Assignment 4: The Relational Algebra

Jelle Hellings

3DB3: Databases – Fall 2021

Department of Computing and Software
McMaster University

Deadline: November 5, 2021

Cheating and plagiarism. 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. 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 the submitted work.

Cheating and plagiarism are serious academic offenses. All cases of academic dishonesty will be handled in accordance with the Academic Integrity Policy via the Office of Academic Integrity.

Late submission policy. There is a late penalty of 20% on the score per day after the deadline. Submissions five days (or later) after the deadline are not accepted. Do not wait until the deadline to ask questions or raise problems.

Description

Our local community leader is further tinkering with the concept of an event website for advertising local events. The leader already worked with a consultant to construct a detailed relational schema that describes how all necessary information is stored. Next, the leader wants to further explore which data can be extracted from this schema and to explore the efficiency by which some queries can be executed. The relational schema consists of the following five relations:

- **user**(uid, name, regdate, postcode).

  The **user** relation contains the user name, registration date (*regdate*), and an optional postal code (*postcode*) of registered users of the website. The postal code will be used to recommend each user with events in its area.

- **event**(eid, title, description, startdate, enddate, organizer, postcode).

  The **event** relation contains the title (or name), description, start and end date, organizer, and postal code (*postcode*) of events. The *organizer* is stored by a user identifier (a value in the column *uid* of the table **user**).

  Events can be organized in other regions than where their organizer is based. Hence, the event postal code does not have to match the postal code of the organizer.

- **keyword**(event, word).

  Each event can be labeled with zero-or-more keywords that describe the event and can be used as search criteria. E.g., a guided tour of a museum with an exhibition on a famous painter could be labeled with: ‘art’, ‘painting’, and ‘exhibition’. The *event* is stored by an event identifier (a value in the column *eid* of the table **event**).

- **region**(postcode, name).

  To simplify searches based on regions (e.g., the golden horseshoe area), postal codes (*postcode*) are mapped to the commonly-used names of regions they are part of. Note that a postal code can be part of many regions and a region can have many associated postal codes.

- **review**(user, event, description, score, reviewdate).

  All users can review events. Such a review consists of a description, a score (0-10), and the date at which the review was written. The *user* is stored by a user identifier (a value in the column *uid* of the table **user**) and the *event* is stored by an event identifier (a value in the column *eid* of the table **event**).

Note that this relational schema is equivalent to the schema used for the second assignment.

The requested queries

The community leader is interested in the following queries:

1. High-level query: 'Find all multi-day events'. Detailed description:

   The community leader wants to include filters on the website to search based on the length of events (e.g., to distinguish between a concert and multi-day festivals).

   Write a query that returns a copy of the event table that only contains those events that are multi-day events.

2. High-level query: 'Find neighborhood events'. Detailed description:

   The community leader wants to assure that all users will easily find the events in their direct neighborhood. To do so, the community leader wants to assure that users always get recommended all events that are organized in their own postal code.

   Write a query that returns pairs (*user*, *event*) in which *user* is a user identifier and *event* is an event identifier such that the user and the event share the same postal code.

3. High-level query: 'Find all unreviewed events'. Detailed description:

   The community leader wants to promote events that have not yet been reviewed, this to encourage participation in new events and encourage reviewing.

   Write a query that returns a copy of the event table that only contains those events that do not have reviews.

4. High-level query: 'Find all optimally-annotated events'. Detailed description:

   According to the community leader, events with *exactly three* keywords have the highest engagement. To aid event organizers, the community leader wants to label these optimally-annotated events.

   Write a query that returns a list of all events (column *eid* of the **event** table) that have exactly three keywords.

5. High-level query: 'Find the most recent activity'. Detailed description:

   To drive engagement with user profiles, the community leader wants to list the most recent activities of users on their profile. In specific, the community leader wants to add the most-recent review and the review of the most-recent event to the user pages of people that reviewed events.

   Write three queries:

(a) a query that returns pairs (*user*, *event*) in which *user* is a user identifier and *event* is the event identifier of the event for which the user wrote a review most-recently (according to *reviewdate*);

(b) a query that returns pairs (*user*, *event*) in which *user* is a user identifier and *event* is the event identifier of the most-recent event (according to *enddate*) for which the user wrote a review;

(c) a query that returns triples (*user*, *lreview*, *levent*) in which *user* is a user identifier. *lreview* is the event identifier of the event for which the user wrote a review most-recently, and *levent* is the event identifier of the most-recent event for which the user wrote a review.

6. High-level query: ‘Find postal code experts’. Detailed description:

    The event website is community-driven and will only succeed with enthusiastic participation of its users. To further encourage such participation, the community leader wants to hand out *postal code badges* to all users that reviewed all events organized in their postal code.

    Write a query that returns all users (column *uid* from the **user** table) that have reviewed all events that are organized in their postal code.

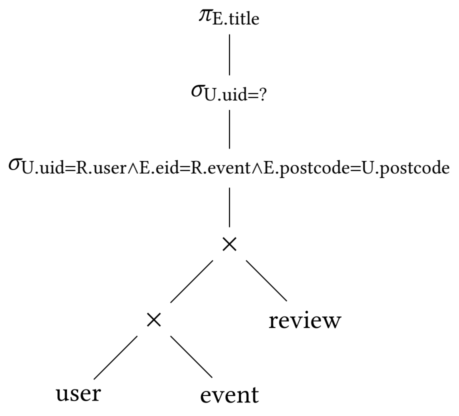
Efficiency of queries

The consultant that originally advised our local community leader is worried about whether some of the complex queries can be implemented efficiently. The consultant has specific concerns about two such queries. To aid the consultant, we will be providing estimates of the complexity of these two queries in terms of the size of intermediate query evaluation results. To do so, the consultant provided the following rough estimates of the involved data: there are 1000 rows in **user**, 5000 rows in **event**, 25000 rows in **keyword**, 5000 rows in **region**, and 25000 rows in **review**. Furthermore, the consultant estimated that all users write an equal amount of reviews (25 reviews per user), that each event has an equal amount of reviews (5 reviews per event), that each event has an equal amount of keywords (5 keywords per event), that there are 50 distinct regions in **region**, and that there are 1250 distinct postal codes in **region** (each associated with 4 regions),

The first query the consultant worries about returns the titles of events a specified user (parameter ?) reviewed in their own postal code. For this query, the consultant already wrote the following relational algebra query:

$$\pi_{\text{E.title}}(\sigma_{\text{U.uid=?}}(\sigma_{\text{U.uid=R.user}\land\text{E.eid=R.event}\land\text{E.postcode=U.postcode}}(\rho_U(\text{user}) \times \rho_E(\text{event}) \times \rho_R(\text{review})))).$$

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 consultant asked us to figure out how costly this evaluation is 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 consultant worries about is the following SQL query that obtains the *keywords* of events a specified user (parameter $?_1$) has reviewed in a specified region (parameter $?_2$):

```
SELECT DISTINCT K.word
FROM user U, review Rv, keyword K
WHERE U.uid = ?1 AND
        Rv.user = U.uid AND
        Rv.event = K.event AND
        K.event IN (SELECT E.eid
                    FROM event E, region Rg
                    WHERE E.postcode = Rg.postcode AND
                            Rg.name = ?2);
```

Unfortunately, the consultant was not convinced whether this query is optimal and could be executed efficiently. To gain some insight into the performance of this query, the consultant 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 multi-day events.*
2. *your relational algebra query that finds neighborhood events.*
...
13. *your estimate of the output size of all intermediate steps in your query execution plan.*

Include separate entries for 5a, 5b, and 5c.

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).