CS2102-2310: Database Systems
Midterm

Date: 07 October 2025, Time: 12:30–13:30

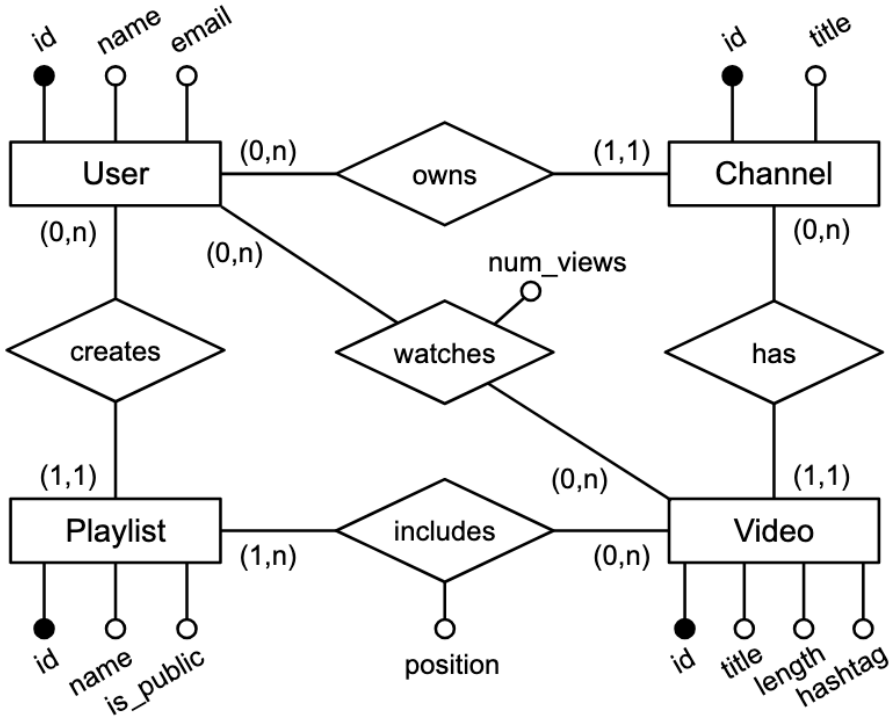
Submission Instructions

1. Please read **ALL** instructions carefully.
2. All the assessment is be done using Examplify; the assessment contains:
   (a) MCQ/MRQ: Questions 1–5
   (b) SQL Queries: Questions 6–10
3. The total number of points is 25
4. This is an open-book assessment.
5. Your Internet connection will be blocked for the duration of the assessment.
6. This assessment starts at 12:30 and ends at 13:30.
   - Submit your answers by 13:30
   - No additional time will be given to submit.
7. For the MCQ/MRQ questions: Please note that the order of the answers might differ from the order in the PDF due to randomization!
8. For the SQL questions, there are additional instructions below; in a nutshell:
   - Use an IDE or text editor to write your queries and test them using `psql` or pgAdmin.
   - Prepare your final answer before submitting it to Examplify according to the instructions in the SQL Part.
   - Allocate sufficient time to prepare your final answers and to copy-&-paste them into Examplify.
9. Failure to follow each of the instructions above may result in deduction of your marks.

Good Luck!

Database Modeling (*12 Marks*)

The ER Diagram below models a very simple video sharing platform like YouTube.



Let's assume the application requirements include the following list of 8 constraints (there might be more, but focus on these 8):

(A) If a Channel gets deleted, all videos of that channel gets automatically deleted as well.

(B) A Channel is owned by exactly one User.

(C) All Videos of the same Channel must have different titles.

(D) A Video uniquely identifies the user who has uploaded that video.

(E) A Playlist cannot be empty but must contain at least 1 video.

(F) For a given Playlist, the value of `position` must be unique and between 1 and N (with N being the number of videos in that Playlist)

(G) `num_views` must be greater or equal to 1.

(H) A video has 1 or more hashtags.

Q1: (3 points) **ER diagram constraints.** Which of the 8 constraints above are **NOT CAPTURED** by the ER diagram? Select all that apply.

