QUERY OPTIMIZATION

Query Optimization

- queries are composed using relational algebra operations and algorithms to implement these operations
- a user can express a query in a wide variety of ways
- query optimization: find a good plan

Relational Model: Query Optimization

- users do not worry about how best to state their queries
  - the DBMS is responsible for query optimization
  - a good optimizer has a wealth of information available to it that human users typically do not have
    - the number of distinct values of each type
    - the number of tuples currently appearing in each relation
    - the number of distinct values appearing in each attribute
  - a reoptimization may be required if the database statistics change over time
  - the optimizer is a program
    - much more patient than a typical user
    - embodying the skills and services of the best human programmers

Review

- alternative ways of evaluating a given query: query plans
  - access to the records
  - order of operations
  - algorithms to perform operations
- goal of query optimization: find a good query plan
  - finding absolute best plan usually not feasible
  - sufficient to find reasonably good plan

Two Approaches

- Heuristic Approach
  - apply heuristic rules to try to speed up query processing
- Cost-based Approach
  - estimate the cost of several execution plans
  - choose the one with the lowest cost

Heuristic Approach

- estimate statistics for intermediate results and reduce the size of intermediary tables
  - apply selection and projections before joining
  - apply most restrictive SELECT statements first
  - apply restricts and projections as early as possible

Example:

- find names of the directors of movies which have titles beginning with the letter Z
- assumption:

number of tuples in movie relation = 10000

number of tuples in person relation = 100

number of movies with titles beginning with Z = 50

Example (Cont.)

Total cost= 1 030 000 I/O operation

```
SELECT name FROM  movie, person
   WHERE movie.directorid = person.id
   AND title like ‘Z%’
```

- join movie and person relation:
  - read 10,000 tuples from movie relation
  - read each of the 100 person 10,000 times (=1,000,000)
  - construct an intermediate result consisting of 10,000 joined tuples and write them to disk
  - read the 10,000 joined tuples back into memory and find the movies with titles beginning with Z
- project the result over name
  - 50 tuples

Example (Cont.)

Total cost = 10 100 I/O operations

```
SELECT name FROM  movie, person
  WHERE title like 'Z%'
  AND      movie.directorid = person.id
```

- restrict movie to just the tuples for movies with titles beginning with Z
  - read 10,000 tuples from movie relation
  - movies with titles beginning with Z: 50 tuples (can be kept in main memory)
- join the result to person relation
  - read 100 persons
  - produce a result 50 tuples (can be kept in main memory)
- project the result over name
  - 50 tuples

Example (Cont.)

- total cost = 150 I/O operations
  - unique index on title attribute of movie relation
    - read 50 tuples from movie relation to get the films that have a title starting with letter Z

- total cost = 100 I/O operations
  - unique index on title attribute of movie relation and id attribute of person relation
    - read 50 tuples from person relation

Query Optimization Process

- cast the query into internal form
- convert to canonical form
- choose candidate low-level procedures
- generate query plans and choose the cheapest

