Name

UIN

Homework Assignment 2

Due Date: March 12, 2025

CS480 - Database Systems

Results

Please leave this empty!

2.1 2.2

2.3 2.4 2.5 2.6 2.7 2.8 2.9 2.10

2.15 2.16 2.17 2.18 2.19 Sum

Instructions

- Try to answer all the questions using what you have learned in class
- **When writing a query, write the query in a way that it would work over all possible database instances and not just for the given example instance!**
- Some questions are marked as bonus. You do not have to answer these questions to get full points for the assignment. However, you can get bonus points for these questions!
- **Please submit the homework electronically using gradescope**
- **Submissions will be automatically graded. You can submit multiple times.**
- **The due date is 03/11!**

Consider the following database schema and example instance storing information about theatre plays:

student

UIN	name	major	year
111111	Peter	CS	1
112333	Bob	CS	2
443223	Alice	EE	3
234242	Veeta	CS	2
366666	Fauly	CS	6
554344	Rufus	BIO	2
777777	Gertrud	CS	1
888333	Gertrud	CS	2

hackathon

title	start_date	end_date	description
Sparkhacks 2024	2024-02-08	2024-02-11	UIC-based event
Sparkhacks 2023	2023-02-01	2023-02-02	UIC-based event
Chasehacks 2022	2022-01-01	2022-01-02	Bank-organized

team

teamname	hackathon
The bosses	Sparkhacks 2024
The bosses	Sparkhacks 2023
The bosses	Chasehacks 2022
The noobs	Sparkhacks 2024
Hack On	Sparkhacks 2024
Hack On	Sparkhacks 2023

student_team

teamname	hackathon	student
The bosses	Sparkhacks 2024	366666
The bosses	Sparkhacks 2024	443223
The bosses	Sparkhacks 2023	366666
The bosses	Sparkhacks 2023	443223
The bosses	Chasehacks 2022	366666
The bosses	Chasehacks 2022	443223
The bosses	Sparkhacks 2024	111111
The noobs	Sparkhacks 2024	777777
The noobs	Sparkhacks 2024	554344
The noobs	Sparkhacks 2024	443223
Hack On	Sparkhacks 2024	234242
Hack On	Sparkhacks 2024	112333
Hack On	Sparkhacks 2023	234242
Hack On	Sparkhacks 2023	112333

organizer

student	hackathon	role
366666	Sparkhacks 2024	director
234242	Sparkhacks 2024	outreach
366666	Sparkhacks 2023	director
111111	Chasehacks 2022	director

project

teamname	hackathon	title
The bosses	Sparkhacks 2024	Boring React Mockup
The bosses	Sparkhacks 2023	Convoluted Haskell Nonsense
The bosses	Chasehacks 2022	Verbose Java Mess
The noobs	Sparkhacks 2024	Too many fancy frameworks
Hack On	Sparkhacks 2024	Static HTML to the Max
Hack On	Sparkhacks 2023	Unnecessary Assembly

sponsorship

hackathon	sponsor	amount
Sparkhacks 2024	UIC	3500
Sparkhacks 2024	Chase	500
Sparkhacks 2024	Oracle	10000
Sparkhacks 2023	UIC	4500
Chasehacks 2022	Chase	20000
Chasehacks 2022	UIC	250

judge

name	affiliation	position
Lazy young prof	UIC	Assistant Professor
Lazy old prof	UIC	Full Professor
No morals industry guy	UIC	Associate Professor

rating

judge	team	hackathon	project	rating
Lazy young prof	The bosses	Sparkhacks 2024	Boring React Mockup	2.0
Lazy old prof	The bosses	Sparkhacks 2024	Boring React Mockup	5.0
No morals industry guy	The bosses	Sparkhacks 2024	Boring React Mockup	3.5
Lazy young prof	The bosses	Sparkhacks 2023	Convoluted Haskell Nonsense	5.0
Lazy old prof	The bosses	Sparkhacks 2023	Convoluted Haskell Nonsense	1.0
No morals industry guy	The bosses	Chasehacks 2022	Verbose Java Mess	5.0
Lazy young prof	The noobs	Sparkhacks 2024	Too many fancy frameworks	2.5
Lazy old prof	The noobs	Sparkhacks 2024	Too many fancy frameworks	4.0
No morals industry guy	The noobs	Sparkhacks 2024	Too many fancy frameworks	4.5
Lazy young prof	Hack On	Sparkhacks 2024	Static HTML to the Max	1.0
Lazy old prof	Hack On	Sparkhacks 2024	Static HTML to the Max	3.0
No morals industry guy	Hack On	Sparkhacks 2024	Static HTML to the Max	1.5
Lazy young prof	Hack On	Sparkhacks 2023	Unnecessary Assembly	1.0
Lazy old prof	Hack On	Sparkhacks 2023	Unnecessary Assembly	5.0

Hints:

- If you are unsure about the type of an attribute look at the SQL script for the homework. The same applied to foreign key constraints which are defined the DDL script for the homework.
- Attributes with black background form the primary key of an relation.

