University of Toronto
CSC343, Fall 2021

Assignment 1

Declare your group on MarkUs (even if working alone): as soon as you begin working together
Assignment due: Thursday, October 14th at 8:00 pm sharp!

Learning Goals

By the end of this assignment you should be able to:

- Read a new relational schema and determine whether or not a particular instance is valid with respect to that schema.
- Apply the individual techniques for writing relational algebra queries and integrity constraints that we learned in class.
- Combine the individual techniques to solve complex problems.
- Identify problems that cannot be solved using relational algebra.

These skills will leave you well prepared to be a strong SQL programmer.

In this course, you will submit your assignments using the online tool MarkUs. For those of you not familiar with it, MarkUs is "an open-source tool which recreates the ease and flexibility of grading assignments with pen on paper, within a web application. It also allows students and instructors to form groups, and collaborate on assignments." (reference: `http://markusproject.org/`). For this assignment, the domain you will be working on is itself MarkUs. You will write queries and integrity constraints on a (simplified) schema for the data that MarkUs stores.

Schema

Relations

- User(<u>userName</u>, lastName, firstName, type)
  A tuple in this relation represents a MarkUs user. *userName* is their cdf user name, *lastName* and *firstName* are obvious, and *type* is the kind of MarkUs user they are. (The possible values for *type* are defined in an integrity constraint below.)
- Assignment(<u>aID</u>, description, due, groupMin, groupMax)
  A tuple in this relation represents an assignment or other course item that is submitted on MarkUs, such as a test, project, or quiz. *aID* is the assignment ID, *description* is a short description of the assignment, *due* is the date and time when it is due, *groupMin* is the minimum number of students who may work together on the assignment, and *groupMax* is the maximum number of students who may work together on the assignment.
- Required(<u>aID, fileName</u>)
  A tuple in this relation represents the fact that submitting a certain file is required for a certain assignment. *aID* is the assignment ID and *fileName* is the name of the required file.
- Group(<u>gID</u>, aID, repo)
  A tuple in this relation represents a group. *gID* is the group ID, *aID* is the assignment of the ID for

which this group exists, and *repo* is the URL of the shared repository where their submitted files are stored.

- Membership(<u>userName, gID</u>, status)
  A tuple in this relation represents that a user belongs to a group. *userName* is their cdf user name, *gID* is the group to which they belong, and *status* records information about which member invited which to the group and whether an invitation was accepted (see the constraints below for more information).

- Submission(<u>sID</u>, fileName, userName, gID, when)
  A tuple in this relation represents the fact that a file was submitted. *sID* is the submission ID, *fileName* is the name of the file that was submitted, *userName* is the user name of the user who submitted the file, *gID* is the group ID of the group to which the assignment was submitted, *when* is the date and time when the file was submitted. The set of attributes {*fileName, userName, when*} is also a key.

- Grader(<u>gID</u>, userName)
  A tuple in this relation represents the fact that a user was assigned to grade a group. *gID* is the ID of the group and *userName* is the user name of the user who was assigned to grade them.

- Result(<u>gID</u>, mark, released)
  A tuple in this relation represents a mark for a group. *gID* is the group ID of the group who was given the mark, *mark*, a number, is the mark they got, and *released* is a boolean indicating whether or not the result has been made available for the group members to see on MarkUs.

- Tag(<u>gID, text</u>)
  A tuple in this relation represents the fact that a tag was applied to a group by their grader. *gID* is the ID of the group, and *text* is the content of the tag.

Integrity constraints

- Required[aID] $\subseteq$ Assignment[aID]
- Group[aID] $\subseteq$ Assignment[aID]
- Membership[userName] $\subseteq$ User[userName]
- Membership[gID] $\subseteq$ Group[gID]
- Membership[status] $\subseteq$ {"inviter", "invited", "accepted"}
- Submission[userName] $\subseteq$ User[userName]
- Submission[gID] $\subseteq$ Group[gID]
- Grader[gID] $\subseteq$ Group[gID]
- Grader[userName] $\subseteq$ User[userName]
- Result[gID] $\subseteq$ Group[gID]
- Tag[gID] $\subseteq$ Group[gID]
- Π_{type}User $\subseteq$ {"instructor", "TA", "student"}
- Π_{type}(Membership $\bowtie$ User) $\subseteq$ {"student"}
- Π_{type}(Grader $\bowtie$ User) $\subseteq$ {"TA", "instructor"}
- $\sigma_{\text{groupMin}<0 \ \vee \ \text{groupMin}>\text{groupMax}}$Assignment $= \emptyset$

