Your Name: Phichayapha Phuengmak (Monika)

1.
```
--Table Creation--

-- Entity tables below: --

CREATE TABLE IF NOT EXISTS Users (
user_id INTEGER NOT NULL PRIMARY KEY,
email TEXT NOT NULL,
first_name TEXT NOT NULL,
last_name TEXT NOT NULL );

CREATE TABLE IF NOT EXISTS Students (
user_id INTEGER NOT NULL PRIMARY KEY,
occupation TEXT,
FOREIGN KEY (user_id) REFERENCES Users );

CREATE TABLE IF NOT EXISTS Instructors (
user_id INTEGER NOT NULL PRIMARY KEY,
title TEXT,
FOREIGN KEY (user_id) REFERENCES Users );

CREATE TABLE IF NOT EXISTS InstructorEducation (
instructor_id INTEGER NOT NULL,
education_id INTEGER NOT NULL,
degree TEXT NOT NULL,
major TEXT NOT NULL,
school TEXT NOT NULL,
graduation_year INTEGER NOT NULL,
PRIMARY KEY (instructor_id, education_id),
FOREIGN KEY (instructor_id) REFERENCES Instructors );

CREATE TABLE IF NOT EXISTS Courses (
course_id INTEGER NOT NULL PRIMARY KEY,
course_name TEXT NOT NULL,
description TEXT );
```

```
CREATE TABLE IF NOT EXISTS Recurrences (
recurr_id INTEGER NOT NULL PRIMARY KEY,
repeat_on TEXT NOT NULL,
end_date DATE NOT NULL );

CREATE TABLE IF NOT EXISTS Meetings (
meeting_id INTEGER NOT NULL PRIMARY KEY,
meeting_name TEXT NOT NULL,
passcode TEXT,
start_at TIMESTAMP NOT NULL,
duration INTEGER NOT NULL,
mute_participants BOOLEAN NOT NULL,
course_id INTEGER NOT NULL,
instructor_id INTEGER NOT NULL,
recurr_id INTEGER NOT NULL,
FOREIGN KEY (course_id) REFERENCES Courses,
FOREIGN KEY (instructor_id) REFERENCES Instructors,
FOREIGN KEY (recurr_id) REFERENCES Recurrences );

CREATE TABLE IF NOT EXISTS Recordings (
recording_id INTEGER NOT NULL PRIMARY KEY,
start_time TIMESTAMP NOT NULL,
end_time TIMESTAMP NOT NULL,
meeting_id INTEGER NOT NULL,
FOREIGN KEY (meeting_id) REFERENCES Meetings );

CREATE TABLE IF NOT EXISTS Posts (
post_id INTEGER NOT NULL PRIMARY KEY,
post_type TEXT NOT NULL,
body TEXT,
created_at TIMESTAMP NOT NULL,
user_id INTEGER NOT NULL,
meeting_id INTEGER NOT NULL,
replied_to_post_id INTEGER,
topics TEXT,
FOREIGN KEY (user_id) REFERENCES Users,
FOREIGN KEY (meeting_id) REFERENCES Meetings );
```

```
-- Relationship tables below: –

CREATE TABLE IF NOT EXISTS EnrolledIn (
user_id INTEGER NOT NULL,
course_id INTEGER NOT NULL,
enroll_date DATE NOT NULL,
PRIMARY KEY (user_id, course_id),
FOREIGN KEY (user_id) REFERENCES Students,
FOREIGN KEY (course_id) REFERENCES Courses );

CREATE TABLE IF NOT EXISTS Teaches (
user_id INTEGER NOT NULL,
course_id INTEGER NOT NULL,
FOREIGN KEY (user_id) REFERENCES Instructors,
FOREIGN KEY (course_id) REFERENCES Courses );

CREATE TABLE IF NOT EXISTS Attended (
user_id INTEGER NOT NULL,
meeting_id INTEGER NOT NULL,
PRIMARY KEY (user_id, meeting_id),
FOREIGN KEY (user_id) REFERENCES Students,
FOREIGN KEY (meeting_id) REFERENCES Meetings );

CREATE TABLE IF NOT EXISTS ThumbsUp (
user_id INTEGER NOT NULL,
post_id INTEGER NOT NULL,
PRIMARY KEY (user_id, post_id),
FOREIGN KEY (user_id) REFERENCES Users,
FOREIGN KEY (post_id) REFERENCES Posts );

CREATE TABLE IF NOT EXISTS Watched (
recording_id INTEGER NOT NULL,
user_id INTEGER NOT NULL,
PRIMARY KEY (recording_id, user_id),
FOREIGN KEY (recording_id) REFERENCES Recordings,
FOREIGN KEY (user_id) REFERENCES Users );

CREATE TABLE IF NOT EXISTS WatchedSegment (
recording_id INTEGER NOT NULL,
user_id INTEGER NOT NULL,
segment_id INTEGER NOT NULL,
watched_from TIMESTAMP NOT NULL,
watched_to TIMESTAMP NOT NULL,
PRIMARY KEY (recording_id, user_id, segment_id),
FOREIGN KEY (recording_id) REFERENCES Recordings,
FOREIGN KEY (user_id) REFERENCES Users );
```

2. `-- Data Loading –`

```
COPY Users
FROM 'E:/CS122D/HW1-Cs122D/users.csv'
DELIMITER ',' CSV HEADER;

COPY Students
FROM 'E:/CS122D/HW1-Cs122D/students.csv'