Part 2.1 SQL DDL (Total: 14 Points)

Question 2.1.1 (5 Points)

Write an SQL statement that adds a *salary* attribute to relation *organizer* which stores the salary (a number with at most two digits after the dot and at most 8 digits in total) the organizer receives. Ensure that salaries are positive (or zero). The default salary should be zero. **Solution**

```
ALTER TABLE organizer
  ADD COLUMN salary NUMERIC(8,2) CHECK (salary >= 0) DEFAULT 0;
```

Question 2.1.2 (9 Points)

Write an SQL statement that creates a new table `conflicts` recording conflicts of interests between students and judges, e.g., a student that did TA for a professor that is a judge has a conflict of interest with the professor. For each conflict record the UIN of the student (`UIN`), the name of the judge (`judge`), and the type of conflict (`coitype`): a string that should be one of the following values *"TA"*, *"RA"*, *"internship"*, or *"family member"*. A student may have multiple conflicts of interest with a professor, e.g., if the student did both TA and RA for this professor.

After creating the table insert the following data (you need it to test queries). You can find the SQL code commented out in the SQL script for the homework.

For the autograder submission, we will insert data, just submit the `CREATE TABLE` statement.

```
INSERT INTO conflicts
VALUES
('111111', 'Lazy young prof', 'TA'),
('111111', 'Lazy young prof', 'RA'),
('443223', 'No morals industry guy', 'internship');
```

Solution

```
CREATE TABLE conflicts (
  UIN TEXT NOT NULL REFERENCES student,
  judge TEXT NOT NULL REFERENCES judge,
  coitype TEXT CHECK (coitype IN ('TA', 'RA', 'internship', 'family member')),
  PRIMARY KEY (UIN, judge, coitype)
  );
```

Part 2.2 SQL Queries (Total: 56 + 10 BONUS Points)

Question 2.2.1 (5 Points)

Write an SQL query that returns the name and major of students that are not studying CS and have been members of a team that participated in Sparkhacks 2023 and Sparkhacks 2024. Make sure that every student fulfilling this condition is only returned once. The result schema should be `(name,major)`.

Solution

```
SELECT DISTINCT s.name, s.major
  FROM student s, student_team sh23, student_team sh24
 WHERE s.UIN = sh23.student
   AND sh23.hackathon = 'Sparkhacks 2023'
   AND s.UIN = sh24.student
   AND sh24.hackathon = 'Sparkhacks 2024'
   AND s.major != 'CS';
```

Question 2.2.2 (5 Points)

Write an SQL query that returns the average rating per project. Return only projects whose average rating is larger than `3.0`. The result schema should be `(project, avgrating)`.

Solution

```sql
SELECT project, avg(rating) AS avgrating
FROM rating
GROUP BY project
HAVING avg(rating) > 3.0;
```

Question 2.2.3 (7 Points)

Write an SQL query that returns for each student the number of hackathons they have participated in. Do not double count if a student participated in multiple teams in the same hackathon. This query should return every student, including students that have not participated in any hackathon so far. The result schema should be `(UIN,name,numhack)`.

Solution

```
SELECT s.UIN,
       s.name,
       COALESCE(count(DISTINCT t.hackathon),0) AS numhack
FROM  student s LEFT OUTER JOIN student_team t ON s.UIN = t.student
GROUP BY s.UIN, s.name;
```

Question 2.2.4 (7 Points)

Write an SQL query that returns computes for each hackathon the total amount of sponsorship the hackathon has received and then returns the number of hackathons that have received more $10,000 of total sponsorship. The result schema should be `(numhack)`.

