Thursday 23/4/2015 Query Execution

Lecture Topics

I. Query Plan
II. Finding Records
III. Operator Re-Ordering
IV. Algorithm Selection

I. Query Plan

A query plan is an extended R.A. tree with annotations for the access/implementation method.
Ex. R(a,b) and S(a,c)

Select R.a
FROM R, S
WHERE R.a = S.a
AND R.b = 100
AND S.c > 10

- For any given query, there may be various ways to execute it.
- The DBMS must:
  - consider different plans (i.e. need an algorithm to search plan space)
  - maintain statistics about the database and workload (e.g. profiling)
- Estimate the cost of a plan
- Decide which plan to use
- Ideally choose the best
- Usually don't choose the worst

II. Finding Records

Example:
Say we have a file with 100,000 records, 100 records per block (1,000 blocks)
- There is a hash index on the records
- File is unclustered
- To answer a query, we need to read all blocks that contain relevant records.

Scenario 1: Find 50 records.
   Probably need to access 50 (or almost) 50 blocks. Good to use an index.

Scenario 2: Find 2,000 records.
   Note only 2% of all records, but very likely that the records are in 1,000 blocks (or almost). Better to traverse the file rather than use an index.
Example:

```sql
SELECT SSN
FROM Employee
WHERE DOB = '1982-05-03'
AND SEX = 'F';
```

Two strategies:

1. Find all blocks for DOB = '1982-05-03'
   and then pick the records where
   SEX = 'F'.

2. Find all blocks for SEX = 'F'
   and then pick records where
   DOB = '1982-05-03'.

If 50% of employees are F, then
(2) is a bad strategy.

Either system must profile, or
SQL programmer must influence
plan.

### Operator Re-ordering

**SELECT**
FROM R
WHERE A = 1
AND B = 'MARY' => 

**IMAGINE THERE ARE INDEXES ON**
A AND B.

=> CAN EASILY FIND
ONE RECORD WITH A = 1
OR ONE RECORD WITH
B = 'MARY'.

**MORE EFFICIENT TO**
**DO THE MORE SELECTIVE operation**
first.

- Some databases profile,
  decide which index is most
  selective.

- Some pick first in sequence (e.g. A)
- Some pick last in sequence (e.g. B.)
  - Oracle used to do this.

- Programmer may have to re-write
  their query:
  ```sql
  WHERE A = 1 AND B = 'MARY'
  OR
  WHERE B = 'MARY' AND A = 1;
  ```

- **Pushing Selection**:
  A **JOIN** is an expensive operation.
  ⇒ simple strategy is nested loops. O(n^2)

```
```
```
IV Algorithm Selection

Example:

\[ R(A, B) \]
\[ S(E, D) \]

- No indexes
- 1,000 blocks in R
- 10,000 blocks in S

Available 3 blocks of RAM

Query:

\[ \text{SELECT * FROM R, S WHERE R.B=S.E} \]

Strategy 1:

Read 2 blocks of R into memory
- Read 1 block of S into memory
- Check for join condition on all resident blocks
- Repeat 10,000x (for all 5 blocks)

Repeat for all subsets of 2 of R
(Total 50x2)

Total reads = 1 Read of R + 500 Reads of S =
5,001,000 blocks read

Strategy 2:

Read 2 blocks of S into memory
- Read 1 block of R
- Check join condition
- Repeat 1,000x (for all 8 blocks)

Repeat for all 3 subsets
(Total 5,000 times)

Total reads = 1 Read of S + 5,000 Reads of R =
5,010,000 blocks read

Order of Joins:

Example:

Lives(Person, City) about people in the U.S.
\( \sim 300,000,000 \) tuples

Oscar(Person) people in the U.S. who
have won an Oscar \( \sim 1,000 \) tuples

Nobel(Person) people in the U.S. who
have won a Nobel \( \sim 100 \) tuples

Query: Find GoodMatch(Person, Person2)
where the two people live in the
same city, and the first person won
an Oscar and the second won
a Nobel

- How would you do this "by hand"?

- With SQL:

\[ \text{SELECT Oscar.} \text{Person as P1, Nobel.} \text{Person as P2 FROM Oscar, Lives, Lives2, Nobel, Lives Lives2 WHERE Oscar.} \text{Person = Lives.} \text{Person AND Nobel.} \text{Person = Lives2.} \text{Person AND Lives1.} \text{City = Lives2. City} \]

Very slow!!!
To sort R:
- Partition R
- Sort each partition
- Merge sorted partitions

Merge Join
- If R is sorted on R1 and S is sorted on S2, we can use merge to join them.
- Merge current candidates, output smaller.
- Compare current candidates, output smaller.

IDEA:
- Hash join is similar to an index join, except that we hash relations using the same hash function.
- Hash join attribute for both relations is in position 1.
- We hash relations into 2 partitions.
- We know that R1 joins in partition 2, so only join with S1 in partition 2.

PDF: