Query Optimisation

Objective

Find the "optimum" set of access paths to retrieve the required data. Applies to updates and queries.

Reasons for automation:

1) Machine can use more info.
2) Re-optimisation easier following data reorg.
3) Optimiser can evaluate more solutions than a human.
4) Automation makes expertise more widely available.

Intervention

- Still scope for human intervention: eg use of "hints" to guide Oracle optimiser, or rigging statistics for Oracle or DB2.
- Still very dependent on programmer skills because query syntax dramatically affects access path choices, eg whether or not an index is used.
- The term optimisation is an over claim.

Example 1.1

- Get names of suppliers who supply part 'p2':

```
select distinct s.sname
    from s, sp
            where s.s# = sp.s#
                    and sp.p# = 'p2’
```

- Database contains 100 suppliers and 10,000 shipments, 50 of which supply 'p2'.

- Consider how to evaluate without optimisation?

Unoptimised

1) Compute cartesian product of s an sp, involves reading the 10,000 sp tuples 100 times, = 1,000,000 tuple reads. Product will contain 1,000,000 tuples which will need to be written back to disk.

2) Apply restriction in the "where" clause, involves 1,000,000 tuple reads but gives a 50 row result which can stay in memory.

3) Project result of step 2 over sname to give final result, containing at most 50 tuples which again we assume can stay in memory.

Optimised

1) Restrict SP to those tuples containing 'p2', 10,000 tuple reads but result has 50 rows which stays in memory.

2) Join result of step 1 to relation s over s#, 100 tuple reads and results in 50 tuples, still in memory.

3) As above, Project result of step 2 over sname to give final result of 50 tuples.

Summary

- Optimised version about 300 times faster in terms of tuple i/o's: unoptimised needs about 3,000,000 i/o's, optimised needs about 10,100.
- So a Restriction followed by a Join instead of a Product and then the Restriction has produced a dramatic improvement.

Indexing

- If SP indexed or hashed on p#, tuples read in step 1 would be 50 rather than 10,000, and optimised version then about 20,000 times faster.

- Also an index on s.s# would reduce the 100 tuple i/o's in step 2 to at most 50.

- Caution: in practise block i/o's are what count.

Step In Query Optimisation

Step 1

Put the query into a form suitable for machine representation. Representation must handle all possible queries, and must be neutral, i.e. must not prejudice the optimisation process. Typical formalisms include Relational Algebra or query trees (see Date).

Step 2

Convert to canonicle form, that is to a standard form which eliminates differences that users may introduce because it is possible to express the same query in different ways within the language. So we apply standard transformation rules to put the queries into a standard and efficient form.

Eg-transformation

- (a Join b) where Restriction on b transforms to:

  a Join (b where Restriction on b)

- I.e. do Restrictions first, or more generally:

  (a Join b) where Restriction on a and Restriction on b Is equivalent to:

  (a where restriction on a) Join (b where Restriction on b).

Step 2

Doing the restrictions before the Join:

1) Reduces the size of the input to the Join.

2) Reduces size of output from the Join, possibly enabling the result to stay in memory and so further reducing disk I/O.

Conjunctive NF

(another transformation)

Converts any Restriction into an equivalent set of conditions which are AND-ed together, where each condition in turn consists of a set of simple comparisons connected only by ORs.

Example

- The "where" clause:

  WHERE p or (q and r)

- Where p, q and r are restriction conditions, can be converted to:

  WHERE (p or q) and (p or r).

- The terms in brackets are called conjuncts.

Advantage

- A condition in conjunctive normal form evaluates to true only if every conjunct evaluates to true.

- The optimiser might then choose to evaluate simpler conditions first, or those that are "least likely" to be true to avoid evaluating other conditions unnecessarily.

Parallelism

Conjuncts may be evaluated simultaneously on parallel processors, the overall process being immediately terminated if any condition evaluates to false.

Step 2-Summary

In general many transformation rules may be applied, but they may not always be independent of one another and may sometimes conflict.

Step 3

- Choose candidate low level procedures.
- This involves deciding how to evaluate the newly represented query.
- Makes use of indexes, clustering, distribution of stored data values, etc.
- Involves the selection of predefined low level procedures to perform relational algebra operations.

Step 3 cont.

Needs to account for dependencies between operations, eg Project requires its input to be sorted into order so that duplicates can be eliminated, so the operation preceding the Projection must provide its output in sequence.

Step 3 cont.