Solution

```
SELECT count(*) AS numhack
  FROM (SELECT hackathon, sum(amount) AS ttlamt
          FROM sponsorship
         GROUP BY hackathon
        HAVING sum(amount) > 10000) s
```

Question 2.2.5 (8 Points)

Write an SQL query that returns a rolling sum over time of the for each sponsor of the total amount of sponsorships all hackathons combined have received from this sponsor until and including this point of time. Use the `start_date` of hackathons as the date for the sponsorships of a hackthon. Order the results by start date and sponsor. The result schema should be `(sponsor, start_date, ttlamount)`.

Solution

```
SELECT sponsor,
       start_date,
       sum(amount) OVER (
         PARTITION BY sponsor
         ORDER BY start_date
       ) AS ttlamount
  FROM sponsorship s, hackathon h
WHERE h.title = s.hackathon
ORDER BY sponsor, start_date;
```

Question 2.2.6 (8 Points)

Write an SQL query that returns for each hackathon the amount of price money each team will receive. For every hackathon, 10% of the total sponsorship should be given as a price to the team with the highest average rating from all the ratings they received for their project at this hackathon. If there are multiple teams that have the highest rating for a hackthon, the price money should be divided evenly among these teams. The query should return every team that participated in each hackathon even if the team did not receive a price at this hackathon. The result schema should be `(hackathon, team, pricemoney)`.

Solution

```sql
WITH pricemoney AS
     (SELECT hackathon,
             sum(amount) * 0.1 AS pricemoney
        FROM sponsorship
       GROUP BY hackathon),
  avgratings AS
    (SELECT hackathon, team, avg(rating) AS avgrating
       FROM rating
      GROUP BY hackathon, team),
  bestteams AS
    (SELECT hackathon, team
       FROM avgratings t
      WHERE avgrating IN (SELECT max(avgrating)
                          FROM avgratings m
                         WHERE m.hackathon = t.hackathon)),
  cntbest AS
    (SELECT hackathon, count(*) AS cntteam
       FROM bestteams b
      GROUP BY hackathon)
SELECT a.hackathon,
       a.team,
       CASE
       WHEN a.team IN (SELECT team
                       FROM bestteams b
                      WHERE b.hackathon = a.hackathon)
         THEN
           p.pricemoney / (SELECT cntteam
                           FROM cntbest c
                          WHERE c.hackathon = a.hackathon)
       ELSE
           0.0
       END AS pricemoney
  FROM pricemoney p, avgratings a
 WHERE p.hackathon = a.hackathon;
```

Question 2.2.7 (8 Points)

Write an SQL query that returns the name and major of the youngest student (determined based on the `year` the student is currently in) to be in the best team at some hackathon. Best should be determined by the maximum rating the team received at the hackathon. The result schema should be `(name)`.

Solution

```
WITH
  maxratings AS
    (SELECT hackathon, team, max(rating) AS maxrating
       FROM rating
      GROUP BY hackathon, team),
  hackmax AS
    (SELECT hackathon, max(maxrating) AS bestrating
       FROM maxratings
      GROUP BY hackathon),
  bestteams AS
    (SELECT hackathon, team
       FROM maxratings r
      WHERE maxrating IN (SELECT bestrating
                            FROM hackmax m
                           WHERE r.hackathon = m.hackathon)),
  bestmembers AS
    (SELECT s.UIN, s.year
       FROM student s, student_team t
      WHERE s.UIN = t.student
      AND t.teamname IN (SELECT team FROM bestteams))

SELECT s.name, s.major
  FROM student s
 WHERE s.UIN IN (SELECT UIN
                   FROM bestmembers
                  WHERE year = (SELECT min(year)
                  FROM bestmembers))
```

Question 2.2.8 (8 Points)

Write an SQL query that returns the name and position of judges that have judged every hackathon. The result schema should be `(name, position)`.

Solution

```
SELECT name, position
FROM judge j
WHERE NOT EXISTS (SELECT *
                  FROM hackathon h
                  WHERE NOT EXISTS (SELECT *
                                    FROM rating r
                                    WHERE r.judge = j.name
                                          AND r.hackathon = h.title));
```

Part 2.3 SQL Updates (Total: 30 Points)

Question 2.3.1 (7 Points)

Delete all organizers which have the role *"director"* or where the hackathon is *"Sparkhacks 2023"*.

Solution

```
DELETE FROM organizer
 WHERE hackathon = 'Sparkhacks 2023' OR role = 'director';
```

Question 2.3.2 (8 Points)

Add a new hackathon named *"Sparkhacks 2025"* taking place from `2025-02-01` to `2025-02-04` with description *"UIC-based event"*. Then add organizers of Sparkhacks 2024 as organizers (with the same role they had in 2024) for Sparkhacks 2025. Do not add organizers that are in their 5 year or later.