- $\sigma_{\text{G1}\neq G2}(\rho_{\text{M1(userName, G1, aID)}}(\text{Membership} \bowtie \Pi_{gID,aID}\text{Group}) \bowtie \rho_{\text{M2(userName, G2, aID)}}(\text{Membership} \bowtie \Pi_{gID,aID}\text{Group})) = \emptyset$
- $(\Pi_{\text{userName,gID}}\text{Submission}) - (\Pi_{\text{userName,gID}}\text{Membership}) = \emptyset$

Warmup: Getting to know the schema

To get familiar with the schema, ask yourself questions like these (but don't hand in your answers): According to the schema,

- Can the same file be submitted more than once by the same group on the same assignment?
- Can a file be submitted for an assignment after the due date?
- Are group IDs unique across each individual assignment or across all assignments?
- How many different graders can mark an assignment? How many can mark a single group's work on an assignment?
- What does each integrity constraint mean in plain English?

Part 1: Queries

Write the queries below in relational algebra. There are a number of variations on relational algebra, and different notations for the operations. You must use the same notation as we have used in this course. You may use assignment, and the operators we have used in class: $\Pi, \sigma, \bowtie, \bowtie_{condition}, \times, \cap, \cup, -, \rho$. Assume that all relations are sets (not bags), as we have done in class, and do not use any of the extended relational algebra operations from Chapter 5 of the textbook (for example, do not use the division operator).

Some additional points to keep in mind:

- Do not make any assumptions about the data that are not enforced by the original constraints given above. Your queries should work for any database that satisfies those constraints.
- Assume that every tuple has a value for every attribute. For those of you who know some SQL, in other words, there are no null values.
- Remember that the condition on a select operation may only examine the values of the attributes in one tuple, not whole columns. In other words, to use a value (other than a literal value such as 100 or "Adele"), you must get that value into the tuples that your select will examine.
- The condition on a select operation can use comparison operators (such as $\leq$ and $\neq$) and boolean operators ($\vee$, $\wedge$ and $\neg$). Simple arithmetic is also okay, *e.g.*, attribute1 $\leq$ attribute2 + 5000.
- In your select conditions, you can compare date-time attributes using comparison operators such as $<$.
- You are encouraged to use assignment to define intermediate results.
- The order of the columns in the result doesn't matter.
- If there are ties, report all of them.

At least one of the queries and/or integrity constraints in this assignment cannot be expressed in the language that you are using. In those cases, simply write "cannot be expressed". Note: The queries are not in order according to difficulty.

1. *Extreme grades:* For each assignment find the highest and the lowest grade any group received. Report the assignment ID, highest grade, and lowest grade. If an assignment has no grades, it won't appear in the result.
2. *Procrastinators:* For each assignment (that has at least one required file) and each group (that has more than one member), find the group member whose earliest submission of a required file came latest among the group members. Report the assignment ID, group ID, and userName. Groups that have submitted no required file on an assignment will not be reported for that assignment.
3. *Good students:* Find every student who meets the following conditions: (a) they have a grade on all the assignments, (b) their grades are all at least 70, and (c) if we consider assignments in order by their due date, their grades have never gone down. Report the userName of these students.
4. *Well-used tags:* Let's say that a tag is "well used" on an assignment if at least 3 different TAs used that tag for at least 3 different groups. Find the assignment whose number of well used tags was the highest. Report the assignment ID and assignment description. If there was a tie for highest, report them all.
5. *Different group sizes:* Find every student who has worked in a group of three (three including themself) on some assignment, a group of two on some assignment (two including themself), and has also worked alone on some assignment. Report their username, highest mark on a group-of-three assignment, highest mark on a group-of-two assignment, and highest mark on a group-of-one assignment.
6. *Consistent groups:* Find all pairs of groups that, for two different assignments, have the exact same students in them, and earned the exact same mark. Report the two group IDs, the two assignment IDs, and the mark. For example, you might report that group 7249 on Assignment 2 and group 8116 on Assignment 4 had the same group members and earned the same grade of 84. Do not include the same pair of groups twice as pseudo-duplicates (*e.g.,* Do not include both (7249, 8116) and (8116, 7249))
7. *Prolific TA:* For each assignment due in 2020, find the TA who was grader for the highest number of groups of all TAs. Report the TA, the assignment, and the highest grade that TA has given on the assignment. If there are multiple TAs tied for grading the most groups on an assignment, report them all. You may assume that for all assignments due in 2020, every group has received a grade.
8. *Worked with top student:* Find all students who've been in a group with a student who has the highest mark in the entire database. (We said "a student" rather than "the student" because there could be several students tied for highest mark; we are looking for students who've worked with any of them.) For each of them, report their user name, their grade on the last assignment (last according to due dates), and the number of *different* students they have been in a group with.

