Assignment: The Relational Algebra

COMPSCI 2DB3: Databases–Winter 2023

Deadline: March 12, 2023

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 have completed the work for that assignment by yourself. 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

A local group of movie fanatics is working further on their review website. The group already has asked hundreds of experts to figure out which data can be extracted via SQL. Next, the group wants to further explore which data can be extracted from this schema and to explore the efficiency by which some data can be extracted. Following is the high-level description of the database schema under consideration:

- **user**(*uid*, *name*, *pwdhash*, *rname*).

  The **user** relation contains a unique user id, the username (which should be unique), and a password hash (*pwdhash*) used to verify login attempts. The user table also stores the real name for users (*rname*), which is used to credit the author of reviews and comments.

- **review**(*rid*, *uid*, *title*, *body*, *rtime*).

  The **review** relation contains the review articles written for the website. Each review has a unique identifier, the author identifier (*uid*, which refers to an author in the **user** relation), and the title of the article, the body of the article, and the date and time of the review (*rtime*).

- **comment**(*cid*, *rid*, *uid*, *body*).

  The **comment** relation contains comments on articles. Each comment has a unique identifier, the review identifier of the review the comment belongs to (*rid*, which refers to a review in the **review** relation), the user identifier of the user that wrote the comment (*uid*, which refers to a user in the **user** relation), and the body of the comment.

- **film**(*fid*, *title*, *year*).

  The **film** relation contains the films discussed in articles on the website. Each film has a unique identifier, a title, and the year of release.

- **fcategory**(*fid*, *category*).

  The **fcategory** relation relates films (*fid*, which refers to a film in the **film** relation) to the categories of this film (e.g., romcom, drama, thriller, horror, action).

- **fmention**(*rid*, *fid*, *score*).

  The **fmention** relation relates reviews (*rid*, which refers to a review in the **review** relation) to the possible-multiple films the article discusses (*fid*, which refers to a film in the **film** relation). Each discussed film receives a reviewer-score *score*.

Note that this relational schema is *similar* to the schema used for the second assignment: we have simplified the **user** relation and we have renamed **article** to **review**.

The requested queries

The movie fanatics are interested in the following queries:

1. *Find all films related to a comment*

   The search function of the website will include a feature to search comments by the films these comments might talk about. To be able to implement this feature, the movie fanatics ask for a query that relates comments with the films they are related to (via the review the comment is placed on and that mentions the film).

   Query: Write a query that returns pairs (*cid*, *fid*) in which *cid* is a comment identifier of a comment on a review that mentions the film with film identifier *fid*.

2. *Find all films mentioned in multiple reviews*

   Popular films typically are reviewed multiple times. Hence, the movie fanatics want to include a filter on the website to search for films with multiple reviews.

   Query: Write a query that returns film identifiers (*fid*) of all films that are mentioned in multiple reviews.

3. *Find all properly-categorized films*

   Most films do not neatly fit in exactly one category. At the same time, it is easy to go overboard and put a film in too many categories. As a rule of thumb, the movie fanatics attempt to assign each film to the two most-significant categories for that movie.

   Query: Write a query that returns film identifiers (*fid*) of all films f that follow the rule of thumb: the film f is assigned to exactly two categories.

4. *Find meaningless categories*

   The movie fanatics have decided that film categories are only meaningful for the website if any of the reviewers actually review films in these categories. To keep the website clean, the movie fanatics want to determine which film categories are meaningless.

   Query: Write a query that returns names of *categories* for which no review mentions a film that is part of that category.

5. *Find latest follow-up reviews*

   Most reviews are written around the release of a film and merely serves as an opinion on whether it is worthwhile to see the film. Sometimes, films get more in-depth follow-up reviews, however. Typically, such follow-up reviews are written for films with a broader cultural impact (e.g., great films). Hence, the movie fanatics want a filter to search only for such follow-up reviews.

   Query: Write a query that returns tuples (*rid*, *uid*, *fid*) such that *rid* is the review identifier of the latest review r written by the author a with author identifier *uid* such that r mentions the film f with film identifier *fid* and r is the latest *follow-up review* for f written by a: there exists another review s written by author a such that both r and s mention the film f and s was written before r. Note that we are only looking for the *latest* follow-up review by author a for f: there must not be any other review written after review r by author a that mentions the film f.

6. *Find popular films*

   Naturally, most users are often heavily interested in *popular films* on which everyone seems to have opinions. To find such films, one has to look for films that have been reviewed by *all reviewers*.

   Query: Write a query that returns a list of film identifiers *fid* of *popular films*. A film is a popular film if it has been reviewed by all reviewers. We note that the **user** table does not only include reviewers: most users only comment, and only the users that ever wrote a review (those that are authors in the **review** table) are considered reviewers.

