# CSCC43 UTSC

## Assignment 1

**Points: 85**

**Release Date: 27th September**

**Due Date: 9th October, 11:59pm**

**Submission Platform: MarkUs**

**Learning Goals:**

By the end of this assignment, you should be able to:
1. Read a new relational schema, and determine whether a particular instance is valid with respect to that schema,
2. Apply the individual techniques for writing relational algebra queries and integrity constraints that we learned in class,
3. Combine the individual techniques to solve complex problems, and
4. identify problems that cannot be solved using relational algebra.

These skills we leave you well prepared to learn SQL.

**Instructions:**

Write (**type; do not handwrite**) the queries below in relational algebra. There are several variations on relational algebra, and different notations for the operations. You must use the same notation as we have used in class and on the slides. You may use assignment, and the operators we have used in class: $\pi$, σ, ⋈, $\bowtie_{\theta}$, ×, ∩, ∪, −, ρ, /. **Assume that all relations are sets (not bags)**, as we have done in class, and **do not use any of the extended relational algebra**.
Some additional points to keep in mind:

- Do not make any assumptions about the data that are not enforced by the original constraints given, including the ones written in English. 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.
- The condition on a select operation can use comparison operators (such as ≤ and ≥) and Boolean operators (∨, ∧ and ¬). Simple arithmetic is also okay, e.g., attribute1 ≤ attribute2 + 5000.
- You are encouraged to use assignment to define intermediate results.
- It's a good idea to add commentary explaining what you're doing. This way, even if your final answer is not completely correct, you may receive part marks.
- The order of the columns in the result does not matter.
- When asked for a maximum or minimum, if there are ties, report all of them.

## Schema 1

For this problem we use the schema of the Health Network database. It includes the following relations. Keys are underlined and each tuple is described.

- Clinic (clinicID, hName, hAddress, hCity)
  - A clinic including the $clinic\ ID$, $name$, $address$ and $city$ of that clinic.
- Room (roomID, clinicID, rNumber, rSize)
  - A clinic room including the clinic ID, room number, and room capacity – the number of patients it accommodates.
- Employee (employeeID, eFirstName, eLastName, eRole, eSalary, eAddress, eCity, eSIN)
  - An employee including their employee ID number, first and last names, role in the clinic (or occupation type, such as physician, technician, etc.), salary, address, city, and Social Insurance Number.
- Department (departmentID, clinicID, dName, dHead)
  - A department in a clinic including the clinic ID, the department ID, the name of the department, and the employee ID of the head of the department.
- EmployeeDepartment (employeeID, departmentID, edStartDate, edEndDate)
  - The departments to which an employee belongs. An employee may work in more than one department. This tuple contains the employee ID, the department ID, and the dates that they started and finished employment in that department.
- Patient (patientID, pFirstName, pLastName, pSex, pOHIP, pOther, pDOB, pAddress, pCity, pProvince, pCountry)
  - A patient in the clinic system including the patient ID, first name, last name, sex (M or F), OHIP number, other form of payment if not on OHIP, date of birth, residence address including the address, city, province, and country.
- PatientAppointment (patientID, departmentID, employeeID, paDate, paTime, paReason)
  - An appointment for a patient, including the department ID of the clinic department, the employee ID of the person whom they are seeing (could be technician), the date, the time, and the purpose of the visit.
- PatientRoomAssignment (patientID, roomID, prInDate, prOutDate, prPhysician)
  - A room that is assigned to a patient who needs to stay in the clinic - including the ID of the patient and room, the date that they were admitted, the date that they left the clinic, and the employee ID of their attending physician during their stay.

**Integrity Constraint**

- Room (clinicID) ⊆ Clinic (clinicID)
- Department (clinicID) ⊆ Clinic (clinicID)
- Department (dHead) ⊆ Employee (employeeID)
- EmployeeDepartment (departmentID) ⊆ Department (departmentID)
- PatientAppointment (patientID) ⊆ Patient(patientID)
- PatientAppointment (departmentID) ⊆ Department (departmentID)
- PatientAppointment (employeeID) ⊆ Employee (employeeID)
- PatientRoomAssignment (patientID) ⊆ Patient (patientID)
- PatientRoomAssignment (roomID) ⊆ Room (roomID)

## Queries: (5 points each)

1. Find the clinic name and department name for all departments whose head is also head of another department.
2. Find the first name and last name of all Physicians whose patients are never in the clinic for less than 10 days.
   (They must have had patients admitted in the clinic at some time.)
3. Find the first and last name of all patients who have exactly two physicians in the clinic's system.
   (Note that physicians are specified in both appointments and room assignment.)
4. Find the clinic and department name for the department at which there were no appointments from April 3, 2019 to April 8, 2019.
5. Find the clinic, room number, and patient first and last names of all semi-private (size = 2) rooms which were occupied by female patients who were over 50 on May 13, 2019.

## Schema 2

The following is a database for *OurTube*, an online video and movie streaming service.

