Databases (TDA357/DIT621)

Home Exam on 20 March 2020 at 8:30-12:30

- Department of Computer Science and Engineering, Chalmers University of Technology & University of Gothenburg.
- Examiner: Thomas Hallgren, will be answering questions via email, Inbox in Canvas or Direct Messages in the course Slack (if you haven't joined Slack yet, there is an invitation to slack in this announcement).
- Results: Will be published within three weeks from exam date.
- Maximum number of points: 60.
- Grade limits for Chalmers: 24p for 3, 36p for 4, 48p for 5.
- Grade limits for GU: 24p for G, 42p for VG.
- Help material: this is a home exam, you can use any help material.

Instructions

- You can answer in English or Swedish.
- Indicate clearly if you make any assumptions that are not given in the question. In particular: in SQL questions, use standard SQL or PostgreSQL. If you use any other variant (such as Oracle or MySQL), say this; but full points are not guaranteed since this may change the nature of the question.

Submitting answers

- The exam is made available as an assignment in Canvas and you should submit your answers in Canvas. Write your answers in files that you upload.
- If you write the answers on your computer: submit them in `.txt`, `.sql`, `.pdf` or `.docx` files.
- If you use pen and paper: scan your answers or take photos of them and submit them as `.pdf` or `.jpg` files. Take photos in good light, and if possible, use an app that has a *document scanning mode* (e.g. the Notes app in iOS) to straighten the photos and adjust the brightness.
- Clearly mark the answer to each question so that they are easy to find.
- You can write all answers in one file, or put the answer to each question in a separate file. If you submit many files, include the question number in the file name, e.g. Q1.pdf, Q2.txt, Q3.txt, etc.
- You can submit multiple times, but only the last submission will be graded.

1 Entity-Relationship Modelling (10p)

1.1 E-R diagram (5p)

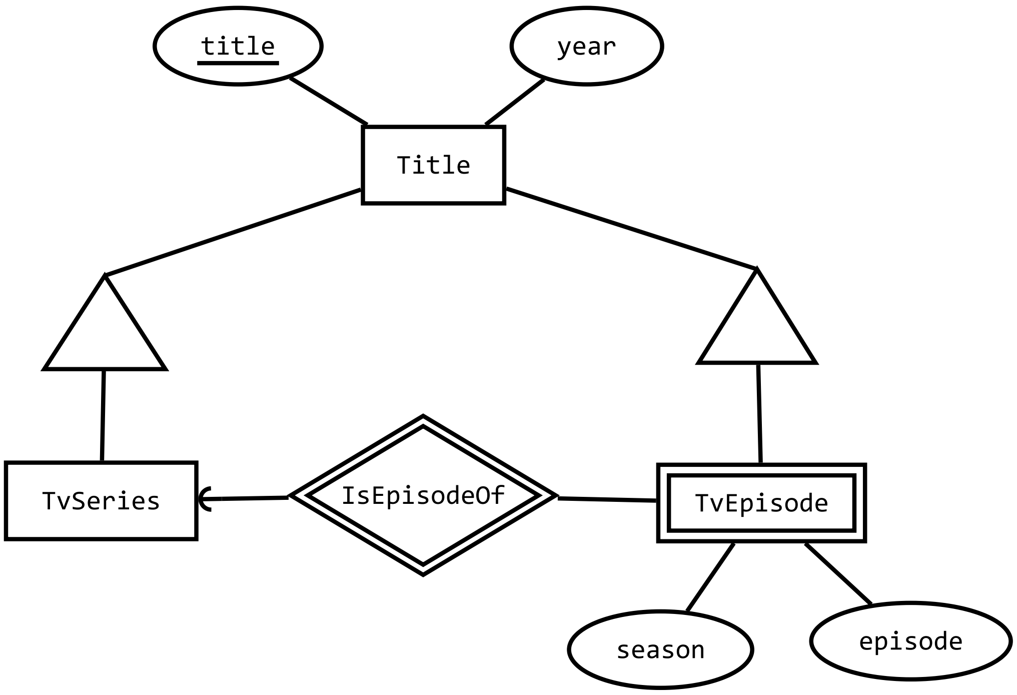
Draw an E-R diagram that describes the following domain:

- A book is identified by its ISBN number. A books also has a title and year of publication.
- An author has a name and a year of birth. Both the name and the year of birth are needed to uniquely identify an author.
- A book is written by one or more authors.
- A book consists of several chapters. Each chapter has a number and a chapter title. Chapter numbers are unique within a book.
- A chapter has an optional chapter author (i.e. there is at most one chapter author for each chapter).

Clarification at 9:51: domain descriptions often are a bit vague. If you need to make any assumptions, write them down in your answer.

1.2 From E-R model to database schema (5p)

Translate this E-R diagram into a relational schema, including keys and constraints.



Clarification at 9:18: The triangles in the ER-diagram are ISA relationships.

2 Functional Dependencies (8p)

Consider the relationship `Bookings(course,name,timeslot,lab,seats)`, for bookings of courses in computer labs:

course	name	timeslot	lab	seats
TDA357	Databases	Monday 10-12	ED3507	30
TDA357	Databases	Friday 13-15	ED3507	30
TDA555	Intro to FP	Monday 10-12	ED3354	16
TDA555	Intro to FP	Monday 10-12	ED3358	16
TDA555	Intro to FP	Friday 15-17	ED3507	30

2.1 Redundancies and inconsistencies (2p)

There is some redundant information in the above table, as evidenced by the repeated values in the `name` and `seats` columns.

