Chapter 23: Query Processing

Dr. Xu Yang

Content

- Overview of query processing.
- Query decomposition.
- Cost of processing a query
- Heuristical approach to query optimization

Query Processing

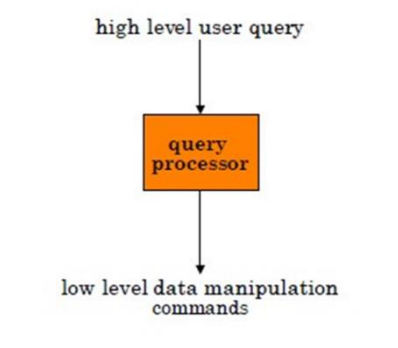
- One of the major criticisms often cited in Relational Model was inadequate performance of queries.
- One of the aims of query processing is to determine in which way to perform a complex query is the most cost effective.

Query Processing

- **Find <u>the most cost effective way</u> to perform a complex query.**
  - **In first generation and hierarchical database system**, the low level procedural query languages is generally embedded in a high-level programming language such as COBOL. **It is the programmer's responsibility to select the most appropriate execution strategy.**
  - In contrast, **with declarative languages such as SQL**, **the user specifies what data is required rather than how it is to be retrieved.** This relieves the user of knowing what constitutes good execution strategy. Also gives DBMS more control over system performance.

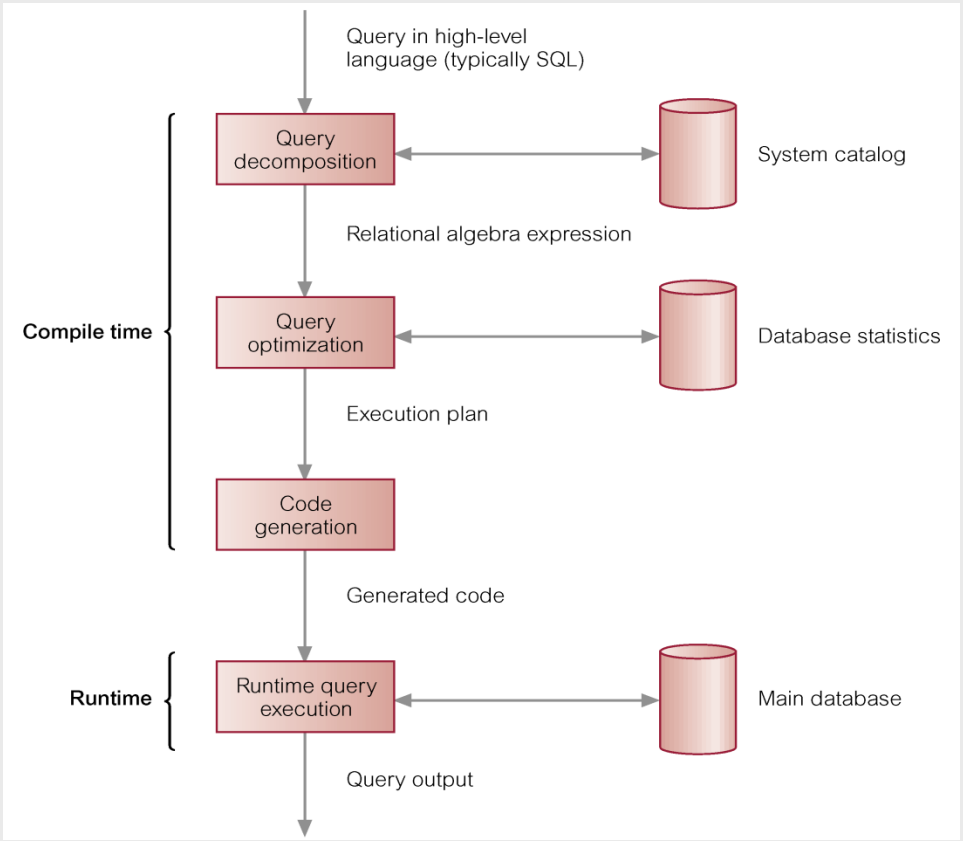
Aims of Query Processing

- **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 Relational Algebra (RA));
  - execute RA strategy to retrieve required data.



