Chapter 18

Query Processing

Transparencies

Chapter - Objectives

- **Objectives of query processing and optimization.**
- **Static versus dynamic query optimization.**
- **How a query is decomposed and semantically analyzed.**
- **How to create a R.A.T. to represent a query.**
- **Rules of equivalence for RA operations.**
- **How to apply heuristic transformation rules to improve efficiency of a query.**

Chapter - Objectives

- **Types of database statistics required to estimate cost of operations.**
- **Different strategies for implementing selection.**
- **How to evaluate cost and size of selection.**
- **Different strategies for implementing join.**
- **How to evaluate cost and size of join.**
- **Different strategies for implementing projection.**
- **How to evaluate cost and size of projection.**

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.**

Introduction

- **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.**

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.**

Example 18.1 - Different Strategies

Find all Managers that work at a London branch.

```
SELECT *
FROM staff s, branch b
WHERE s.bno = b.bno AND
(s.position = 'Manager' AND b.city = 'London');
```

Example 18.1 - Different Strategies

- **Three equivalent RA queries are:**

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

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

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

Example 18.1 - Different Strategies

- **Assume:**
  - **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.**

Example 18.1 - Cost Comparison

- **Cost (in disk accesses) are:**

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

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

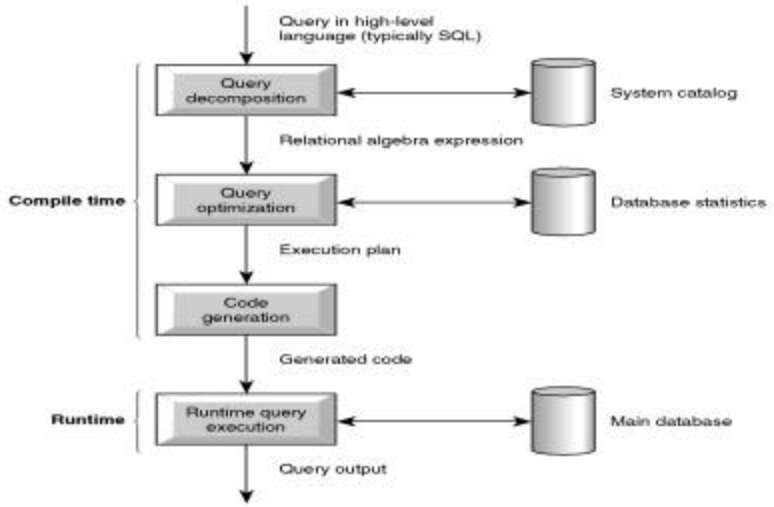
(3) 1000 + 2*50 + 5 + (50 + 5) = 1 160

- **Cartesian product and join operations are much more expensive than selection, and third option significantly reduces size of relations being joined together.**

Phases of Query Processing

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

Phases of Query Processing



Dynamic versus Static 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.**

Query Decomposition

- **Aims are to transform high-level query into RA query and check that query is syntactically and semantically correct.**
- **Analyze query lexically and syntactically using compiler techniques.**
- **Verify relations and attributes exist.**
- **Verify operations are appropriate for object type.**

Analysis - Example

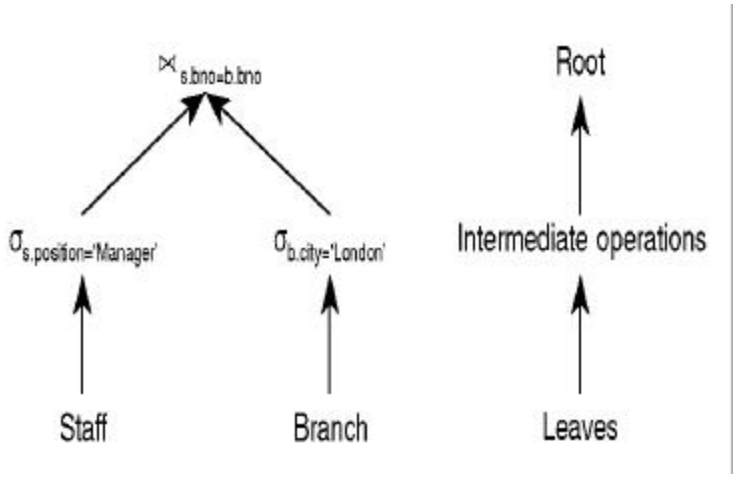
```
SELECT staff_no
FROM staff
WHERE position > 10;
```

- **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.**

Analysis

- **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.