DELIMITER ',' CSV HEADER;

COPY Instructors
FROM 'E:/CS122D/HW1-Cs122D/instructors.csv'
DELIMITER ',' CSV HEADER;

COPY InstructorEducation
FROM 'E:/CS122D/HW1-Cs122D/instructor_education.csv'
DELIMITER ',' CSV HEADER;

COPY Courses
FROM 'E:/CS122D/HW1-Cs122D/courses.csv'
DELIMITER ',' CSV HEADER;

COPY EnrolledIn
FROM 'E:/CS122D/HW1-Cs122D/enrolled_in.csv'
DELIMITER ',' CSV HEADER;

COPY Teaches
FROM 'E:/CS122D/HW1-Cs122D/teaches.csv'
DELIMITER ',' CSV HEADER;
```

```
COPY Recurrences
FROM 'E:/CS122D/HW1-Cs122D/recurrences.csv'
DELIMITER ',' CSV HEADER;

COPY Meetings
FROM 'E:/CS122D/HW1-Cs122D/meetings.csv'
DELIMITER ',' CSV HEADER;

COPY Attended
FROM 'E:/CS122D/HW1-Cs122D/attended.csv'
DELIMITER ',' CSV HEADER;

COPY Recordings
FROM 'E:/CS122D/HW1-Cs122D/recordings.csv'
DELIMITER ',' CSV HEADER;

COPY Posts
FROM 'E:/CS122D/HW1-Cs122D/posts.csv'
DELIMITER ',' CSV HEADER;

COPY ThumbsUp
FROM 'E:/CS122D/HW1-Cs122D/thumbs_up.csv'
DELIMITER ',' CSV HEADER;

COPY Watched
FROM 'E:/CS122D/HW1-Cs122D/watched.csv'
DELIMITER ',' CSV HEADER;

COPY WatchedSegment
FROM 'E:/CS122D/HW1-Cs122D/watched_segment.csv'
DELIMITER ',' CSV HEADER;
```

3. --Query Answers--

```
-- Problem A –

SELECT COUNT(DISTINCT s.user_id), COUNT(DISTINCT i.user_id),
COUNT(DISTINCT c.course_id)
FROM Students AS s, Instructors AS i, Courses AS c;


-- Problem B –

SELECT u.user_id, u.first_name, u.last_name
FROM Instructors AS i, EnrolledIn AS e, Users AS u
WHERE i.user_id = e.user_id AND i.user_id = u.user_id
GROUP BY u.user_id, u,first_name, u.last_name
HAVING COUNT(e.course_id) > 5;


-- Problem C –

SELECT c.course_id, u.first_name, u.last_name, COUNT(e.user_id)
FROM Courses AS c, EnrolledIn AS e, Teaches AS t, Users AS u
WHERE c.course_id = e.course_id AND c.course_id = t.course_id
AND t.user_id = u.user_id
GROUP BY c.course_id, u.first_name, u.last_name
ORDER BY COUNT(DISTINCT e.user_id) DESC
LIMIT 10;
```

```
-- Problem D –

WITH first(f_count) AS
      ( SELECT COUNT(DISTINCT e.user_id)
      FROM EnrolledIn AS e
      WHERE e.enroll_date > '2020-03-01' AND e.enroll_date <
'2020-04-01' ),
second(s_count) AS
      ( SELECT COUNT(DISTINCT e.user_id)
      FROM EnrolledIn AS e
      WHERE e.enroll_date > '2020-04-01' AND e.enroll_date <
'2020-05-01' )
SELECT f.f_count, s.s_count, (f.f_count - s.s_count) AS
difference
FROM first AS f, second AS s;


-- Problem E –

SELECT AVG(r.end_time - r.start_time)
FROM Instructors AS i, Recordings AS r, Meetings AS m, Users AS
u
WHERE u.first_name = 'Sabrina' AND u.last_name = 'Lawrence' AND
u.user_id = i.user_id AND
i.user_id = m.instructor_id AND r.meeting_id = m.meeting_id AND
m.start_at >= '2020-01-02' AND
m.start_at <= '2020-02-01';


-- Problem F –

SELECT p.post_id, p.post_type, string_to_array(p.topics, ', ')
AS topics, array_length(string_to_array(p.topics, ', '), 1) AS
num_topics
FROM Posts AS p
WHERE p.created_at > '2020-06-21 23:00:00' AND p.created_at <
'2020-06-21 24:00:00';
```

```
-- Problem G --

SELECT p.post_id, p.post_type, each_topic
FROM Posts AS p, UNNEST(string_to_array(p.topics, ', ')) AS
each_topic
WHERE p.created_at > '2020-06-21 23:00:00' AND p.created_at <
'2020-06-21 24:00:00'
ORDER BY p.post_id;


-- Problem H --

WITH newposts(post_id, each_topic) AS
( SELECT p.post_id, each_topic
FROM Posts AS p, UNNEST(string_to_array(p.topics, ', ')) AS
each_topic )
SELECT np.each_topic, COUNT(DISTINCT t.user_id) AS thumbCounts
FROM ThumbsUp AS t, newposts AS np
WHERE np.post_id = t.post_id
GROUP BY np.each_topic
ORDER BY thumbCounts DESC
LIMIT 5;


-- Problem I --

/*
This query takes 116 msec the first time it runs, and reduces
down to below 50 msec after the second run.
