Assignment: Advanced Querying: SQL and Relational Algebra

COMPSCI 2DB3: Databases–Winter 2024

Deadline: March 23, 2024

Department of Computing and Software
McMaster University

Please read the *Course Outline* for the general policies related to assignments.

Plagiarism is a *serious academic offense* and will be handled accordingly.
All suspicions will be reported to the *Office of Academic Integrity*
(in accordance with the Academic Integrity Policy).

This assignment is an *individual* assignment: do not submit work of others. All parts of your submission *must* be your own work and be based on your own ideas and conclusions. Only *discuss or share* any parts of your submissions with your TA or instructor. You are *responsible for protecting* your work: you are strongly advised to password-protect and lock your electronic devices (e.g., laptop) and to not share your logins with partners or friends!

If you *submit* work, then you are certifying that you are aware of the *Plagiarism and Academic Dishonesty* policy of this course outlined in this section, that you are aware of the Academic Integrity Policy, and that you have completed the submitted work entirely yourself. Furthermore, by submitting work, you agree to automated and manual plagiarism checking of all submitted work.

Late submission policy. Late submissions will receive a late penalty of 20% on the score per day late (with a five hour grace period on the first day, e.g., to deal with technical issues) and submissions five days (or more) past the due date are not accepted. In case of technical issues while submitting, contact the instructor *before* the deadline.

Description

Consider the following relational schema for an online marketplace that consists of the following relations:

- **user**(*uid*, name, city).

  The **user** relation contains the user name of users and the city in which they live.

- **offer**(*oid*, seller_id, product, price).

  The **offer** relation contains products that are put up for sale by a user (identified via seller_id, a foreign key referencing **user**) for a specified price.

- **sale**(*sid*, customer_id, offer_id, time).

  The **sale** relation contains a record of all sales. In a sale, a customer (identified via customer_id, a foreign key referencing **user**) buys an offer (identified via offer_id, a foreign key referencing **offer**) at a given time (a timestamp that includes both the date and time).

- **sreview**(*sale_id*, score).

  After a sale (identified via sale_id, a foreign key referencing **sale**), the customer can review their experience with the seller by rating their experience (a score between 1 and 10).

- **preview**(*product*, *user_id*, score).

  Each user (identified via user_id, a foreign key referencing **user**) can review their experience with products (identified via product, which matches the product names used in **offer**) by rating their experience (a score between 1 and 10).

Part 1: The requested SQL queries

1. *Compute product overviews.*

   Each product will have a small overview of how well the product is behaving in the marketplace.

   Query: Write a query that returns a tuple (product, avgprice, avgscore, numsales, numsellers) such that *product* identifies a product, *avgprice* is the average price of all offers for that product, *avgscore* is the average review score for the product, *numsales* is the total number of sales of that product, and *numsellers* is the total number of distinct sellers of the product.

2. *Duplicate sellers.*

   An issue on the marketplace is people that spam their products via multiple seller accounts. Hence, the marketplace wants to identify multiple seller accounts operated by the same people: to identify such accounts, we look for distinct pairs of seller accounts A and B such that they offer exactly the same set of products for the same price.

   Query: Write a query that returns pairs (*aid*, *bid*) of distinct user identifiers *aid* and *bid* of sellers A and B such that sellers A and B offer exactly the same products for exactly the same price. If a seller A is offering the same product for multiple prices, then seller B only has to match one of these prices and vice versa.

3. *Community-support score.*

   The marketplace thrives due to user-generated content: users that review products and sellers are the main source of information used by other users to make purchasing decisions. Hence, the marketplace wants to reward all users that write a significant amount of reviews. As a first step in doing so, the marketplace wants to compute a community-support score per user.

   The community support score is a value s, $0 \leq s \leq 20$. If the user did not write at-least 10 reviews, then the user does not have a community support score. Otherwise, if the user wrote at-least 10 reviews (for either products or sellers), then the community support score for user u is determined as follows: let n_p be the number of distinct products bought by user u, let n_s be the number of distinct sellers user u bought products from, let r_p be the total number of products reviewed by user u, and r_s be the total number of distinct sellers reviewed by user u. The community support score is given by

   $$10 \cdot \left(\frac{r_p}{n_p} + \frac{r_s}{n_s} \right).$$

   Query: Write a query that returns pairs (uid, css) in which *uid* is the user identifier of a user U and *css* is the community support score of user U. You can exclude users without community support scores.

4. *Time-scaled seller rating.*

   The marketplace has noticed that the scores of sellers fluctuate over time. Hence, the current rating of a seller should not be based on the *average review score*; instead *recent* reviews should carry a higher weight than older reviews.

