6.830 Problem Set 2 (2009)

Assigned: September 24, 2009

Due: October 20, 2009, 11:59 PM

The purpose of this problem set is to give you some practice with concepts related to schema design, query planning, and query processing.

Part 1 - Query Plans

We show two SQL queries over the NSF database used in Problem Set 1 below. For these two queries, you will examine the query plan that Postgres uses to answer the query, and try to understand why it produces the plan it does.

To understand what query plan is being used, PostgreSQL includes the `EXPLAIN` command. It prints the plan for a query, including all of the physical operators and access methods being used. For example, the following SQL command displays the query plan for the `SELECT`:

```
EXPLAIN SELECT name FROM researchers WHERE researchers.name like '%Madden%' ;
```

You can run this by first connecting to hancock's PostgreSQL server with

```
psql -h hancock.csail.mit.edu -p 5433 6.830-2009 username
```

To understand the output of `EXPLAIN`, you will probably want to read the performance tips chapter of the PostgreSQL documentation:

`http://www.postgresql.org/docs/8.3/static/performance-tips.html`

We have run `VACUUM FULL ANALYZE` on all of the tables in the database, which means that all of the statistics used by PostgreSQL should be up to date.

For the following two queries, answer these questions:

a. What physical plan does Postgres use? Your answer should consist of a drawing of a query tree annotated with the access method types and join algorithms.

b. Why do you think Postgres selected this particular plan?

c. What does Postgres estimate the size of the result set to be?

d. When you actually run the query, how big is the result set?

e. Run some queries to compute the sizes of the intermediate results in the query. Where do Postgres's estimates differ from the actual intermediate result cardinalities?

f. Given the tables and indices we have created, do you think Postgres has selected the best plan? You can list all indices with `\di`, or list the indices for a particular table with `\d tablename`. If not, what would be a better plan? Why?

1. [5 points]: Query 1:

```
SELECT r1.name
FROM researchers AS r1, researchers AS r2, grants, grant_researchers AS gr
WHERE grants.pi = r2.id
AND grants.id = gr.grantid
AND gr.researcherid = r1.id
AND r1.org = 10
AND r1.org != r2.org;
```

2. [10 points]: Query 2:

```
SELECT researchers.name, gr.grantid
FROM grant_researchers AS gr, grant_programs AS gp, grant_fields AS gf, researchers
WHERE gr.grantid = gp.grantid
AND gr.grantid = gf.grantid
AND gr.researcherid = researchers.id
AND gf.fieldid < 200
AND gp.programid < 200
ORDER BY gr.grantid;
```

3. [10 points]:

Use `EXPLAIN` to find the query plans (including access methods types and join algorithms) for the following three queries, draw the query tree for each of them, and answer the questions below.

1. ```
   SELECT researchers.org
   FROM researchers, grants
   WHERE researchers.id = grants.pi
   AND grants.amount > 40000000;
   ```
2. ```
   SELECT researchers.org
   FROM researchers, grants
   WHERE researchers.id = grants.pi
   AND grants.amount > 3500000;
   ```
3. ```
   SELECT researchers.org
   FROM researchers, grants
   WHERE researchers.id = grants.pi
   AND grants.amount > 35000;
   ```

a. You may notice that the query plans for the three queries above are different, even though the all have the same general form. Explain in a sentence or two why the plans are different.

b. More generally, if we're running the queries above where we vary the constant s after the '$>$', for what value of s does Postgres switch between the three plans? Why does it decide to switch plans at that value? Please be as quantitative as possible in your explanation.

c. Suppose the crossover point (from the previous question) between queries 1 and 2 is s_0. Compare the actual running times corresponding to the two alternative query plans at the crossover point. How much do they differ? Inside `psql`, you can measure the query time by using the `\timing` command to turn timing on for all queries. To get accurate timing, you may also wish to redirect output to `/dev/null`, using the command `\o /dev/null`; you can stop redirection just by issuing the `\o` command without a file name. You may also wish to run each query several times, throwing out the first time (which may be longer as data is loaded into the buffer pool) and averaging the remaining runs.

d. Based on your answers to the previous two problems, is s_0 actually the best place to switch plans, or is it an overestimate/underestimate of the best crossover point? If s_0 is not the actual crossover point, can you estimate the actual best crossover point without running queries against the database? State assumptions underlying your answer, if any.
   ```

Part 2 - Access Methods

Ben Bitdiddle is building a database to catalog his extensive wardrobe. Being a computer scientist, he records each item of clothing, as well as compatible "outfits" which look good together.

Ben creates four tables:

- A table of clothes called `C`, with schema <*cid int, type char(50), description char(100), value float*> and |`C`| records in it,
- A table with metadata about outfits in it `O`, with schema <*oid int, color char(20), outfittype char(50)*> and |`O`| records in it, and
- A many-to-many mapping table specifying which clothes are in a particular outfit `CO`, with schema <*co_cid int, co_oid int*> and |`CO`| records in it.
- A table containing a list of events and the outfit he wore to each `E`, with schema <*eid int, e_description char(100), e_oid id, t timestamp*> and |`E`| records in it.