Phases of Query Processing

- **Query Processing : the activities involved in retrieving data from database.**



Phases of Query Processing

- **Query Decomposition**
  - To transform a high-level query into a relational algebra query,
  - and to check that the query is syntactically and semantically correct.
- **Query Optimization**
  - The activity of choosing an efficient execution strategy for processing a query.
- **Code Generation**
- **Runtime Query**

Two Choices of first three phases of QP

- **Two choices when first three phases of QP can be carried out:**
  - **Dynamic Query Optimization.** Dynamically carry out decomposition and optimization every time query is run;
  - **Static Query Optimization**. The query is parsed, validated, and optimized once, when query is first submitted.

Dynamic Query Optimization

- **Advantages** of dynamic QO arise from the fact that all information required to select an optimum strategy is up to date.

- **Disadvantages** are that performance of query is affected because the query has to be parsed, validated, and optimized before it can be executed. Time may limit finding optimum strategy.

Static Query Optimization

- **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, where the query is re-optimized if the system detects that the database statistics have changed significantly since the query was last compiled.

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.

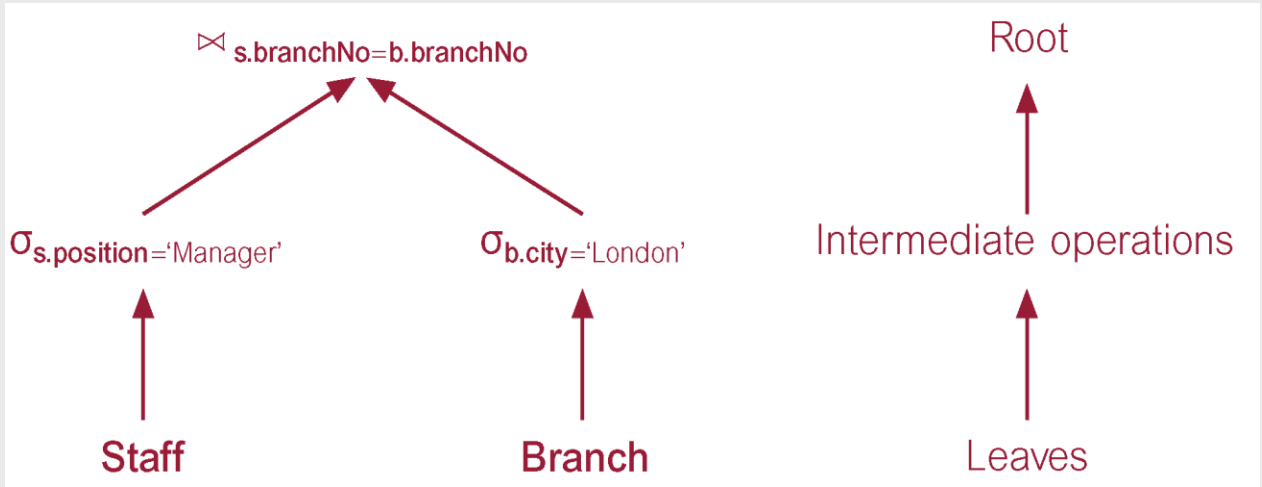
Example Relational Algebra Tree

Relational Algebra Tree

- 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.

$(\sigma_{position='Manager'}(Staff)) \bowtie_{Staff.branchNo=Branch.branchNo} (\sigma_{city='London'}(Branch))$

```
SELECT *
FROM Staff s, Branch b
WHERE s.branchNo =
b.branchNo AND
(s.position = 'Manager'
AND b.city = 'London');
```



Query Optimization

- **Two main techniques for query optimization:**
  - **comparing different strategies based on relative costs, and selecting one that minimizes resource usage.**
  - **heuristic rules that order operations in a query;**

- Cost is estimated using statistical information from the database catalog
  - » e.g. number of tuples in each relation, size of tuples, etc.

Query Optimization

Activity of choosing an efficient execution strategy for processing query.

- 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**.
- Problem computationally intractable with large number of relations, so strategy adopted is reduced to **finding near optimum solution**.

Cost of Query Evaluation

- The cost of query evaluation can be measured in terms of a number of different resources including
  - Disk Access
  - CPU time to execute a query,
  - in a distributed or parallel database system, the cost of communication
- **Disk access** tends to be dominant cost in query processing for centralized DBMS.
  - Disk access is slow compared with memory access
  - CPU speeds have been improving much faster than have disk speeds.