*/
WITH well_watched(recording_id, duration) AS
      ( SELECT ws.recording_id, SUM(ws.watched_to -
ws.watched_from)
      FROM Recordings AS r, Watched AS w, WatchedSegment AS ws
      WHERE r.recording_id = w.recording_id AND w.recording_id =
ws.recording_id AND w.user_id = ws.user_id
      GROUP BY ws.recording_id, ws.user_id
      HAVING SUM(ws.watched_to - ws.watched_from) > '00:30:00'
      ORDER BY ws.recording_id )
SELECT COUNT(DISTINCT ww.recording_id)
FROM well_watched AS ww;


-- Problem J –

ALTER TABLE WatchedSegment
ADD duration INTERVAL;

UPDATE WatchedSegment
SET duration = watched_to - watched_from;


-- Problem K --

/*
After adding the duration field into WatchedSegment,
the running time decreases to 77 msec.
*/

WITH well_watched(recording_id, duration) AS
      ( SELECT ws.recording_id, SUM(ws.watched_to -
ws.watched_from)
      FROM Recordings AS r, Watched AS w, WatchedSegment AS ws
      WHERE r.recording_id = w.recording_id AND w.recording_id =
ws.recording_id AND w.user_id = ws.user_id
      GROUP BY ws.recording_id, ws.user_id
      HAVING SUM(ws.duration) > '00:30:00'
      ORDER BY ws.recording_id )
SELECT COUNT(DISTINCT ww.recording_id)
FROM well_watched AS ww;

/*
After an index for duration is created, the running time
is 64 msec. I think the main reason for the difference is
because indexes help find the duration faster. Since the table
is not sorted by duration, it needs to go through the tuples
```

```
one by one to see if the duration exceeds 30 minutes. With the
index, the search can skip blocks of duration and start
searching at the largest duration smaller or equal to 30
minutes.
*/

CREATE INDEX IF NOT EXISTS duration_index
ON WatchedSegment(duration);

WITH well_watched(recording_id, duration) AS
      ( SELECT ws.recording_id, SUM(ws.watched_to -
ws.watched_from)
      FROM Recordings AS r, Watched AS w, WatchedSegment AS ws
      WHERE r.recording_id = w.recording_id AND w.recording_id =
ws.recording_id AND w.user_id = ws.user_id
      GROUP BY ws.recording_id, ws.user_id
      HAVING SUM(ws.duration) > '00:30:00'
      ORDER BY ws.recording_id )
SELECT COUNT(DISTINCT ww.recording_id)
FROM well_watched AS ww;


-- Problem L –

/*
CUBE() displays the number of instructors of all possible
subsets of (ie.school, ie.degree), while ROLLUP() displays the
instructor counts for the prefix attributes.
*/

SELECT ie.school, ie.degree, COUNT(DISTINCT ie.instructor_id)
FROM InstructorEducation AS ie
GROUP BY CUBE(ie.school, ie.degree)
ORDER BY COUNT(DISTINCT ie.instructor_id) DESC
LIMIT 10;

SELECT ie.school, ie.degree, COUNT(DISTINCT ie.instructor_id)
FROM InstructorEducation AS ie
GROUP BY ROLLUP(ie.school, ie.degree)
```

```
ORDER BY COUNT(DISTINCT ie.instructor_id) DESC
LIMIT 10;


-- Problem M (extra credit) --

WITH with_rank(course_id, enroll_count, count_rank) AS
( SELECT e.course_id, COUNT(DISTINCT e.user_id), dense_rank()
OVER (ORDER BY (COUNT(DISTINCT e.user_id)) DESC)
FROM EnrolledIn AS e
GROUP BY e.course_id )
SELECT *
FROM with_rank AS wr
WHERE wr.count_rank <= 10
GROUP BY wr.course_id, wr.enroll_count, wr.count_rank
ORDER BY wr.count_rank;
```