You may assume ints, floats, and timestamps are 4 bytes, and characters are 1 byte. Disk pages are 10 kilobytes. In the following three questions, we provide you with a physical database design and a query workload, and ask you to explain why the database system chooses a particular plan or what plan you think the database would select.

Ben's database supports heap files and B+Trees (clustered and unclustered.) Assume that you can have at most one clustered index per file, and that the database system has up-to-date statistics on the cardinality of the tables, and can accurately estimate the selectivity of every predicate. Assume B+Tree pages are 50% full, on average. Assume disk seeks take 10 ms, and the disk can sequentially read 100 MB/sec.

For the queries below, you are given the following statistics:

Statistic	Value
Clothing Items Per Outfit	5
Outfits Per Clothing Item	20
Distinct Types of Clothing	10
Distinct Types of Outfits	5
Number of Clothing Items `|C|`	10^6
Number of Outfits `|O|`	4×10^6
Number of Events `|E|`	1000
First Event	1/1/1990

In the absence of other information, assume that attributes values are uniformly distributed (e.g., that there are approximately the same number of each type of clothing, etc.)

Ben creates an unclustered B+Tree over `C.type`. He then runs the query `SELECT * FROM C WHERE type = 'sock'`. He finds that when he runs this query using the `EXPLAIN` command, the database does not use the B+Tree he just created.

4. [5 points]: Why not? What does the database probably do instead? How much more expensive would it be to use the B+Tree, in terms of total disk I/Os and disk seeks?

He now runs the following query to find the value of all of his formalwear outfits worn to events since 1999 (note that this query may include the value of some items of clothing multiple times.)

```
SELECT SUM(value) FROM C,O,CO,E
WHERE co_cid = cid AND co_oid = oid
AND e_oid = oid AND outfittype = 'formalwear'
AND t >= '1/1/1999'.
```

5. [5 points]: Draw the query plan you think the database would use for this query given that Ben had created a clustered B+Tree over `oid`, a clustered B+Tree over `cid`, a clustered B+Tree over `co_oid, co_cid`, and a clustered B+Tree over `e_oid`.

6. [5 points]: Draw the query plan you think the database would use for this query, supposing that only heap files were available. Estimate the cost difference (in terms of total seeks and I/Os) between this plan and the plan above based on B+Trees.

7. [5 points]: Now suppose that the indices were not clustered (assume the keys are randomly distributed throughout the heap file.) Estimate the cost (in terms of total seeks and I/Os) of a query plan using these indices as the access method for each of the tables. Compare this cost to the one for using the clustered indices described in Q5 as the access methods for each of the tables.

Part 3 – Schema Design and Query Execution

Betty Butcher is building a database to store data about patients she's treated in her roadside surgery shack. She wants to record information about her patients, their visits to her shack, the treatments she has available, and the treatments her patients have received. Specifically, her database has the following requirements:

1. It must record each patient's name, address, and date of birth, and the date of death (something that happens a bit too often in Betty's shack.)
2. It must record dates that each patient visited the shack, and the treatments they received on each visit. Each patient may receive multiple treatments per visit.
3. It must record names (e.g., "hacksaw"), types (e.g., "amputation") , descriptions (e.g., "cut that sucker off"), and per-unit cost (e.g., "$100") of different treatments. Different patients may receive different numbers of each treatment (e.g., two amputations would be recorded as two-units of 'hacksaw') and each patient may receive different amounts of each treatment on different visits.

8. [5 points]: Suggest a set of non-trivial functional dependencies for this database from Betty's requirements given above. You may make additional assumptions when deriving these FDs (e.g., that patients only die or are born once, or addresses don't change.)

Not having taken 6.830, Betty isn't sure how to use FDs to generate a schema, so she decides to make something up. Specifically, she represents her database with two tables; a "patientvisits" table which includes information about a patient and his or her visits, and a table of "treatments" in which the treatments that were administered are recorded.

patientvisits : {`pv_id | p_name | p_addr | p_bday | p_diedon | pv_visitdate` }

treatments : {`t_id | t_visitid REFS patientvisits.pv_id| t_name | t_type | t_description | t_cost | t_amount` }

In the `patientvisits` table, a given patient may appear multiple times if he or she has visited the shack repeatedly. Similarly `treatments` table includes one row for each treatment a patient receives on each visit.

This representation works fine for a few months, but as Betty's database grows, she finds it is very hard to maintain, particularly as the popularity of her service grows.

9. [5 points]: Based on your functional dependencies, list three problems that Betty's schema may cause as her database grows and is subject to lots of inserts and deletes.

10. [10 points]: Draw an ER diagram for the data Betty wants to store.

11. [5 points]: Use your ER diagram to derive a normalized schema for Betty's database. You may want to introduce different or additional attributes (e.g., `id` fields) than those Betty chose to use in her initial schema.

12. [5 points]: Use your functional dependencies to derive a schema in BCNF for Betty's database.

13. [2 points]: Are the two representations you derived equivalent? If not, in what way do they differ?

14. [3 points]: Do both schemas address the three problems you listed above? Do they both preserve all of the functional dependencies you gave?