   To do so, the marketplace is proposing a time-scaled rating. Let $r_1, \ldots, r_n$ be all reviews of a seller U in **sreview** ordered on increasing time of sale and let $s_1, \ldots, s_n$ be the corresponding scores of reviews

$r_1, \ldots, r_n$. We note that s_n is the score of the most-recent review r_n seller U received and s_1 is the score of the least-recent review r_1 seller U received. The time-scaled rating for seller U is determined as follows:

$$\text{tsr}(U) = 2 \cdot \frac{1 \cdot s_1 + 2 \cdot s_2 + \cdots + n \cdot s_n}{n \cdot (n+1)}$$

We note that $1 + 2 + \cdots + n = \frac{n \cdot (n+1)}{2}$. Hence, if all review scores $s_1, \ldots, s_n$ are the same, then $\text{tsr}(U)$ is equivalent to these review scores.

Query: Write a query that returns pairs (uid, tsr) in which *uid* is the user identifier of a seller U and *tsr* is the time-scaled rating $\text{tsr}(U)$ of that seller. You may ignore users that did not receive any seller reviews. You may assume that no two sales happen at exactly the same point in time.

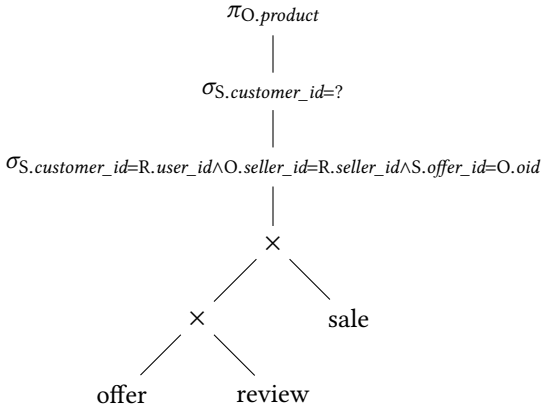
Part 2: Efficiency of queries

There are 2000 rows in **user**, 5000 rows in **offer** (5 offers per user), 20000 rows in **sale** (10 sales per user, each offer having 4 sales), and 10000 rows in **review** (5 reviews per user, each offer having 2 reviews).

Consider the following query that returns the products a specified user (parameter ?) bought and for which the user wrote a review for the seller of that product:

$$\pi_{\text{O}.product}(\sigma_{\text{S}.customer\_id=?}(\sigma_{\text{S}.customer\_id=\text{R}.user\_id \wedge \text{O}.seller\_id=\text{R}.seller\_id \wedge \text{S}.offer\_id=\text{O}.oid}(\rho_S(\text{sale}) \times \rho_O(\text{offer}) \times \rho_R(\text{review})))).$$

Assume that this query will be evaluated via the following query execution plan:



Proceed in the following steps:

1. Estimate the size of the output of all intermediate steps in the above query execution plan.
2. Provide a *good* query execution plan for the above relational algebra query. Explain why your plan is good.
3. Estimate the size of the output of all intermediate steps in your query execution plan. Explain each step in your estimation.
4. Finally, also provide an SQL query equivalent to the original relational algebra query.

Next, consider the following SQL query that returns the price of offers bough from a specified user (parameter $?_1$) from a seller that lives in a specified city (parameter $?_2$).

```
SELECT O.price
FROM offer O, user U, sale S
WHERE U.uid = ?1 AND
        S.customer_id = U.uid AND
        S.offer_id = O.oid AND
        O.seller_id IN (SELECT uid
                        FROM user
                        WHERE city = ?2);
```

Proceed in the following steps:

5. Translate this SQL query into a relational algebra query (that uses only relation names, selections, projections, renamings, cross products, and conditional joins). You are allowed to simplify the query if you see ways to do so.
6. Provide a *good* query execution plan for your relational algebra query. Explain why your plan is good.
7. Estimate the size of the output of all intermediate steps in your query execution plan. Explain each step in your estimation.

Assignment

Part 1 Write the above queries. Hand in a single plain-text file (`txt`) with each of the requested queries in the following format:

```
-- Query [number]
[the query]

-- Clarifications: [any description].


[next query]
```

In the above, `[number]` is the query number (1, 2, 3, 4). Your submission:

1. must include your student number and MacID at the top of the file as a comment, e.g.:
   `-- Student number: XXXXXXXX, MacID: YYYYYY;`
2. must *only* use the constructs presented on the slides or in the book (we do *not* allow any SQL constructs that are not on the slides or in the book);
3. must work on the provided example dataset when using IBM Db2 (on `cs2db3.cas.mcmaster.ca`): test this yourself!

Part 2 Write a `PDF` file in which you answer the above 7 items. Hand in a single `PDF` file in which each answer is clearly demarcated. Your submission must include your student number and MacID.

Submissions that do not follow the above requirements will get a grade of zero.

Grading

Part 1 counts for 60%, Part 2 counts for 40%. Each of the four queries in Part 1 count equally toward the final grade for this assignment. We grade the queries in Part 1 in the same was as we did for the SQL queries of Assignment 2. Each of the seven problems in Part 2 count equally toward the final grade for this assignment. You get the full marks on a problem if it solves the described problem exactly.