Example - Different Strategies

Find all Managers who work at a London branch.

```
SELECT *
FROM Staff s, Branch b
WHERE s.branchNo = b.branchNo AND
(s.position = 'Manager' AND b.city = 'London');
```

Three equivalent RA queries are:

(1) $\sigma_{(position='Manager') \wedge (city='London') \wedge (Staff.branchNo=Branch.branchNo)} (Staff \times Branch)$

(2) $\sigma_{(position='Manager') \wedge (city='London')} (Staff \bowtie_{Staff.branchNo=Branch.branchNo} Branch)$

(3) $(\sigma_{position='Manager'}(Staff)) \bowtie_{Staff.branchNo=Branch.branchNo} (\sigma_{city='London'}(Branch))$

Think about if we increase the number of tuples in Staff to 10000!

Example - Different Strategies

Metrics:

- 1000 tuples in the Staff relation
- 50 tuples in the Branch relation
- 50 employees are 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 of processing the following query alternate based on the number of disk accesses required.**

$\sigma_{(position='Manager') \wedge (city='London') \wedge (Staff.branchNo=Branch.branchNo)}$ **(Staff X Branch)**

Cartesian product of Staff and Branch:

(1000 + 50) record I/O's to read the relations

\+ (1000 * 50) record I/O's to create an intermediate relation to store result

Selection on result of Cartesian product:

(1000 * 50) record I/O's to read tuples and compare against predicate

Total cost of the query:

(1000 + 50) + 2*(1000 * 50) = **101, 050** record I/O's.

Example - Different Strategies

Metrics:

- 1000 tuples in the Staff relation
- 50 tuples in the Branch relation
- 50 employees are 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 of processing the following query alternate based on the number of disk accesses required.**

$\sigma_{(position='Manager') \wedge (city='London')}($ **Staff** $\bowtie_{Staff.branchNo=Branch.branchNo}$ **Branch)**

Join of Staff and Branch:

(1000 + 50) record I/O's to read the relations

\+ 1000 record I/O's to create an intermediate relation to store result

Selection on result of Join:

(1000) record I/O's to read tuples and compare against predicate

Total cost of the query:

(1000 + 50) + 2*(1000) = **3050** record I/O's.

Example - Different Strategies

- **Cost of processing the following query alternate based on the number of disk accesses required.**

$(\sigma_{position='Manager'}(Staff)) \bowtie_{Staff.branchNo=Branch.branchNo} (\sigma_{city='London'}(Branch))$

Select *'Managers'* in Staff:

(1000) record I/O's to read the relations

\+ (50) record I/O's to create an intermediate relation to store select result

Select *'London'* in Branch:

(50) record I/O's to read the relations

\+ (5) record I/O's to create an intermediate relation to store select result

Join of previous two selections over branchNo:

(50 + 5) record I/O's to read the relations

Total cost of the query:

(1000 2*(50) + 5 +(50 +5)) = **1, 160** record I/O's.

Cost Comparison

- **Cartesian product and join operations are much more expensive than selection .**

- one of the fundamental strategies in query processing is to perform the unary operations Selection and Projection, as early as possible.

Heuristical Approach to Query Optimization

Use Transformation Rules to convert one relational algebra expression into an equivalent form that is known to be more efficient.

- Two relational-algebra expressions are said to be **equivalent** if, on every legal database instance, the two expressions generate the same set of tuples.
  - Order of tuples is irrelevant
- By applying transformation rules, the system can restructure the relational algebra tree generated during query decomposition.

Transformation Rules for RA Operations

Conjunctive Selection operations can cascade into individual Selection operations (and vice versa).

$$\sigma_{p \wedge q \wedge r}(R) = \sigma_p(\sigma_q(\sigma_r(R)))$$

- **Sometimes referred to as cascade of Selection.**

$$\sigma_{branchNo='B003' \wedge salary>15000}(Staff) = \sigma_{branchNo='B003'}(\sigma_{salary>15000}(Staff))$$

Transformation Rules for RA Operations

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

$$\Pi_L \Pi_M \ \ldots \Pi_N(R) = \Pi_L (R)$$

- **For example:**

$$\Pi_{lName} \Pi_{branchNo,\ lName}(Staff) = \Pi_{lName} (Staff)$$

Transformation Rules for RA Operations

Commutativity of Selection and Projection.