**Date Time Operations**

In this database, you will be working with datetime fields that are formatted like so: YYYY-MM-DD HH:MI:SS. You may get only the date portion of such a field by doing *datetime.date* which will be formatted: YYYY-MM-DD. You may also just get only the time portion with *datetime.time* which will be formatted: HH:MI:SS. You may also do comparison (<, >, =, !=, etc.) between datetimes, dates, and times e.g: ‘2019-12-31’ < ‘2020-01-01’ is true. You may also do arithmetic with datetimes, dates, and times. If you subtract datetimes or times, you will get a value representing the number of seconds between them e.g: ‘2020-01-01 12:01:40’ – ‘2020-01-01 12:00:00’ = 100. If you subtract dates, you will get a value representing the number of days between them e.g: ‘2020-01-02’ – ‘2020-01-01’ = 1. You may also add or subtract constants to these types of fields and they will be treated similarly e.g: ‘2020-01-01 12:00:00’ + 1 = ‘2020-01-01 12:00:01’ or ‘2020-01-02’ – 1 = ‘2020-01-01’.

**Relations**

- User(uid, uname, email, phone, date_joined)
    - A tuple in this relation represents an OurTube user. *uid* is the user’s id as an integer and *uname, email,* and *phone* are strings. *date_joined* is a datetime of when the user created their account.
- Subscription(subscriber, subscribed, date_subscribed)
    - A tuple in this relation represents an OurTube user subscribing to another OurTube user. *subscriber* is the uid of the subscriber. *subscribed* is the uid of the user that *subscriber* is subscribing to. *date_subscribed* is a datetime of when *subscriber* subscribed to *subscribed*.
- Video(vid, title, description, duration, uploader, date_uploaded)
    - A tuple in this relation represents an OurTube video. *vid* is the video’s id as an integer and *title* and *description* are strings. *duration* is an integer representing how long the video is in **seconds**. *uploader* is the uid of the user who uploaded the video. *date_uploaded* is a datetime of when the video was uploaded.
- WatchVideo(uid, vid, date_watched)
    - A tuple in this relation represents when an OurTube user watched an OurTube video. *uid* is the user, *vid* is the video, *date_watched* is a datetime of when the video was watched. **Note that a user can watch the same video multiple times.**
- LikeDislikeVideo(uid, vid, date, value)
    - A tuple in this relation represents an OurTube user liking or disliking an OurTube video. *uid* is the user, *vid* is the video, *date* is a datetime of when the video was liked or disliked. *value* is an integer that takes on one of two values: 1 or -1 to represent whether the video was liked or disliked respectively.

- Playlist(pid, pname, privacy, uid, date_created)
  - A tuple in this relation represents an OurTube playlist created by an OurTube user. *pid* is the playlist's id as an integer and *pname* is a string. *privacy* is a string that takes on one of two values: 'public' or 'private'. *uid* is the user that created this playlist. *date_created* is a datetime of when the playlist was created.
- PlaylistItem(pid, vid, date_added)
  - A tuple in this relation represents an OurTube video belonging to an OurTube playlist. *pid* is the playlist, *vid* is the video, *date_added* is a datetime of when the video was added to the playlist. **Note that the same video can appear in the same playlist multiple times.**
- Movie(mid, title, description, duration, rating, provider, price, date_released)
  - A tuple in this relation represents an OurTube movie. *mid* is the movie's id as an integer and *title* and *description* are strings. *duration* is an integer of how long the movie is in **seconds**. *rating* is the rating of the movie according to the Motion Picture Association film rating system and is a string that takes on one of five values: 'G', 'PG', 'PG13', 'R', or 'NC17'. *provider* is a string that represents the provider of the movie, examples include 'Disney' or 'Warner Bros. Entertainment'. *price* is a float that represents the price a user must pay to gain access to watching the movie. *date_released* is a datetime of when the movie was released. **Movies that are free to watch are represented with a price of 0.**
- WatchMovie(uid, mid, date_watched)
  - A tuple in this relation represents when an OurTube user watched an OurTube movie. *uid* is the user, *mid* is the movie, *date_watched* is a datetime of when the movie was watched. **Note that a user can watch the same movie multiple times.**
- PurchaseMovie(uid, mid, date_purchased)
  - A tuple in this relation represents when an OurTube user purchased an OurTube movie. *uid* is the user, *mid* is the movie, *date_purchased* is a datetime of when the movie was purchased.
- ReviewMovie(uid, mid, date_reviewed, description, stars)
  - A tuple in this relation represents an OurTube user leaving a review for an OurTube movie. *uid* is the user, *mid* is the movie. *date_reviewed* is a datetime of when the review was written. *description* is a string which is the details of the review. *stars* is an integer that takes one of five values: 1, 2, 3, 4, or 5.
- Genre(gid, tag, description)
  - A tuple in this relation represents a genre that any OurTube movie stored in the database could be tagged with. *gid* is the genre's id as an integer, *tag* is a string, *description* is a string and is a description of the genre.
- MovieGenre(mid, gid)
  - A tuple in this relation represents an OurTube movie being tagged with a specific genre. *mid* is the movie and *gid* is the genre. **Note that a movie can be tagged with multiple genres.**