Query Optimization Process

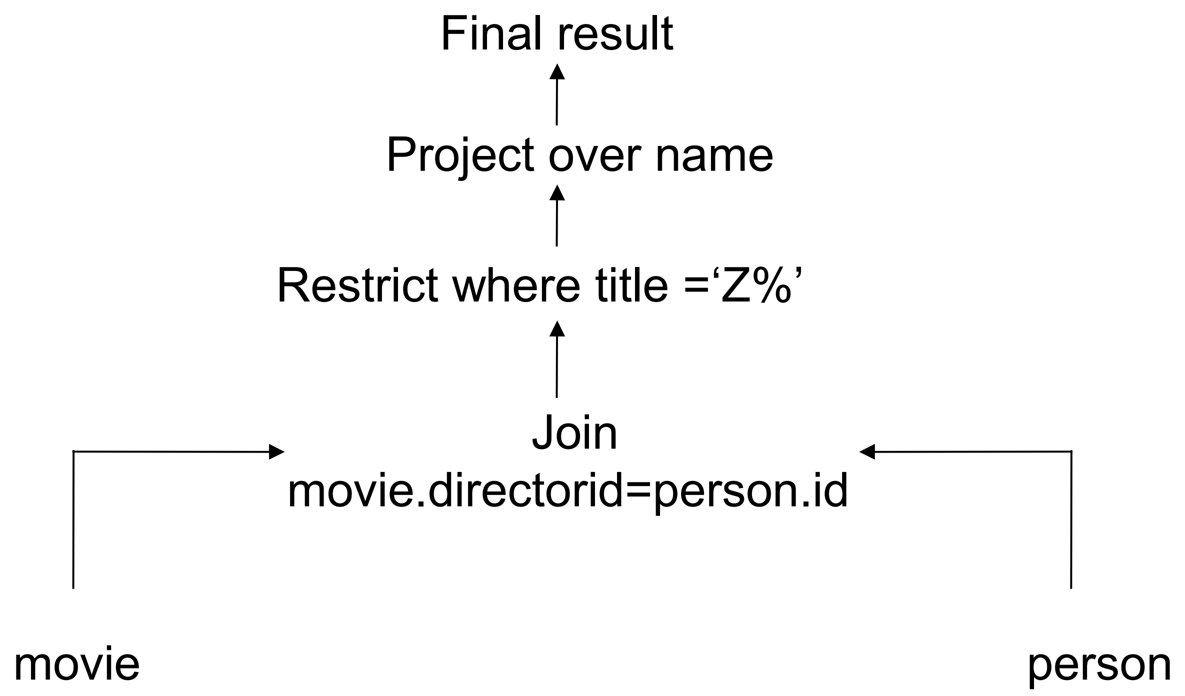
- **cast the query into internal form**
- convert to canonical form
- choose candidate low-level procedures
- generate query plans and choose the cheapest

Cast the query into internal form

- conversion of the original query into some internal representation that is more suitable for machine manipulation
  - all queries in the external query language should be representable
  - it should not prejudice subsequent choices
  - abstract syntax tree or query tree

Query Tree

```
SELECT name FROM  movie, person
  WHERE movie.directorid = person.id
  AND     title like 'Z%'
```



Query Optimization Process

- cast the query into internal form
- **convert to canonical form**
- choose candidate low-level procedures
- generate query plans and choose the cheapest

Convert to canonical form

- queries can be expressed in a variety of ways

- convert the internal representation into some equivalent canonical form
  - guaranteed to be good
  - regardless of the actual data values and physical access paths

Expression Transformation

- a sequence of restrictions on the same relation can be transformed into a single *AND*ed restriction

```
SELECT ... FROM
   (SELECT ... FROM A WHERE p1) AS ...
  WHERE p2
SELECT ... FROM A WHERE (p1 AND p2)
```

Example: Expression Transformation

```
SELECT * FROM
(SELECT * FROM movie WHERE score > 7) AS m
WHERE year = 1985
```

- equivalent expression

```
SELECT * FROM movie
WHERE score > 7 AND year = 1985
```

Expression Transformation

- in a sequence of projections against the same relation, all but the last can be ignored

```
SELECT acl2 FROM
    (SELECT acl1 FROM A) AS ...
SELECT acl2 FROM A
```

Example: Expression Transformation

```
SELECT title FROM
(SELECT title, year FROM movie) AS m
```

- equivalent expression

```
SELECT title FROM movie
```

Expression Transformation

- a restriction of a projection can be transformed into a projection of a restriction

```
SELECT ... FROM
     (SELECT acl FROM A) AS ...
  WHERE p

SELECT acl FROM
   (SELECT * FROM A WHERE p) AS ...
SELECT acl FROM A WHERE p
```

Distributivity

- f is said to be distributive over $\circ$ if

  $f(A \circ B) \equiv f(A) \circ f(B)$

- restriction distributes over union, intersection and difference
  - in some cases over join
- projection distributes over union and intersection
  - projection does not distribute over difference

Expression Transformation

- other transformation laws
  - commutativity

    $A \circ B \equiv B \circ A$
  - associativity

    $A \circ (B \circ C) \equiv (A \circ B) \circ C$
  - idempotence

    $A \circ A \equiv A$
  - absorption

    $A \text{ UNION } (A \text{ INTERSECT } B) \equiv A$

Semantic Transformations

```
SELECT MOVIEID FROM CASTING,MOVIE
   WHERE (CASTING.MOVIEID=MOVIE.ID)
```

```
SELECT MOVIEID FROM CASTING
```

- due to the integrity constraint
  - foreign to matching candidate key join
    - every casting tuple does join to some movie tuple