Solution

```
INSERT INTO hackathon
VALUES ('Sparkhacks 2025', '2025-02-01'::date, '2025-02-04'::date, 'UIC-based event');

INSERT INTO organizer (SELECT student, 'Sparkhacks 2025', role
                         FROM organizer
                        WHERE hackathon = 'Sparkhacks 2024'
                          AND student NOT IN (SELECT UIN FROM student WHERE year >= 5)
);
```

Question 2.3.3 (6 Points)

Increase the year for all CS students by 1;

Solution

```
UPDATE student
SET year = year + 1
WHERE major = 'CS';
```

Question 2.3.4 (9 Points)

Delete all students that have not participated in any hackathon.

Solution

```
DELETE FROM student s
WHERE NOT EXISTS (SELECT * FROM student_team t WHERE s.UIN = t.student);
```

Part 2.4 SQL Bonus (Total: 15 Points)

Question 2.4.1 BONUS (15 Points)

Write an SQL query which computes a k-means clustering of a set of points using euclidean distance. If you are unaware of the k-means clustering algorithm you can find an explanation here: https://en.wikipedia.org/wiki/K-means_clustering. The query should return the centroids of the k clusters. The input of the query is a relation `points` with schema `(x,y,z)`. The result schema should be `(cid,x,y,z)` containing the k centroids (`cid` from 1 to k). You should start from an initial selection of centroids which contains the k first points sorted by x, y, z. We will use $k = 4$.

- No procedural SQL is allowed.
- **This question is quite challenging. Probably solve all other questions first before attempting to solve this one!**

Solution

```
WITH RECURSIVE kmeans AS (
  -- determine initial centroids (first k=4 points in x,y,z oder)
  (SELECT 0 AS iter, cid, x::float,y::float,z::float
   FROM generate_series(1,4) clusters(cid)
        NATURAL JOIN
       (SELECT x,y,z, ROW_NUMBER() OVER (ORDER BY x,y,z) AS cid
        FROM points))
   UNION
   -- filter if we have converged
   (SELECT iter, cid, x,y,z
    FROM
      (SELECT *, sum(abs(prevx - x) + abs(prevy - y) + abs(prevz - z)) OVER () AS diff
       FROM
         -- return updated centroids
         (SELECT DISTINCT iter, cid, xnc AS x, ync AS y, znc AS z, prevx, prevy, prevz
          FROM
            -- update centroids (avg of points)
            (SELECT cid, iter,
                    avg(x) OVER (PARTITION BY cid) AS xnc,
                    avg(y) OVER (PARTITION BY cid) AS ync,
                    avg(z) OVER (PARTITION BY cid) AS znc,
                    prevx, prevy, prevz
             FROM
               -- assign points to closest centroid
               (SELECT *
                FROM
                  (SELECT *, min(cid) OVER (PARTITION BY x,y,z) AS mcid
                   FROM
                     -- compute closest centroids
                     (SELECT cid, iter, cx, cy, cz, x, y, z, dist,
                             min(dist) OVER (PARTITION BY x, y, z) AS mind,
                             prevx, prevy, prevz
                      FROM
                      -- compute distances
                        (SELECT cid, iter, newiter.x AS cx, newiter.y AS cy, newiter.z AS
                                p.x, p.y, p.z,
                                sqrt((newiter.x - p.x)^2 + (newiter.y - p.y)^2
                                     + (newiter.z - p.z)^2) AS dist, prevx, prevy, prevz
                         FROM
                           points p,
                           -- generate new iteration of centroids
                           (SELECT iter + 1 AS iter, cid, x, y, z,
                                   iter AS previter,
                                   x AS prevx, y AS prevy, z AS prevz
                            FROM kmeans) newiter -- generate new iteration centroid tuples
                         ) dist
                      ) alldist -- all distances between points and current centroids
                   WHERE dist = mind) -- keep min dist centroids for each point
                WHERE cid = mcid) -- break ties
             ) newcentroids
          ) updatedcentroids
    ) convergediff -- compute diff
    WHERE diff != 0) -- new iter
) -- whole kmeans computation

SELECT cid,x,y,z
 FROM kmeans
WHERE iter = (SELECT max(iter) FROM kmeans);
```