√ **A**
◯ B
√ **C**
◯ D
◯ E
√ **F**
√ **G**
√ **H**

Solution:

How B, D, and E are captured

- (B) A Channel is owned by exactly one User.
  - Total participation constraint + cardinality constraint *(i.e., key constraint)* on `Channel` w.r.t `owns`.
- (D) A Video uniquely identifies the user who has uploaded that Video.
  - Total participation constraint + cardinality constraint *(i.e., key constraint)* on `Channel` w.r.t `has` **AND** total participation constraint + cardinality constraint *(i.e., key constraint)* on `Channel` w.r.t `owns`.
- (E) A Playlist cannot be empty but must contain at least 1 video.
  - Total participation constraint on `Playlist` w.r.t. `includes`.

Q2: (2 points) **Database schema.** Your task is now to translate the ER diagram into a database schema. As a convention, if tables are merged, use the name of the entity set as the name for the table.

Assume that the database schema already contains the following tables

- `Playlist(id, name, is_public, user_id)`
- `Watches(user_id, video_id, num_views)`
- `Includes(playlist_id, video_id, position)`

Which other table need to be created to capture as many constraints of the ER diagram as possible? Select all that apply. (Note: there might be more tables not listed here.)

◯ `owns(channel_id, user_id)`

√ `User(id, name, email)`

√ `Channel(id, title, user_id)`

◯ `User(id, name, email, channel_id)`

√ `Video(id, title, length, hashtag, channel_id)`

Q3: (2 points) Which of the following foreign key (FK) constraints will be part of your database schema from Q2 incl. the three already given tables? Select all that apply. (Note: there might be more FK constraints not listed here; again, consider that merged tables get the name from the entity set.)

√ `Channel.user_id` → `User.id`

◯ `Has.channel_id` → `Channel.id`

√ `Watches.user_id` → `User.id`

◯ `Video.user_id` → `User.id`

√ `Playlist.user_id` → `User.id`

Q4: (3 points) **Database schema constraints.** Which of the 8 constraints constraints from Q1 **CANNOT BE CAPTURED** when converting the ER Diagram into the database schema using only concepts covered in lectures 1–5? Select all that apply.

◯ A

◯ B

◯ C

◯ D

√ **E**

√ **F**

◯ G

√ **H**

Solution: How the Rest are Captured

- (A) If a Channel gets deleted, all videos of that channel gets automatically deleted as well.
  - The foreign key `Video.cid` → `Channel.cid` can be defined with `ON DELETE CASCADE`.

- (B) A Channel is owned by exactly one User.
  - We have the foreign key `Channel.uid` → `User.uid` and we can add the constraint `Channel.uid` to be `NOT NULL`.
- (C) All Videos of the same Channel must have different titles.
  - Add `UNIQUE(cid, title)` to table `Channel`.
- (D) A Video uniquely identifies the user who has uploaded that Video.
  - We have the 2 foreign keys `Video.cid` → `Channel.cid` as well as `Channel.uid` → `User.uid` for form a chain Video→Channel→User.
- (G) `num_views` must be greater or equal to 1.
  - Add `CHECK (num_views > 0)` constraint to table `Watches`.

Q5: (2 points) **Querying the database.** Consider the following SQL query that finds the total number of videos that are not included in a playlist.

```
SELECT COUNT(*)
FROM Video v
WHERE v.id NOT IN (SELECT i.video_id
                   FROM Includes i);
```

Which of the following queries A-E are equivalent to the query above?

Query **A**

```
SELECT COUNT(*)
FROM Video v
WHERE NOT EXISTS (SELECT 1
                  FROM Includes i
                  WHERE v.id = i.video_id);
```

Query **B**

```
SELECT COUNT(*)
FROM Video v LEFT JOIN Includes i
  ON (v.id = i.video_id)
WHERE i.playlist_id IS NULL;
```

Query **C**

```
SELECT COUNT(*)
FROM (SELECT v.id FROM Video v
      EXCEPT ALL
      SELECT i.video_id FROM Includes i);
```

Query **D**

```
SELECT COUNT(*)
FROM Video v
WHERE v.id <> ANY (SELECT i.video_id
                   FROM Includes i);
```

Query **E**

```
SELECT COUNT(*)
FROM Video v FULL OUTER JOIN Includes i
  ON (v.id = i.video_id)
WHERE i.playlist_id IS NULL;
```

Select all queries that are equivalent to the original query!