Query Optimization Process

- cast the query into internal form
- convert to canonical form
- **choose candidate low-level procedures**
- generate query plans and choose the cheapest

Divide-and-Conquer Strategy

- break down complex queries into a sequence of smaller queries
  - detachment
  - tuple substitution
- example: names starting with ‘A’ of the lead actors of the movies with votes more than 500 and made after 1990

Example

```
SELECT DISTINCT ACTOR.NAME
  FROM ACTOR,CASTING,MOVIE
  WHERE (ACTOR.NAME LIKE ‘A%’)
    AND (ACTOR.ID=CASTING.ACTORID)
    AND (CASTING.ORD=1)
    AND (CASTING.MOVIEID=MOVIE.ID)
    AND (MOVIE.YEAR>1990)
    AND (MOVIE.VOTES>500)
```

Example

```
D1: SELECT ID FROM MOVIE
      WHERE (YEAR>1990)
        AND (VOTES>500)
SELECT DISTINCT ACTOR.NAME
  FROM ACTOR,CASTING,D1
  WHERE (ACTOR.NAME LIKE 'A%')
    AND (ACTOR.ID=CASTING.ACTORID)
    AND (CASTING.ORD=1)
    AND (CASTING.MOVIEID=D1.ID)
```

Example

```
D2: SELECT ACTORID,MOVIEID FROM CASTING
      WHERE (ORD=1)
SELECT DISTINCT ACTOR.NAME
  FROM ACTOR,D2,D1
  WHERE (ACTOR.NAME LIKE 'A%')
    AND (ACTOR.ID=D2.ACTORID)
    AND (D2.MOVIEID=D1.ID)
```

Example

```
D3: SELECT ID,NAME FROM ACTOR
      WHERE (NAME LIKE 'A%')
SELECT DISTINCT D3.NAME
  FROM D3,D2,D1
  WHERE (D3.ID=D2.ACTORID)
    AND (D2.MOVIEID=D1.ID)
```

Example

```
D4: SELECT D2.ACTORID FROM D2,D1
      WHERE (D2.MOVIEID=D1.ID)
SELECT DISTINCT D3.NAME
  FROM D3,D4
  WHERE (D3.ID=D4.ACTORID)
```

Query Optimization Process

- cast the query into internal form
- convert to canonical form
- choose candidate low-level procedures
- **generate query plans and choose the cheapest**

Generate query plans and choose the cheapest

- construct a set of candidate query plans
  - a set of candidate implementation procedures, one procedure for each of the low-level operation in the query
    - instead of generating all plans, generate a subset of plans using heuristic techniques
- assign a cost to any given plan
  - sum of the costs of the invidual procedures
  - cost formulas depend on the size(s) of relations to be processed and intermediate results
    - use database statistics

Database Statistics

- for each base table:
  - number of tuples
- for each column of each table:
  - number of distinct values in this column
  - max./min. value, second max./min. value, average value
  - for indexed columns, the ten most frequently occuring values

Implementing the Relational Operators

- brute force
- index lookup
- hash lookup
- merge
- hash
- combinations of the above methods

Example

- join two relations: R, S
  - number of tuples in relation R = m
  - number of tuples of relation R on a page = pR
  - number of tuples in relation S = n
  - number of tuples of relation S on a page = pS
- C is the common attribute
- dCR: number of distinct values of the C attribute in relation R
- dCS : number of distinct values of the C attribute in relation S

Brute Force

```
do i := 1 to m
  do j := 1 to n
     if R[i].C = S[j].C then
          add joined tuple R[i]*S[j] to
  result;
  end;
end;
```

- Cost:
  - m + (m * n)
  - (m / pR) + (m * n) / pS

Index Lookup

- an index X on attribute C of S
  - X[1] ...X[k] indexes

```
do i := 1 to m
  do j := 1 to k
  /* let tuple of S indexed by X[j] be S[j] */
     if R[i].C = S[j].C then
          add joined tuple R[i] * S[j] to
  result;
  end;
end;
```

- Cost:
  - (m / pR) + ((m * n) / dCS)
  - (m / pR) + ((m * n) / dCS) / pS

Hash Lookup

- S.C is a hash

```
do i := 1 to m
  k := hash (R[i].C);
  /* let there be h tuples stored at */
  do j := 1 to h
     if R[i].C = S[j].C then
     add joined tuple R[i] * S[j] to result;
  end;
end;
```