Query Optimization

Introduction

- In Non-procedural DMLs (eg. SQL), user specifies what data is required rather than how it is to be retrieved.
- Relieves user of knowing what constitutes good execution strategy.
- 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

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

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

Example 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’);
```

3 equivalent RA queries are:

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

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

($\sigma_{position='Manager'}$(Staff)) $\bowtie$ ($\sigma_{city='London'}$ (Branch))

Example 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 1 - Cost Comparison

Cost (in disk accesses) are:

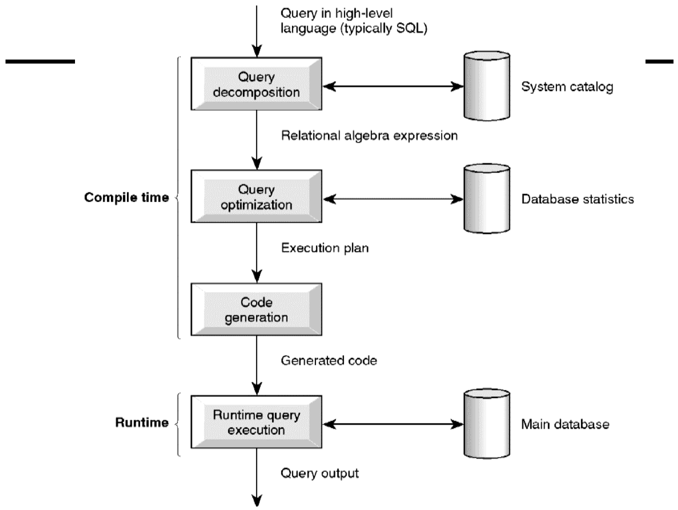
(1) (1000 + 50) + 2*(1000 * 50) = 101 050
(2) (1000 + 50) + 2*1000 = 3 050
(3) 1000 + 50 + 50 + 5 + (50 + 5) = 1 160

- **Cartesian product and join operations are much more expensive than selection**
- **(3) significantly reduces size of relations being joined together.**

Phases of Query Processing

QP has 4 main phases:

- **decomposition**
  - Aims are to transform high-level query into RA query and check that query is syntactically and semantically correct.
- **optimization**
- **code generation**
- **execution.**



Optimization: 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.

Optimization: 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.

Optimization: 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 <u>disk access</u>, which is usually dominant cost in QP.
- Many estimates are based on <u>cardinality</u> 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.

Pipelining

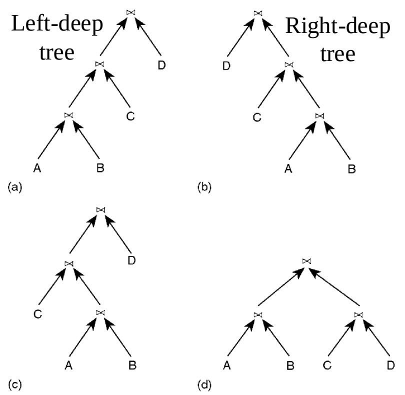
- Materialization - output of one operation is stored in temporary relation for processing by next.
- Could also pipeline results of one operation to another without creating temporary relation.
- Known as pipelining or on-the-fly processing.
- Pipelining can save on cost of creating temporary relations and reading results back in again.
- Generally, pipeline is implemented as separate process or thread.

Pipelining

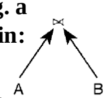
Types of Trees:

Linear Trees: (a),(b),(c)

Non-linear Tree: (d)



Eg. a join:



Outer relation **Inner relation**

Pipelining

- With linear trees, relation on one side of each operator is always a base relation.
- However, as need to examine entire inner relation for each tuple of outer relation, inner relations must always be materialized.
- This makes left-deep trees appealing as inner relations are always base relations.
- Reduces search space for optimum strategy, and allows QO to use dynamic processing.
- Not all execution strategies are considered.