- √ **A**
- √ **B**
- √ **C**
- ◯ D
- √ **E**

SQL Queries (*13 Marks*)

In this part of the exam your task is to formulate queries in SQL. We use the same database that we introduced in Assignment 1. This means you can directly run and test your queries using `psql` or `pgAdmin`. If needed, we provide a condensed representation the database schema in the Appendix.

Instructions. To ensure successful submission and grading of your SQL queries adhere to the following instructions:

- For each question, you are to write a **single SELECT SQL statement** to answer the question.
- You are only allowed to use SQL constructs that have been covered in the lectures.
- Each answer must be a syntactically valid SQL query and executable on PostgreSQL. Test with `psql` or `pgAdmin`!
- Each question must be answered independently of any other questions.
- You must not enter any extraneous text for your answer (e.g., "My answer is: ...), or enter multiple answers (i.e., submit multiple SQL statements). Your answer should not contain any comments.
- Note: The exact names of the columns in the result do not matter!

Recommendations. We recommend the following steps to answer each question:

- Write the answer query using your IDE or text editor of choice, and test it on your machine with `psql` or `pgAdmin`.
- Once you are happy with your query, copy the `SELECT` statement for the question into the corresponding answer field in Examplify.

Q6: (2 points) For each country, print the number of drivers from that country. Sort the result by the number of drivers from most to least. Exclude countries that have no driver. Print the country name and corresponding number of drivers.

Solution:

```
SELECT country, COUNT(*) AS driver_count
FROM drivers
GROUP BY country
ORDER BY driver_count DESC;
```

Q7: (2 points) Print the different drivers from the region of 'Asia' (note: You may use the constant 'Asia'). Output only the forename, surname, and country of the drivers. Sort the result in ascending order of forename. If there are ties, sort the ties in descending order of surname.

Solution:

```
SELECT d.forename, d.surname, d.country
FROM drivers d
WHERE country IN (SELECT name
                  FROM countries c
                  WHERE c.region = 'Asia')
ORDER BY d.forename ASC, d.surname DESC;
```

or

```
SELECT d.forename, d.surname, d.country
FROM drivers d, countries c
WHERE d.country = c.name
  AND c.region = 'Asia'
ORDER BY d.forename ASC, d.surname DESC;
```

Q8: (3 points) For each region, print the number of races in the region. Output only the name of the region and the number of races. Sort the result in descending order of number of races.

Solution:

```
SELECT n.region, COUNT(*) AS race_count
FROM circuits c, races r, countries n
```

```
WHERE r.circuit = c.name
  AND c.country = n.name
GROUP BY n.region
ORDER BY race_count DESC;
```

Q9: (3 points) Find the different races for which all drivers that start the race successfully reach the finish. A driver starts the race if the driver appears in the `results` table. A driver finishes the race if the column `position` in the table `results` is not `null`. Output only the name of the race and the name of the circuit where the race takes place. Sort the result in ascending order of date.

Solution:

```
SELECT rc.name, rc.circuit
FROM races rc
WHERE NOT EXISTS (
    SELECT *
    FROM results r
    WHERE rc.date = r.race
      AND r.position IS NULL
)
ORDER BY rc.date ASC;
```

Q10: (3 points) Find the different circuits that have been used for the least number of races. Include circuits that have not been used (if any). Output only the circuit name and the number of times the circuit is used. Sort the result by the circuit name in ascending order.