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.**

Simplification

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

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.**

$$\sigma_{bno='B3' \wedge salary>15000}(Staff) = \sigma_{bno='B3'}(\sigma_{salary>15000}(Staff))$$

Transformation Rules for RA Operations

Commutativity of selection.

$$\sigma_p(\sigma_q(R)) = \sigma_q(\sigma_p(R))$$

- **For example:**

$$\sigma_{bno='B3'}(\sigma_{salary>15000}(Staff)) = \sigma_{salary>15000}(\sigma_{bno='B3'}(Staff))$$

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:**

$$\Pi_{lname} \Pi_{bno}(Staff) = \Pi_{lname}(Staff)$$

Transformation Rules for RA Operations

Commutativity of selection and projection.

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

$$\Pi_{A_i, \ldots, A_m}(\sigma_p(R)) = \sigma_p(\Pi_{A_i, \ldots, A_m}(R))$$

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 equi-join and natural join. For example:**

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

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

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

Transformation Rules for RA Operations

- **If selection predicate is conjunctive predicate having form (p Ù q), where p only involves attributes of R, and q only attributes of S, selection and theta-join operations commute as:**

$$\sigma_{p \wedge q}(R \bowtie_r S) = (\sigma_p(R)) \bowtie_r (\sigma_q(S))$$

$$\sigma_{p \wedge q}(R \times S) = (\sigma_p(R)) \times (\sigma_q(S))$$

Transformation Rules for RA Operations

- **For example:**

$$\sigma_{position='Manager' \wedge city='London'}(Staff \bowtie_{staff.bno=branch.bno} Branch) =$$

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

Transformation Rules for RA Operations

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:**

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

Transformation Rules for RA Operations

Associativity of theta-join (and Cartesian product).

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

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

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

- **If join condition q involves attributes only from S and T, then theta-join is associative as follows:**

  $(R \bowtie_p S) \bowtie_{q \wedge r} T = R \bowtie_{p \wedge r} (S \bowtie_q T)$

Transformation Rules for RA Operations

- For example:

$$(\text{Staff} \bowtie_{\text{staff.sno=property\_for\_rent.sno}} \text{Property\_for\_Rent}) \bowtie_{\text{ono=owner.ono} \wedge \text{staff.lname=owner.lname}} \text{Owner} =$$

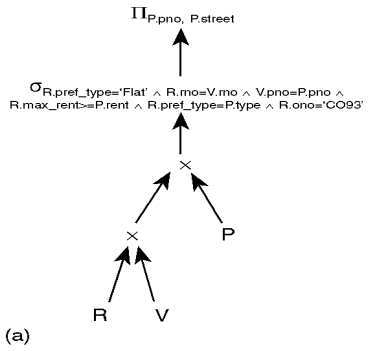
$$\text{Staff} \bowtie_{\text{staff.sno=property\_for\_rent.sno} \wedge \text{staff.lname=lname}} (\text{Property\_for\_Rent} \bowtie_{\text{ono}} \text{Owner})$$

Example 18.3 Use of Transformation Rules

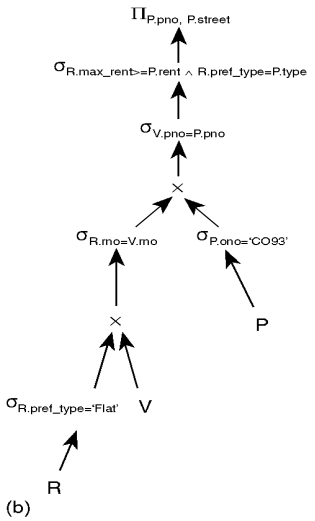
For prospective renters of flats, find properties that match requirements and owned by CO93.

```
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';
```