Examples of pre-defined procs:

- Perform equi-Join.
- Restriction for when the restriction field is indexed.
- One for when there is no index but the data is clustered on the restriction field.etc.
- Each such proc will have an associated cost measure.

Step 3 cont.

- Using statistical and syntactic information, and the inter-dependencies between operations, a pre-defined proc will be chosen for each low level operation in the query.
- This process is sometimes called access path selection.

Step 4

- Generate query plans and choose the cheapest.
- Each plan is built by combining sets of candidate implementation procedures corresponding to low-level query operations.
- Heuristics will be needed to limit the number of query plans generated.

Step 4 cont.

- Evaluating the cost of query plans may involve a complex cost formula.
- Usually however disk I/O is the dominant component, perhaps followed by CPU time.
- The estimation of intermediate results generated during query execution complicates the costing as such results are very data dependent.

Steps summary

1) Formulate query into a suitable machine representation.

2) Convert to canonical form.

3) Choose candidate low level procedures.

4) Generate query plans and choose the cheapest.

Approaches to optimisation

- Rule based (also called syntax based) optimisers.
- Uses structure of SQL statements, including: Functions called, Use of nulls, Order of clauses, Availability of indexes, clustering.
- Actions are predictable whereas stats optimisation is data dependent.

Statistics based (also called cost based)

<u>**Uses:**</u>

- No. of tuples in tables
- No. of columns
- Similar statistics about available indexes
- Max and min values
- No. of distinct values
- Data sequencing.

Oracle

- Uses both syntax and stats.
- Defaults to stats providing stats are available.
- User must choose which component to use.
- Stats are collected with the "analyse table" or "analyse index" commands.
- Stats can be computed exactly or estimated by sampling a subset of data, (first 1,000 rows).
- No stats are collected unless you "analyse".

Good SQL practice I

1) Efficient structuring of the data, schema design, indexes, clusters, keys.

2) Efficient structuring of queries.

3) Providing more info to the optimiser.

Good SQL practice II

- Avoid the use of * and count(*)
- Avoid long table and column names
- use brief table aliases in Joins
- Specify table.columns in Joins

Good SQL practice III

- Eliminate rows as early as possible:

```
Select job,avg(sal)
   from emp
          where job != 'MANAGER'
                    group by job
```

- is better than:

```
Select job,avg(sal)
   from emp
          group by job
                    having job != 'MANAGER'.
```

Good SQL practice IV

- Be aware of peculiarities and assumptions of your DBMS's optimiser.

  Typical examples include:

- Indexes not used with:

  - Maths on index column in WHERE clause.
  - Like '%c%' - must specify start of string.
  - Comparison with null in WHERE clause.
  - *, not equals: assumes returning majority of rows so index not used.

Good SQL practice V

- Order of tables in FROM clause matters: aim to get as many tables in main memory as possible.
- Order of clauses in WHERE clause matters: aim to remove as many rows as early as possible.
- on composite indexes

  -If knows all values then uses the index.

  -If just has leading part then it uses that.

  -If only has trailing part then composite index not used.

Good SQL Practice summary

- Joins often faster than subqueries.
- Remove rows as early as possible.
- Indexes speed queries with low hit rate, but slow updates.
- Normally index primary and foreign keys consider indexing other attributes heavily used in WHERE clauses.
- Be aware of the order in which FROM and WHERE clauses are processed.

DISTRIBUTED QUERY OPTIMISATION

- May Involve global and local optimisation plans.
- Must account for cost of comms between sites.
- Opportunity for parallell execution?

Issues I

- Where should optimiser be in the network?
- Local V. global optimisation.
- Do as much locally as possible to minimise network traffic.
- Transmission rate across network is a major new factor in choosing between query execution strategies.

Issues II

- Possible site differences in processor speeds, reliability, processor loads, user priority, storage capacities.
- Optimiser requires dictionary. But where should the dictionary be stored?
- Use of Primary copy of dictionary?
- Moving primary copy?
- Site autonomy?

Data Replication

If use data replicas then:

- Which table fragments should be duplicated?
- How should they be fragmented: horizontally, vertically, mixed?
- What should be the frequency of updates?

SEMI-JOINS

- JOIN table T1 at site A with table T2 at site B:

1) PROJECT the JOIN column, T1.J from T1 and send to site B.
2) JOIN T2 with T1.j and send results back to A.

- Advantages:

1) Only send key rather than whole rows in first transfer
2) Only bring back those rows that satisfy the JOIN.

Statistics

- For optimisers with a stats component, account must be taken of local changes in data volumes, distributions etc.

- execution plans may become out of date due to local changes in stats.