Solution:

```
SELECT c.name, COUNT(r.circuit)
FROM circuits c LEFT JOIN races r ON (c.name = r.circuit)
GROUP BY c.name
HAVING COUNT(r.circuit) <= ALL (
        SELECT COUNT(r.circuit)
        FROM circuits c LEFT JOIN races r ON (c.name = r.circuit)
        GROUP BY c.name
)
ORDER BY c.name ASC;
```

Appendix

Database Schema for SQL Queries

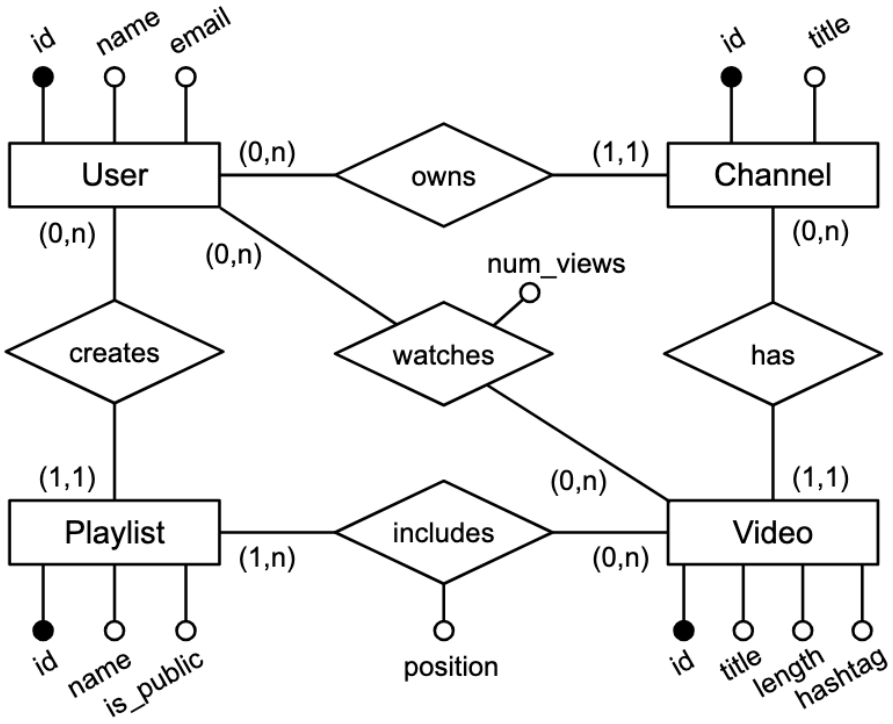
For this test, we use the same database as for Assignment 1. You should therefore be already familiar with it. For convenience, we include the schema of the database.

- countries(name, subregion, region)
- seasons(year, url)
- circuits(name, location, country, lat, lng, alt, url)
- drivers(forename, surname, code, number, dob, country, url)
- constructors(name, country, url)
- statuses(id, text)
- races(date, season, name, circuit, url)
- results(race, driver_forename, driver_surname, constructor, number, position, position_text, position_order, points, laps, time, fastest_lap, fastest_lap_time, fastest_lap_speed, status)
- constructor_results(race, constructor, points)
- pit_stops(race, driver_forename, driver_surname, lap, duration)
- qualifyings(race, driver_forename, driver_surname, number, constructor, position, q1, q2, q3)
- sprint_results(race, driver_forename, driver_surname, number, constructor, position, position_text, position_order, points, laps, time, fastest_lap, fastest_lap_time, status)

Additionally, the foreign key constraints are also specified.

- (circuits.country) ⇝ (countries.name)
- (drivers.country) ⇝ (countries.name)
- (constructors.country) ⇝ (countries.name)
- (races.season) ⇝ (seasons.year)
- (races.circuit) ⇝ (circuits.name)
- (results.race) ⇝ (races.date)
- (results.driver_forename, results.driver_surname) ⇝ (drivers.forename, drivers.surname)
- (results.constructor) ⇝ (constructors.name)
- (results.status) ⇝ (statuses.id)
- (constructor_results.race) ⇝ (races.date)
- (constructor_results.constructor) ⇝ (constructors.name)
- (pit_stops.race) ⇝ (races.date)
- (pit_stops.driver_forename, pit_stops.driver_surname) ⇝ (drivers.forename, drivers.surname)
- (qualifyings.race) ⇝ (races.date)
- (qualifyings.driver_forename, qualifyings.driver_surname) ⇝ (drivers.forename, drivers.surname)
- (qualifyings.constructor) ⇝ (constructors.name)
- (sprint_results.race) ⇝ (races.date)
- (sprint_results.driver_forename, sprint_results.driver_surname) ⇝ (drivers.forename, drivers.surname)
- (sprint_results.constructor) ⇝ (constructors.name)
- (sprint_results.status) ⇝ (statuses.id)

The ER Diagram of video sharing platform:



Constraints:

(A) If a Channel gets deleted, all videos of that channel gets automatically deleted as well.