Part 2: Additional Integrity Constraints

Express the following integrity constraints with the notation $R = \emptyset$, where R is an expression of relational algebra. If a constraint cannot be expressed using this notation, simply write "cannot be expressed". You are welcome to define intermediate results with assignment and then use them in an integrity constraint.

1. A grader cannot be assigned to mark the same student for more than one assignment.
2. Students are not allowed to "overinvite": You can't invite more people to join your group for an assignment than the maximum number of students allowed to work together on that assignment.
3. A group who submits one or more files after the deadline cannot receive a mark over 80 unless the grader is an instructor.

When writing your queries for Part 1, don't assume that these additional integrity constraints hold.

Formatting instructions

Your assignment must be typed; handwritten assignments will not be marked. You may use any word-processing software you like. Many academics use LaTeX. It produces beautifully typeset text and handles mathematical notation well. If you would like to learn LaTeX, there are helpful resources online. Many people use `overleaf.com` to do their LaTeX work in the cloud. It also makes co-editing a document with your partner easy. If you want to work locally, `TeXShop` is a good option.

Whatever you choose to use, you need to produce a final document in pdf format.

If you use software that lets you choose a font size, it must be at least 10. If you use LaTeX, the default font size (or larger) is acceptable.

Submission instructions

You must declare your team (whether it is a team of one or two students) and hand in your work electronically using the MarkUs online system. Instructions for doing so are posted on the Assignments page of the course website. Well before the due date, you should declare your team and try submitting with MarkUs. You can submit an empty file as a placeholder, and then submit a new version of the file later (before the deadline, of course); look in the "Replace" column.

For this assignment, hand in just one file: A1.pdf. If you are working in a pair, only one of you should hand it in.

Check that you have submitted the correct version of your file by downloading it from MarkUs; new files will not be accepted after the due date.

How your assignment will be marked

Most of the marks will be for the correctness of your answers. However, there will be additional marks allocated to each of these:

- Comments:
  Does every assignment statement have a comment above it specifying clearly exactly what rows get to be in this relation? Comments should describe the data (*e.g.*, "The student number of every student who has passed at least 4 courses.") not how to find it (*e.g.*, "Find the student numbers by self-joining ..."). Put comments *before* the assignment, and two dashes on each line of your comment.
- Attribute names given on the LHS:
  Does every assignment statement name the attributes on the LHS? This should be done even if the names are not being changed. Together with the comments, it allows you to understand what a "subquery" is supposed to do without reading it. Think of this as analogous to good parameter names and good comments on a function.
- Relation and attribute names:
  Does every relation and every attribute have a name that will assist the reader in understanding the query quickly? Apply the same high standards you would when writing code.
- Formatting:
  Is the algebra formatted with appropriate line breaks and indentation to reveal the structure of the algebra for ease of understanding?

Final advice

These are my top pieces of advice for doing a great job of A1, painlessly:

- Have the summary of specific techniques beside you like a "cheat sheet" (and have mastered understanding each one)
- Make a concrete instance of the relevant relations and what the result of the query should be for this instance. Write it down. Think like a computer scientist and make sure it tests out a few good conditions.
- Write down the LHS of the last step. Name the relation and attributes well and write a comment explaining what it takes to get into that relation. Don't bother with the RHS yet. Then imagine some intermediate result that would make that last step easy. Don't worry about how to create it, just write down the LHS and the comment. Keep reasoning backwards. Don't write down any RHSs at all — no algebra! — until you have the whole thing broken down.
- Leave plenty of time for typing up your answers; the formatting will take longer than you may realize.