Databasesystemer, Forår 2006

Rasmus Pagh
Esben Rune Hansen

Øvelser d. 23. marts

Øvelserne for denne uge er relateret til forelæsningen om "Database Effektivitet". De indeholder et ark med en opgave fra sidste års eksamen og et ark med nogle maskinøvelser. Det anbefales at du bruger ca. lige meget tid på begge typer opgaver. Eksamens opgaven nævner 2 forespørgsler. Forespørgel 2 indeholder en *sub-query* som I først lærer hvad er i næste uge. Det giver dog også god mening at løse en del af opgaven, hvor man kun bestragter den første forespørgsel.

1 Exam problem from Databasesystemer, June 2005 (15 %)

Consider the following SQL queries:

```
1:     SELECT speciale
       FROM Behandler
       WHERE postnr=1000 AND sygdom='hundegalskab';

2:     SELECT speciale
       FROM Behandler
       WHERE postnr=1000 AND id IN (SELECT id
                                    FROM Behandler
                                    WHERE sygdom='hundegalskab');
```

The two queries refer to the attributes `speciale`, `postnr`, `sygdom` and `id`. All attributes are selective in the sense that only a small fraction of the tuples will have a particular value on any given attribute. The attribute `id` is even a key for the relation (but not declared as primary key).

To improve the running time of the queries, it is considered to create one or more indexes. This problem is concerned with evaluating different possibilities for indexing.

1. Evaluate for each of the mentioned attributes the effect of having an index on this attribute as the *only* index. State which of the two queries (if any) the index would make run faster. Which possibility (or possibilities) would be best, with regards to making both queries fast? Argue for your answer.

2. Consider each of the following four possibilities for forming a composite index:

   - `CREATE INDEX idx1 ON Behandler(postnr,sygdom);`
   - `CREATE INDEX idx2 ON Behandler(sygdom,postnr);`
   - `CREATE INDEX idx3 ON Behandler(id,postnr);`
   - `CREATE INDEX idx4 ON Behandler(postnr,id);`

   Again, evaluate for each of the mentioned indexes the effect of having it as the *only* index. State which of the two queries (if any) the index would make run faster.

3. Propose a combination of two indexes that would make the queries run as fast as possible. Argue for your choice.

2 Hands-on exercises

Before we can begin we need to do three things:

1. Create a table used by autotrace
2. Turn autotrace on
3. Create at large table (projects)

These three things can be done by copy-pasting from the file:
http://www.itu.dk/people/pagh/DBS06/efficiencyinitial.sql into oracle. Please drop the relation `PROJECTS` when you are done with the exercises, they take up a lot of space at ITUs servers.

The statistics shows a lot of information. You should in general about three things:

The Execution Plan: E.g. whether the query is done by full table scan or by index scan.

Consistent gets: The number of requested reads on the hard disk.

Physical reads: The actual number of reads on the hard disk – i.e. number of requested read that are not in oracles internal buffer.

We are now ready to begin the actual exercises

1. Run these example statements:
```
SELECT count(*) FROM projects WHERE last2='Gearloose'and last3='Rabbit';
INSERT INTO projects values ('Rasmus','Pagh',11,'Toger','Norgaard',10,'Lise','Gregersen',10);
CREATE INDEX groupmembers23 on projects (last2,last3,first2,first3);
INSERT INTO projects values ('Rasmus','Pagh',13,'Toger','Norgaard',11,'Lise','Gregersen',11);
SELECT count(*) FROM projects WHERE last2='Gearloose'and last3='Rabbit';
SELECT count(*) FROM projects WHERE last2='Gearloose';
SELECT count(*) FROM projects WHERE last3='Rabbit';
```
   For each query notice the kind of scan used, as shown in the execution plan. Try to compare and explain the changes in the number of consistent gets and physical reads for similar operations.

2. Run these example statements:
```
SELECT count(*) FROM projects WHERE last1='Gearloose'and last2='Rabbit';
SELECT max(grade2) FROM projects WHERE last1='Gearloose'and last2='Rabbit';
```
   Notice and explain any changes in the number of consistent gets and physical reads.

3. Compare the speed of updates to the database with and without an index, as follows:
   - `commit` all previous changes.
   - Delete all projects with `grade1` $<$ `6`. Note the time in seconds and the number of db gets / physical reads.
   - `rollback` to the previous state.
   - Create an index on `grade2`.
   - Again delete all projects with `grade1` $<$ `6`.

   Again note the number of consistent gets and physical reads.