not too large, store result and reuse it when required.
  - Useful when querying views, as same expression is used to construct view each time.
Example 18.3 Use of Transformation Rules

For prospective renters of flats, find properties that match requirements and owned by CO93.

```
SELECT p.pno, p.street
FROM renter r, viewing v, property_for_rent p
WHERE r.pref_type = 'Flat' AND
    r.rno = v.rno AND v.pno = p.pno AND
    r.max_rent >= p.rent AND
    r.pref_type = p.type AND p.ono = 'CO93';
```

(a)

(b)
Cost Estimation for RA Operations

- Many different ways of implementing RA operations.
- Aim of QO is to choose most efficient one.
- Use formulae that estimate costs for a number of options, and select one with lowest cost.
- Consider only cost of disk access, which is usually dominant cost in QP.
- Many estimates are based on cardinality of the relation, so need to be able to estimate this.

Database Statistics

- Success of estimation depends on amount and currency of statistical information DBMS holds.
- Keeping statistics current can be problematic.
- If statistics updated every time tuple is changed, this would impact performance.
- DBMS could update statistics on a periodic basis, for example nightly, or whenever the system is idle.

Typical Statistics for Relation R

- `ntuples(R)` - number of tuples in R (cardinality).
- `bfactor(R)` - blocking factor of R. (number of tuples of R that fit in one block).
- `nblocks(R)` - number of blocks required to store R:

  \[ nblocks(R) = \left\lfloor \frac{ntuples(R)}{bfactor(R)} \right\rfloor \]

Typical Statistics for Attribute A of Relation R

- `ndistinctA(R)` - number of distinct values that appear for attribute A in R.
- `minA(R),maxA(R)` - minimum and maximum possible values for attribute A in R.
- `SCA(R)` - selection cardinality of attribute A in R.
  
  Average number of tuples that satisfy an equality condition on attribute A.
  
  If values of A are uniformly distributed in R and that there is at least one value that satisfy the condition, then:

  \[
  SCA(R) = \begin{cases} 
  1 & \text{if A is a key attribute of R} \\
  \left\lfloor \frac{ntuples(R)}{ndistinctA(R)} \right\rfloor & \text{otherwise}
  \end{cases}
  \]

Statistics for Multilevel Index I on Attribute A

- `nlevelsA(I)` - number of levels in I.
- `nlfblocksA(I)` - number of leaf blocks in I.