(B) A Channel is owned by exactly one User.

(C) All Videos of the same Channel must have different titles.

(D) A Video uniquely identifies the user who has uploaded that video to one of his or her own channels.

(E) A Playlist cannot be empty but must contain at least 1 video.

(F) For a given Playlist, the value of `position` must be unique and between 1 and N (with N being the number of videos in that Playlist)

(G) `num_views` must be greater or equal to 1.

(H) A video has 1 or more hashtags.

F1 Database Schema:

- countries(name, subregion, region)
- seasons(year, url)
- circuits(name, location, country, lat, lng, alt, url)
- drivers(forename, surname, code, number, dob, country, url)
- constructors(name, country, url)
- statuses(id, text)
- races(date, season, name, circuit, url)
- results(race, driver_forename, driver_surname, constructor, number, position, position_text, position_order, points, laps, time, fastest_lap, fastest_lap_time, fastest_lap_speed, status)
- constructor_results(race, constructor, points)
- pit_stops(race, driver_forename, driver_surname, lap, duration)
- qualifyings(race, driver_forename, driver_surname, number, constructor, position, q1, q2, q3)
- sprint_results(race, driver_forename, driver_surname, number, constructor, position, position_text, position_order, points, laps, time, fastest_lap, fastest_lap_time, status)

F1 Database Foreign Key (FK) constraints:

- (circuits.country) ⇝ (countries.name)
- (drivers.country) ⇝ (countries.name)
- (constructors.country) ⇝ (countries.name)
- (races.season) ⇝ (seasons.year)
- (races.circuit) ⇝ (circuits.name)
- (results.race) ⇝ (races.date)
- (results.driver_forename, results.driver_surname) ⇝ (drivers.forename, drivers.surname)
- (results.constructor) ⇝ (constructors.name)
- (results.status) ⇝ (statuses.id)
- (constructor_results.race) ⇝ (races.date)
- (constructor_results.constructor) ⇝ (constructors.name)
- (pit_stops.race) ⇝ (races.date)
- (pit_stops.driver_forename, pit_stops.driver_surname) ⇝ (drivers.forename, drivers.surname)
- (qualifyings.race) ⇝ (races.date)
- (qualifyings.driver_forename, qualifyings.driver_surname) ⇝ (drivers.forename, drivers.surname)
- (qualifyings.constructor) ⇝ (constructors.name)
- (sprint_results.race) ⇝ (races.date)
- (sprint_results.driver_forename, sprint_results.driver_surname) ⇝ (drivers.forename, drivers.surname)
- (sprint_results.constructor) ⇝ (constructors.name)
- (sprint_results.status) ⇝ (statuses.id)

SQL Queries

In this part of the exam your task is to formulate queries in SQL. We use the same database that we introduced in Assignment 1. This means you can directly run and test your queries using `psql` or `pgAdmin`. If needed, we provide a condensed representation the database schema in the Appendix.

Instructions. To ensure successful submission and grading of your SQL queries adhere to the following instructions:

- For each question, you are to write a **single SELECT SQL statement** to answer the question.
- You are only allowed to use SQL constructs that have been covered in the lectures.
- Each answer must be a syntactically valid SQL query and executable on PostgreSQL. Test with `psql` or `pgAdmin`!
- Each question must be answered independently of any other questions.
- You must not enter any extraneous text for your answer (e.g., "My answer is: ...), or enter multiple answers (i.e., submit multiple SQL statements). Your answer should not contain any comments.
- Note: The exact names of the columns in the result do not matter!

Recommendations. We recommend the following steps to answer each question:

- Write the answer query using your IDE or text editor of choice, and test it on your machine with `psql` or `pgAdmin`.
- Once you are happy with your query, copy the `SELECT` statement for the question into the corresponding answer field in Examplify.