Assignment: SQL–The Structured Query Language

COMPSCI 2DB3: Databases–Winter 2024

Deadline: February 16, 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).

The requested queries (SQL)

1. *Find all self-reviews.*

   Marketplace websites are often abused by scammers that want to look legit. One way to look legit is by trying to increase the seller review score, e.g., by reviewing oneself: an indicator of foul play.

   Query: Write a query that returns a copy of the **sreview** table that only contains those reviews where sellers review themselves. Note that a seller can do so if the seller was selling products to themselves (the customer of the referenced sale is also the seller of that sale). Order the result on decreasing *score*.

2. *Profit-based sellers.*

   Some users are active on the marketplace to make money: they buy products for low prices, just to flip them around and sell them for higher prices.

   Query: Write a query that returns a list of user identifiers of all users that have bought a product and *after that sale* sold that same product for a higher price.

3. *Inactive users.*

   One way of scammers to seem legitimate is by using *bot* accounts to place fake reviews: they make an account and only use the account to only place reviews. Hence, the marketplace wants to be able to identify accounts that have not been used at all (besides reviewing products).

   Query: Write a query that returns a copy of the **user** table, except that the copy only keeps those users that have placed product reviews, but do not participate in any other way (they do not offer products, do not participate in sales, and do not review sales).

4. *Good offers.*

   A good offer for a product is an offer that is significantly cheaper than other offers for the same product.

   Query: Write a query that returns a copy of the **offer** table that keeps, for each product, those offers that are at-least 20% cheaper than the average price of the product.

5. *Emphasize participation.*

   One way to emphasize participation by users (by reviewing products and sales) is by ranking users on their *review completeness ratio*. In this, the review completeness of a user u is the ratio $\frac{x}{y}$ where y is the number of sales the user participated in and x is the number of those sales that received a review.

   Query: Write a query that returns triples (*user_id*, x, y) that provides all necessary information to compute the review completeness ratio for all users. Your query does *not* have to include users that did not participate in sales ($y = 0$), but *must* include users that participated in sales without ever reviewing ($x = 0, y > 0$).

6. *Find related products.*

   One way to recommend products to users is by looking at their history and comparing this to other users: if a user A bought all products that user B bought, then we can suggest user B to buy any products bought by A that user B did not yet buy.

   Query: Write a query that returns pairs (*uid*, *product*) of user identifiers (*uid*) of users B and products p such that user B did not yet buy product p, but some other user A did buy all products bought by B and also bought product p.

Assignment

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, 5, 6). 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!

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

Grading

Each of the seven problems count equally toward the final grade for this assignment. You get the full marks on a problem if it solves the described problem exactly. We take the following into account when grading:

1. Is the query correct (does it solve the stated problem)?

   Hint: Your queries must not only work on the provided example dataset in file (`sql_example.txt`), but also on all other valid datasets: think about edge cases (e.g., empty tables).

2. Are all required columns present?
3. Are no superfluous columns present?
4. Does the result satisfy the ordering requirements provided?
5. Does the result have unnecessary duplicate values (none of the queries should have duplicates in their output)?

6. Are no superfluous statements present (e.g., **DISTINCT** when the query cannot produce duplicates, **ORDER BY** if the query does not need to be ordered, **GROUP BY** a column that is already unique)?

If you are stuck and cannot solve a query, then provide a query showing the parts that you managed to solve and describe in your clarification what you managed to solve. We will give partial grades for solutions that are not complete, but do solve a *core component* correctly.