Example 18.3 Use of Transformation Rules

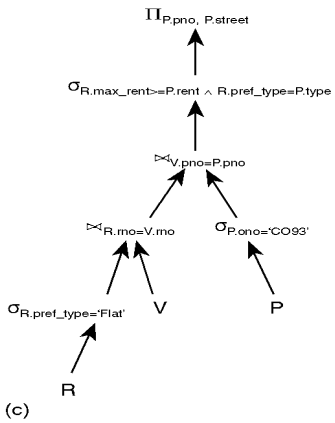


(a)

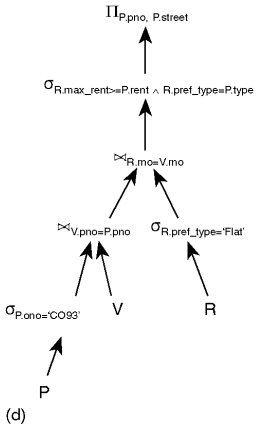


(b)

Example 18.3 Use of Transformation Rules

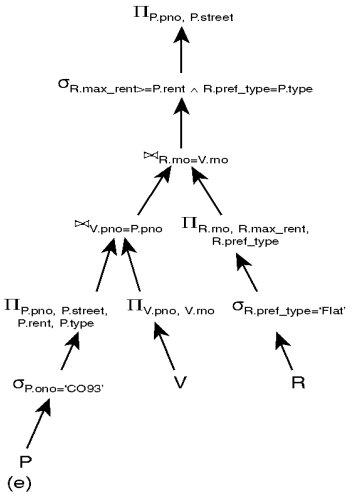


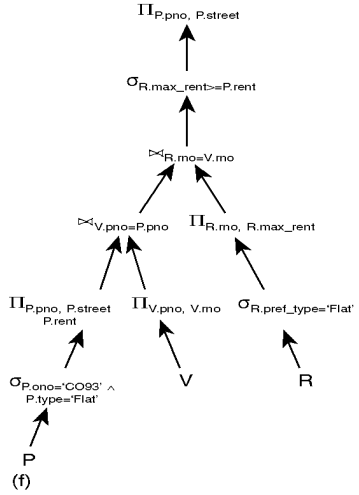
(c)



(d)

Example 18.3 Use of Transformation Rules





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.**
- **Use associativity of binary operations to rearrange leaf nodes so leaf nodes with most restrictive selection operations executed first.**

Heuristical Processing Strategies

- **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.**

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.

bfactor(R) - blocking factor of R.

nblocks(R) - number of blocks required to store R:

nblocks(R) = [ntuples(R)/bfactor(R)]

Typical Statistics for Attribute A of Relation R

$ndistinct_A(R)$ - number of distinct values that appear for attribute A in R.

$min_A(R)$, $max_A(R)$

- **minimum and maximum possible values for attribute A in R.**

$SC_A(R)$ - selection cardinality of attribute A in R.

Average number of tuples that satisfy an equality condition on attribute A.

Statistics for Multilevel Index I on Attribute A

$nlevels_A(I)$ - number of levels in I.

$nlfblocks_A(I)$ - number of leaf blocks in I.

Selection Operation

- **Predicate may be simple or composite.**
- **Number of different implementations, depending on file structure, and whether attribute(s) involved are indexed/hashed.**
- **Main strategies are:**
  - **Linear Search (Unordered file, no index).**
  - **Binary Search (Ordered file, no index).**
  - **Equality on hash key.**
  - **Equality condition on primary key.**

Equality of Hash Key

- **If attribute A is hash key, apply hashing algorithm to calculate target address for tuple.**
- **If there is no overflow, expected cost is 1.**
- **If there is overflow, additional accesses may be necessary.**

Equality Condition on Primary Key

- **Can use primary index to retrieve single record satisfying condition.**
- **Need to read one more block than number of index accesses, equivalent to number of levels in index, so estimated cost is:**

$$\text{nlevels}_A(I) + 1$$

Inequality Condition on Primary Key

- **Can first use index to locate record satisfying predicate (A = x).**
- **Provided index is sorted, records can be found by accessing all records before/after this one.**
- **Assuming uniform distribution, would expect half the records to satisfy inequality, so estimated cost is:**

$$\text{nlevels}_A(I) + [\text{nblocks}(R)/2]$$

Join Operation

- **Main strategies for implementing join:**
  - **Block Nested Loop Join.**
  - **Indexed Nested Loop Join.**
  - **Sort-Merge Join.**
  - **Hash Join.**