Efficiency of queries

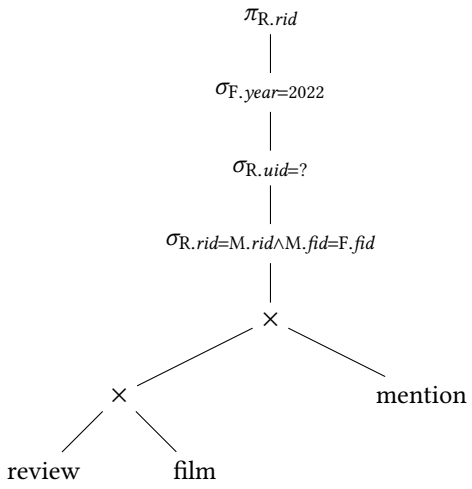
The movie fanatics are worried about whether some of the complex queries can be implemented efficiently, especially regarding two complex queries. To aid the movie fanatics, we will provide estimates of the complexity of these two queries in terms of the size of intermediate query evaluation results.

To allow us to do so, the movie fanatics already provided the following rough estimates of the involved data: there are 50 reviewers among the 10 000 users in **user**, each reviewer is expected to write 20 reviews per year, each review will receive up-to-a-1000 comments, there are 700 films released per year, on average each film belongs to 3 categories, and typical reviews mention two films.

The first query the movie fanatics worry about is used to make year-overviews: for a specified reviewer (parameter ?), this query returns all films released in the last year reviewed by the reviewer. For this query, we already wrote the following relational algebra query:

$$\pi_{\mathrm{R}.rid}(\sigma_{\mathrm{F}.year=2022}(\sigma_{\mathrm{R}.uid=?}(\sigma_{\mathrm{R}.rid=\mathrm{M}.rid\wedge\mathrm{M}.fid=\mathrm{F}.fid}(\rho_R(\mathrm{review}) \times \rho_M(\mathrm{fmention}) \times \rho_F(\mathrm{film}))))).$$

The consultant expects that this query will be evaluated via the following query execution plan:



To gain insight into the performance of this query, the movie fanatics asked us to figure out how costly this evaluation is *assuming the website has already been operated for ten years* and to improve on this plan. To do so, proceed in the following steps:

7. Estimate the size of the output of all intermediate steps in the query execution plan provided by the consultant.
8. Provide a *good* query execution plan for the above relational algebra query. Explain why your plan is good.
9. Estimate the size of the output of all intermediate steps in your query execution plan. Explain each step in your estimation.
10. Finally, also provide an SQL query equivalent to the original relational algebra query.

The second query the movie fanatics worry about is the following SQL query that obtains the *films* in a specific category (parameter $?_1$) and reviewed by a user (parameter $?_2$):

```
SELECT DISTINCT F.fid
FROM user U, film F, review R, mention M
WHERE U.uid = ?2 AND
        U.uid = R.uid AND R.rid = M.rid AND
        R.rid = M.rid AND M.fid = F.fid AND
        M.fid IN (SELECT C.fid
                  FROM fcategory F
                  WHERE F.category = ?1);
```

Unfortunately, the movie fanatics are not convinced whether this query is optimal and could be executed efficiently. To gain some insight into the performance of this query, the movie fanatics asked us to figure out how this query could be evaluated in practice. To do so, proceed in the following steps:

11. 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.
12. Provide a *good* query execution plan for your relational algebra query. Explain why your plan is good.
13. Estimate the size of the output of all intermediate steps in your query execution plan. Explain each step in your estimation.

Assignment

Write a document in which you answer the above 13 items. Hand in a single PDF file in which each answer is clearly demarcated. E.g.,

> **1.** *your relational algebra query that finds all films related to a comment.*
> **2.** *your relational algebra query that finds all films mentioned in multiple reviews.*
> ...
> **13.** *your estimate of the output size of all intermediate steps in your query execution plan.*

Your submission:

1. must be typed (e.g., generate a PDF via LaTeX or via your favorite word processor): handwritten documents will not be accepted or graded;
2. all relational algebra queries must use the basic relational algebra (with set semantics) as summarized on the slide "A formal grammar of the relational algebra" (Slide 8 of the slide set "The Relational Algebra");
3. all SQL queries must use the constructs presented on the slides or in the book (we do *not* allow any system-specific constructs that are not standard SQL);
4. all SQL queries must work on the provided example dataset when using DB2 (on `db2srv2`): test this yourself!

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

Grading

Part 1 (items 1–6 in "The requested queries") will receive 60% of the grade, equally divided over all queries. The first query of Part 2 (items 7–10 in "Efficiency of queries") will receive 20% of the grade, and the last query of Part 2 (items 11–13 in "Efficiency of queries") will receive 20% of the grade.

The SQL query of item 10 is graded the same as all SQL queries of Assignment 2. For the relational algebra queries, you get the full marks on a query 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)?
2. Are all required columns present?
3. Are no superfluous columns present?

If you are stuck and cannot solve a query, then provide a query showing the parts that you managed solve and describe in your clarification what you managed to solve.

For all other items (items 7–9 and items 12 and 13) we will look at whether your solution is correct, how close to optimal your solution is, and whether it is complete (is a proper explanation included).