Database Tuning

Goal

Make a set of applications execute “as fast as possible”.

- optimize for response time?
- optimize for overall throughput?

How can we affect performance (as DBAs)?

- make queries run faster (data structures, clustering, replication)
- make updates run faster (locality of data items)
- minimize congestion due to concurrency

How to Speed up Queries?

- generate all physical plans equivalent to the query
- pick the one with the lowest cost

How does it Search for Plans?

- Query equivalence is undecidable!!
  - always good transformations, e.g.,
    \[ \sigma_{A=\text{p}}(R \bowtie S) \leftrightarrow (\sigma_{A=\text{p}}(R)) \bowtie S \]
- rewrites using integrity constraints
- cost-based access path selection and join ordering

\[ \ldots \text{the query optimizer may not find the right plan (query tuning)} \]
Cost Models for Query Optimization

How do we figure out how fast a particular plan is??

For every stored relation $R$ with an attribute $A$ we keep:

- $|R|$: the cardinality of $R$ (the number of tuples in $R$)
- $b(R)$: the blocking factor for $R$
- $\min(R, A)$: the minimum value for $A$ in $R$
- $\max(R, A)$: the maximum value for $A$ in $R$
- $\text{distinct}(R, A)$: the number of distinct values of $A$

⇒ in practice much more complex statistics are kept

Based on these values we try to estimate the cost of query plans
⇒ costs estimated in number of disk I/O operations

Example: Cost of Retrieval

Schema: `Mark(Studnum, Course, Assignnum, Mark)`
Query: `SELECT Studnum, Mark FROM Mark WHERE Course = 'PHYS' AND Studnum = 100 AND Mark > 90`
Indices: clustering index `CourseInd` on `Course`
non-clustering index `StudnumInd` on `Studnum`
Stats:
- $|\text{Mark}| = 10000$
- $b(\text{Mark}) = 50$
- 500 different students
- 100 different courses
- 100 different marks

Strategy 1: Use `CourseInd`

Assuming uniform distribution of tuples over the courses, there will be about $|\text{Mark}|/100 = 100$ tuples with $\text{Course} = \text{PHYS}$.

Searching the `CourseInd` index has a cost of 2. Retrieval of the 100 matching tuples adds a cost of $100/b(\text{Mark})$ data blocks.

The total cost is 4.

Strategy 2: Use `StudnumInd`

Assuming uniform distribution of tuples over student numbers, there will be about $|\text{Mark}|/500 = 20$ tuples for each student.

Searching the `StudnumInd` has a cost of 2. Since this is not a clustered index, we will make the pessimistic assumption that each matching record is on a separate data block, i.e., 20 blocks will need to be read.

The total cost is 22.

Selection of $N$ tuples from relation $R$ using a clustered index has a cost of $2 + N/b(R)$.

Selection of $N$ tuples from relation $R$ using a non-clustered index has a cost of $2 + N$. 
Strategy 3: Scan the Relation

The relation occupies 10,000/50 = 200 blocks, so 200 block I/O operations will be required.

Selection of $N$ tuples from relation $R$ by scanning the entire relation has a cost of $|R|/b(R)$.

in practice: more complex designs available (= more choices) ... and the estimation is extended to all relational operators

More complex Designs

Multi-attribute Indices

complex search/join conditions (in lexicographical order!) index-only plans (several clustered indices)

Join Indices

allow replacing joins by index lookups

Materialized Views

allow replacing subqueries by index lookups

Problem 1

how does the query optimizer know if/where to use such indices/views?

Problem 2

balance between cost of rematerialization and savings for queries.

Index Selection and Tools

Idea

Convert physical design into another optimization problem

- generate the space of all possible physical designs
- pick the best one based on a given WORKLOAD

Workload

An abstraction of applications executed against a database:

- list of queries
- list of updates
- frequencies/probabilities of the above
- sequencing constraints
- ...

Query Plan Tools (EXPLAIN)

How do we know what plan is used (and what the estimated cost is)?

⇒ db2expln and dynexpln tools

select name from author, wrote where aid=author

(without index)

Estimated Cost = 50
Estimated Cardinality = 120
Optimizer Plan:

RETURN
MSJOIN
\TBSCAN \TBSCAN
SORT SORT
\TBSCAN \TBSCAN
Table: Table: AUTHOR WROTE

(index on wrote(author))

Estimated Cost = 25
Estimated Cardinality = 120
Optimizer Plan:

RETURN
MSJOIN
\TBSCAN IXSCAN
\SORT \Index: Table: AW WROTE
\TBSCAN
\Table: AUTHOR
Index Selection and Tools Example

```bash
rees$ db2advis -d cs338
   -s "select name from author,wrote where aid=author"
```

Calculating initial cost (without recommended indexes) [25.390385]
Initial set of proposed indexes is ready.
Found maximum set of [2] recommended indexes
Cost of workload with all indexes included [0.364030] timerons
total disk space needed for initial set [ 0.014] MB
total disk space constrained to [ -1.000] MB
2 indexes in current solution
25.3904] timerons (without indexes)
[ 0.3640] timerons (with current solution)
98.57] improvement

Trying variations of the solution set.
--
-- execution finished at timestamp 2006-11-23-12.25.24.205770
--
-- LIST OF RECOMMENDED INDEXES
--
-- index[1], 0.009MB
 create index WIZ8 on "DAVID "."AUTHOR" ("AID" ASC, "NAME" ASC);
-- index[2], 0.005MB
 create index AW on "DAVID "."WROTE" ("AUTHOR" ASC);
--
Index Advisor tool is finished.

⇒ can be fed a workload instead of a single query

Schema Tuning and Normal Forms

So far we only added data structures to improve performance.
what to do if this isn’t enough?

Changes to the conceptual design

Goals:
- avoid expensive operations in query execution (joins)
- retrieve related data in fewer operations

Techniques:
- alternative normalization/weaker normal form
- co-clustering of relations (if available)/denormalization
- vertical/horizontal partitioning of data (and views)
- avoiding concurrency hot-spots

Summary

Physical design has enormous impact on performance

- decisions based on understanding what the DBMS is doing
  ⇒ query execution, transaction processing, and query optimization
- modern systems provide tools for DBAs (EXPLAIN)
- VERY active area of research