Estimating Cardinality of Join

- **Cardinality of Cartesian product is:**
  ntuples(R) * ntuples(S)
- **More difficult to estimate cardinality of any join as depends on distribution of values.**
- **Worst case, cannot be any greater than this value.**

Estimating Cardinality of Join

- If assume uniform distribution, can estimate for equi-joins with a predicate (R.A = S.B) as follows:
  - If A is key of R: ntuples(T) £ ntuples(S)
  - If B is key of S: ntuples(T) £ ntuples(R)
- Otherwise, could estimate cardinality of join as:

  ntuples(T) = $SC_A(R)$*ntuples(S) or

  ntuples(T) = $SC_B(S)$*ntuples(R)

Block Nested Loop Join

- Simplest join algorithm is nested loop that joins two relations together a tuple at a time.
- Outer loop iterates over each tuple in R, and inner loop iterates over each tuple in S.
- As basic unit of reading/writing is a disk block, better to have two extra loops that process blocks.
- Estimated cost of this approach is:

  nblocks(R) + (nblocks(R) * nblocks(S))

Indexed Nested Loop Join

- If have index (or hash function) on join attributes of inner relation, can use index lookup.
- For each tuple in R, use index to retrieve matching tuples of S.
- Cost of scanning R is nblocks(R), as before.
- Cost of retrieving matching tuples in S depends on type of index and number of matching tuples.
- If join attribute A in S is PK, cost estimate is:

$$\text{nblocks}(R) + \text{ntuples}(R) * (\text{nlevels}_A(I) + 1)$$

Sort-Merge Join

- For equi-joins, most efficient join is when both relations are sorted on join attributes.
- Can look for qualifying tuples merging relations.
- May need to sort relations first.
- Now tuples with same join value are in order.
- If assume join is M:N and each set of tuples with same join value can be held in database buffer at same time, then each block of each relation need only be read once.

Sort-Merge Join

- **Cost estimate for the sort-merge join is:**

  nblocks(R) + nblocks(S)

- **If a relation has to be sorted, R say, add:**

  nblocks(R)*[$\log_2$(nblocks(R)]

Hash Join

- **For natural or equi-join, hash join may be used.**
- **Idea is to partition relations according to some hash function that provides uniformity and randomness.**
- **Each equivalent partition should hold same value for join attributes, although it may hold more than one value.**
- **Cost estimate of hash join as:**

  3(nblocks(R) + nblocks(S)) + 2*max_partitions

Projection Operation

- To implement projection need following steps:
  - Removal of attributes that are not required.
  - Elimination of any duplicate tuples produced from previous step. Only required if projection attributes do not include a key.
- Two main approaches to eliminating duplicates:
  - Sorting
  - Hashing.

Duplicate Elimination using Sorting

- Sort tuples of reduced relation using all remaining attributes as sort key.
- Duplicates will now be adjacent and can be removed easily.
- Estimated cost of sorting is:

  $nblocks(R)*[\log_2(nblocks(R)]$.
- Combined cost is:

  $nblocks(R) + nblocks(R)*[\log_2(nblocks(R)]$

Duplicate Elimination using Hashing

- **Two phases: partitioning and duplicate elimination.**
- **In partitioning phase, for each tuple in R, remove unwanted attributes and apply hash function to combination of remaining attributes, and write reduced tuple to hashed value.**
- **Two tuples that belong to different partitions are guaranteed not to be duplicates.**
- **Estimated cost is: nblocks(R) + nb**

Aggregate Operations

SELECT avg(salary)
FROM staff;

- **To implement query, could scan entire Staff relation and maintain running count of number of tuples read and sum of all salaries.**
- **Easy to compute average from these two running counts.**

Aggregate Operations

```
SELECT avg(salary)
FROM staff
GROUP BY bno;
```

- For grouping queries, can use sorting or hashing algorithms similar to duplicate elimination.
- Can estimate cardinality of result using estimates derived earlier for selection.