**Integrity Constraints**

- Subscription[subscriber] ⊆ User[uid]
- Subscription[subscribed] ⊆ User[uid]
- Video[uploader] ⊆ User[uid]
- WatchVideo[uid] ⊆ User[uid]
- WatchVideo[vid] ⊆ Video[vid]
- LikeDislikeVideo[uid] ⊆ User[uid]
- LikeDislikeVideo[vid] ⊆ Video[vid]
- LikeDislikeVideo[value] ⊆ {-1, 1}
- Playlist[uid] ⊆ User[uid]
- Playlist[privacy] ⊆ {'public', 'private'}
- PlaylistItem[pid] ⊆ Playlist[pid]
- PlaylistItem[vid] ⊆ Video[vid]
- Movie[rating] ⊆ {'G', 'PG', 'PG13', 'R', 'NC17'}
- WatchMovie[uid] ⊆ User[uid]
- WatchMovie[mid] ⊆ Movie[mid]
- PurchaseMovie[uid] ⊆ User[uid]
- PurchaseMovie[mid] ⊆ Movie[mid]
- ReviewMovie[uid] ⊆ User[uid]
- ReviewMovie[mid] ⊆ Movie[mid]
- ReviewMovie[stars] ⊆ {1, 2, 3, 4, 5}
- MovieGenre[mid] ⊆ Movie[mid]
- MovieGenre[gid] ⊆ Genre[gid]

## Part 1: Queries (5 points for each query)

1. For each pair of users that are subscribed to each other, find the videos that they both like and were uploaded by neither of them. Report only one row for each video the pair likes. Report back user1id, user2id, vid.
2. For each user, report their playlists that do not contain any videos they dislike. Report back uid, uname, pid, pname.
3. For each user, report their public playlists that contain only videos they've liked, and the uploaders are users they are subscribed to. These playlists should also contain at least one video. Report back uid, pid.
4. For each user, find the videos that they have watched multiple times and at least one of these times was on the day the video was released. These videos must also be at least 8 minutes long. Report back uid, vid, title.
5. Find the playlists such that all videos in the playlist were uploaded before the creation of the playlist. These playlists should also contain at least one video. Report back uid, pname, date_created.
6. For each user, report the users they are subscribed to that they've watched every video of. These users should also have at least one video uploaded. Report back uid, subscribed.

7. Find the movies that have been tagged with at least 3 different genres. One of which must be 'Action' and we do not want the ones that have been tagged with 'Horror'. Report back mid, title, description.
8. Find the free movies that are rated 'G', 'PG', or 'PG13' and were never reviewed 1 star by a user after they watched the movie. Report back mid, title, rating.
9. For all paid movies provided by Disney in 2016, find the users that purchased the movie within a year of the movie's release date and watched the movie within a year of when they purchased the movie ("within a year" means "on or before the same day next year"). Assume no movies were released on February 29, 2016. Report only one row per user-movie pair. Report back mid, uid, date_released, date_purchased, date_watched. The date_watched reported per user-movie pair should be the first time the user watched that movie.
10. Find the movies that were released before 2020, are at least 2 hours long, and are tagged with 'Fantasy' and at least one of 'Action' or 'Adventure'. These movies must also have been given at least one review for each possible star rating with a non-empty description at least once. Report back mid, title, date_released, duration.

## Part 2: Additional Integrity Constraints (7 points)

1. All videos in a playlist must be distinct.
2. If a movie is not free, it can only be purchased by a user after the movie's released date and can only be watched after they have purchased it. Movies that are free can only be watched by a user after the movie's release date. Movies can only be reviewed by a user after they have watched the movie.

When writing your queries, do not assume that these additional integrity constraints hold.

## Style and formatting requirements (3 points)

To make your algebra more readable, and to minimize errors, we are including these style and formatting requirements:

- In your assignment statements, you must include names for all attributes in the intermediate relation you are defining.

  For example, write

  $$Highest\ Grade(sID, oID, grade) := \ldots$$

- Use meaningful names for intermediate relations and attributes, just as you would in a program.
- If you want to include comments, put them before the algebra that they pertain to, not after. Make them stand out from the algebra, for example by using a different font.

  For example, this looks reasonable:

  /* Students who had very high grades in any offering of a csc */

$$course.High(sID) := \pi_{sID}\sigma_{dept='csc' \wedge grade>95}(Took \bowtie Offering)$$

A modest portion of your mark will be for good style and formatting.

## Submission Instructions:

For this assignment, you may work in **group of maximum 2**.
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.
Whatever you choose to use, you need to produce a final document in pdf format. You must declare your team (whether it is a team of two or three students) and hand in your work electronically using the MarkUs online system.
Well before the due date, you should declare your team and try submitting with MarkUs.
For this assignment, hand in just one file: **A1.pdf**.
If you are working in a group, only one of you should hand it in. Check that you have submitted the correct version of your file by downloading it from MarkUs.
**Late submissions with 10% penalty will be accepted until day 3 (72 hours) past the original submission deadline.** Later submissions will not be graded unless valid reason for the late submission is provided.