Exam 2 - Advanced Databases

Antonio Badia

Spring 2024

This exam asks you to answer several questions; on each question, part of the answer is SQL and part are direct answers. Combine all your answers in one document. The document can be plain text, Word, or pdf. Mark clearly which question you are answering. Give only one answer per query. If you have any doubts, please contact the instructor ASAP.

1. **Unnesting and Query Optimization** In this question, you are asked to run and time several queries from the TPC-H Benchmark. Note that all queries in the Benchmark come with *parameters*, placeholders which need to be substituted with a certain value (constant). For each query, the benchmark provides several acceptable values (Substitution Parameters) and one particular value that is used to make sure the query returns the right answer (Query Validation). We will always run the queries with the query validation values. To capture query plans, use EXPLAIN (`https://www.postgresql.org/docs/current/sql-explain.html`).

   (a) Using the file provided on Blackboard, create the schema for the TPC-H benchmark database in Postgres. Using the files provided in the TPC-H data folder, upload the data into the files, using the COPY command (`https://www.postgresql.org/docs/current/sql-copy.html`) or other means. You are welcome to use any help you can find online (e.g. `https://programmer.help/blogs/import-tpch-generated-data-into-postgres-database.html` or others) to get the data in the database.

   What you need to provide: a `SELECT COUNT(*)` on each table and the result you get once you have uploaded the data into the database.

   (b) Run Q17 of the TPCH benchmark and capture the query plan and the time it takes to run the query in your system (all you need to provide for this question is the time but you will need the query plan for later).

   What you need to provide: the time it takes to run the query.

   (c) Unnest Q17 using the procedure seen in class, run the unnested query and capture the query plan and the time it takes to run the unnested query. **Extra credit:** use magic sets.

   What you need to provide: the unnested query, and the time it takes to run it.

   (d) Compare query plans and times. Which version is faster, and by how much? For whatever version is faster, where are the gains coming from (use the query plan to justify your answer)?

   What you need to provide: a short, focused answer. Explain the gains by pointing to 2-3 differences between the query plan.

2. **Materialized Views (I)** Create a materialized view that will accelerate Q5 of the TPCH benchmark. The view can join at most 3 tables, group by at most 3 attributes, and include at most one selection. Rewrite Q5 to take advantage of this materialized view. NOTE: to make a difference, the view should include a fact table (ORDER or LINEITEM or both). Proceed in order:

   (a) Create the materialized view.

   (b) Run the two original query and get the time.

(c) Rewrite the query to take advantage of the materialized view.

(d) Run the rewritten version of the query and get the time.

What you need to provide: the `CREATE MATERIALIZED VIEW` command; the query as rewritten by you to take advantage of the view; the time it takes to run the query originally; and the time it takes to run the rewritten query using the materialized view. NOTE: make sure the output you get with your rewritten query is the same as the one you get with the original query; this will help establish that your rewriting is correct. If the rewriting is not correct, it does not matter how fast it is!

3. **Materialized Views, II** Assume the following SQL, which defines a materialized view on the TPCH database.

```
CREATE MATERIALIZED VIEW OrdCost AS
SELECT c_custkey, sum(o_totalprice)
FROM ORDER, CUSTOMER
WHERE ORDER.c_custkey = CUSTOMER.c_custkey
      and o_shippriority = 0 and c_acctbal > 0
GROUP BY c_custkey
```

Show the update needed to keep the materialized view in sync when there are insertions in ORDER or in CUSTOMER by

- showing the code to create the ORDER-INSERTIONS table and the CUSTOMER-INSERTIONS table that are needed to update the view.
- the code needed to update the view using these tables (hint: this is the count algorithm).

4. **ETL Processes** The database `sakila` is a sample database provided by MySQL, available at `https://dev.mysql.com/doc/index-other.html`. The file `sakila-schema.sql` in Blackboard contains the CREATE TABLE statements necessary to create the database; however, this script won't run on Postgres as is. Modify it so that it creates the database in Postgres. HINT: the file as is does not run in Postgres for a multitude of reasons. You will have to fix all these problems; it'll probably take you several attempts. One way to do this is to run the file through Postgres and pay attention to the error messages you get.

   What you need to provide: the modified file as an SQL script; a list of the changes you made.

5. **Column Stores** We are going to estimate the gains one can get from a column store using the TPC-H database.

   (a) From the schema declaration in Postgres, calculate the size of storing table Lineitem at 1 SF size in a traditional (row-based) file. Give file size in Megabytes.

   (b) For each column in Lineitem that is used in Q1, calculate the size (note: you can assume the columns are compressed by a factor of 5). Give file sizes in Megabytes.

   (c) Use EXPLAIN to get the query plan the system uses to run Q1, and the cost.

   (d) Provide an sketch of a query plan for the column-based approach using late materialization.

   (e) Using the sizes above, calculate a cost for the columnar-based query processing (note: cost of Invisible Join and Jive/Flash Join on two columns of sizes M and N can be approximated by the cost of merge-sort join: roughly, $M \log_2 M + N \log_2 N + M + N$).

   (f) How to the costs of both approaches compare?