- If predicate p involves only attributes in projection list, Selection and Projection operations commute:

$$\Pi_{Ai, \ldots, Am}(\sigma_p(R)) = \sigma_p(\Pi_{Ai, \ldots, Am}(R))$$
$$\text{where } p \in \{A_1, A_2, \ldots, A_m\}$$

- For example:

$$\Pi_{fName, lName}(\sigma_{lName='Beech'}(Staff)) =$$
$$\sigma_{lName='Beech'}(\Pi_{fName,lName}(Staff))$$

Transformation Rules for RA Operations

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 Equijoin and Natural join. For example:**

$$Staff \bowtie_{staff.branchNo=branch.branchNo} Branch = Branch \bowtie_{staff.branchNo=branch.branchNo} Staff$$

Transformation Rules for RA Operations

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:**

$$\sigma_p(R \bowtie_r S) = (\sigma_p(R)) \bowtie_r S$$

$$\sigma_p(R \times S) = (\sigma_p(R)) \times S$$

where $p \in \{A_1, A_2, \ldots, A_n\}$

Naturel joins are also commutative.

Transformation Rules for RA Operations

Commutativity of Projection and Theta join (or Cartesian product).

$$\Pi_{L1, L2}(R \bowtie_r S) = (\Pi_{L1}(R)) \bowtie_r (\Pi_{L2}(S))$$

Transformation Rules for RA Operations

Commutativity of Selection and set operations (Union, Intersection, and Set difference).

$$\sigma_p(R \cup S) = \sigma_p(S) \cup \sigma_p(R)$$

$$\sigma_p(R \cap S) = \sigma_p(S) \cap \sigma_p(R)$$

$$\sigma_p(R - S) = \sigma_p(S) - \sigma_p(R)$$

Transformation Rules for RA Operations

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).

$$(R \cup S) \cup T = S \cup (R \cup T)$$

$$(R \cap S) \cap T = S \cap (R \cap T)$$

Transformation Rules for RA Operations

Associativity of Theta join (and Cartesian product).

- **Cartesian product and Natural join are always associative:**

  $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

  $(R \times S) \times T = R \times (S \times T)$

we compute and store a smaller temporary relation.

Heuristical Processing Strategies

- **In essence, a basic SQL query block can be thought of as an algebra expression consisting of**
  - the Cartesian product of all relations in the FROM clause,
  - the selections in the WHERE clause
  - the projections in the SELECT clause.

SELECT * **FROM** Staff s, Branch b **WHERE** s.branchNo = b.branchNo **AND** (s.position = 'Manager' **AND** b.city = 'London');

$$\sigma_{(position='Manager') \wedge (city='London') \wedge (Staff.branchNo=Branch.branchNo)} (Staff \times Branch)$$

Heuristical Processing Strategies

- **Algebra equivalences allow us**
  - Push selections and projections ahead of joins.
  - Convert cross-products to joins
    - » Combine Cartesian product with subsequent Selection whose predicate represents join condition into a Join operation.
    - » $\sigma_{R.a\ \theta\ S.b}(\mathrm{R} \times S) = R \ \bowtie_{R.a\ \theta\ S.b} \mathrm{S}$
    - » Performing the selection and projection as early as possible reduces the size of the relation to be joined.
  - Choose different join orders,
    - » The smaller join is performed first, which means that the second join will also be based on a smaller first operand.

Example

SELECT * FROM Staff s, Branch b **WHERE** s.branchNo = b.branchNo **AND** (s.position = ‘Manager’ **AND** b.city = ‘London’);

(1) $\sigma_{(position='Manager') \wedge (city='London') \wedge (Staff.branchNo=Branch.branchNo)}$ **(Staff X Branch)**

(2) $\sigma_{(position='Manager') \wedge (city='London')}$**(** **Staff** $\bowtie_{Staff.branchNo=Branch.branchNo}$ **Branch)**

(3) $(\sigma_{position='Manager'}$**(Staff))** $\bowtie_{Staff.branchNo=Branch.branchNo}$ $(\sigma_{city='London'}$ **(Branch))**

Summary

- The objectives of query processing and optimization.
- Static versus dynamic query optimization.
- How to create a Relational Algebra Tree to represent a query.
- Cost of Processing a query
- Heuristical processing strategies