Give an example of an SQL `UPDATE` statement that updates the number of seats in ED3507 to 40, without introducing any inconsistency.

Also give an example of a SQL `UPDATE` statement that introduces an inconsistency in the table.

2.2 Dependencies and keys (3p)

Write down the functional dependencies and the possible keys. In addition to the redundancies, consider also that it shouldn't be possible to book more than one course in the same lab at the same time.

2.3 Normalization (3p)

Which functional dependencies are BCNF violations? Show the result of BCNF decomposition, i.e. the relational schemas with keys and constraints, and the contents resulting tables.

3 SQL queries (12p)

Below is the schema for a database of movies, movie stars (actors, actresses, etc) and movie ratings.

```
Movies(_title,releaseYear)

MovieStars(_name,birthYear)

AppearsIn(_name,_title)
    name -> MovieStars(name)
    title -> Movie(title)

Ratings(_title,averageRating,numberOfVotes)
    title -> Movies(title)
```

Correction at 9:42: `name -> Stars(name)` was corrected to `name -> MovieStars(name)`

Correction after the exam: `MoviesStars` to `MovieStars`

3.1 (2p)

Write an SQL query that lists the titles of all movies with an average rating of at least 8.5 and at least 10000 votes.

3.2 (3p)

Write an SQL query that lists the movies (title and release year) in which both Max von Sydow and Angela Lansbury have appeared.

3.3 (3p)

Write an SQL query that lists the names of movie stars that were very young (less than 16 years old) when they first appeared in a movie.

3.4 (4p)

Write an SQL query that for each year ≥1970 lists highest rated movie released in that year. Only consider movies with at least 10000 votes. If more than one movie received the same (highest) rating in the same year, include them all in the listing.

Part of the result might look something like this:

```
 year | rating |                       title
------+--------+---------------------------------------------------
 …
 1998 |    8.6 | Saving Private Ryan
 1999 |    8.8 | Fight Club
 2000 |    8.5 | Gladiator
 2001 |    8.8 | The Lord of the Rings: The Fellowship of the Ring
 2002 |    8.7 | The Lord of the Rings: The Two Towers
 2003 |    8.9 | The Lord of the Rings: The Return of the King
 2004 |    8.5 | Black Friday
 2004 |    8.5 | The Lizard
 2005 |    8.7 | Earthlings
 …
```

4 Algebra and theory (8p)

4.1 (4p)

Consider again the database schema from question 3. Write a relational algebra expression that calculates the set of movie titles, paired with their average rating, in which Tom Cruise has appeared. The result set might include the following movies:

averageRating	title
7	Jack Reacher
7	Oblivion
7.9	Edge of Tomorrow
7.4	Mission: Impossible - Rogue Nation
6.1	Jack Reacher: Never Go Back
5.4	The Mummy

4.2 (4p)

Give an example of an SQL query that has no direct translation to relational algebra. Explain what the problem is.

5 Views, Constraints and Triggers (12p)

Consider again the database schema from question 3. Also consider an implementation of a database with this schema, where

- There is also a relation `Users(username, …)` for registered users of the database.
- The relation `Ratings` is not implemented as a table, but as a view created from a table `Votes(title,username,vote)` containing the individual votes from users of the database.
- The SQL type `TEXT` is used for all names and titles.

5.1 Constraints (4p)

Write the `CREATE TABLE` statements that implements the relation `Votes(title,username,vote)`. The following constraints should be enforced:

- Only registered users are allowed to vote.
- You can only vote for movies listed in the `Movies` table.
- Each user can only vote for each movie once.
- A vote is one of numbers 0, 1, 2, … or 10.

5.2 Views (4p)

Write the `CREATE VIEW` statement for the `Ratings` view.

Clarification at 11:07: the view should be like the `Ratings(title,averageRating,numberOfVotes)` relation in question 3.

5.3 Triggers (4p)

Movie titles sometimes change. A movie might be known under a working title (e.g. Mission Impossible 4) before it is released and an official title (e.g. Mission Impossible - Ghost Protocol) after it is released, so we might want to make updates like the following:

```
UPDATE Movies SET   title='Mission Impossible - Ghost Protocol'
              WHERE title='Mission Impossible 4';
```

The best way to prevent updates like this from causing problems is probably to use `ON UPDATE CASCASE` in all tables that have references to `Movies(title)`. But suppose for some reason `ON UPDATE CASCASE` can not be used and you have to use a trigger instead, how would you define a trigger that propagates updates of movie titles to the `AppearsIn` and the `Votes` tables?

You may assume that the checks that foreign key constraints remain valid are not performed immediately after each update, but instead postponed until the end of the transaction.

6 Semi-structured data (10p)

Consider again the database schema from question 3.

6.1 (5p)

Write an SQL query that creates one big JSON document containing filmographies for all movie stars, i.e. the document should be an array that for each movie star contains an object with the name, birth year and the list of movies that the star has appeared in. Each movie is an object containing the title, release year and average rating of the movie.

Hint: the functions `jsonb_agg` and `jsonb_build_object` will probably be useful.

6.2 (5p)

Write down the JSON Schema for the document you created in 6.1.

The schema should contain enough detail about the structure of the JSON document so that if you validate the document you created in 6.1 against the schema you will detect if there is a problem with it, e.g. if the document has the wrong structure, contains the wrong type of information or if some information is missing.

Databases

TDA357 (Chalmers), DIT621 (University of Gothenburg)

Home Exam on 20 March 2020 at 8:30-12:30

Suggested solutions

…

1 Entity-Relationship Modelling (10p)

1.1 E-R diagram (5p)

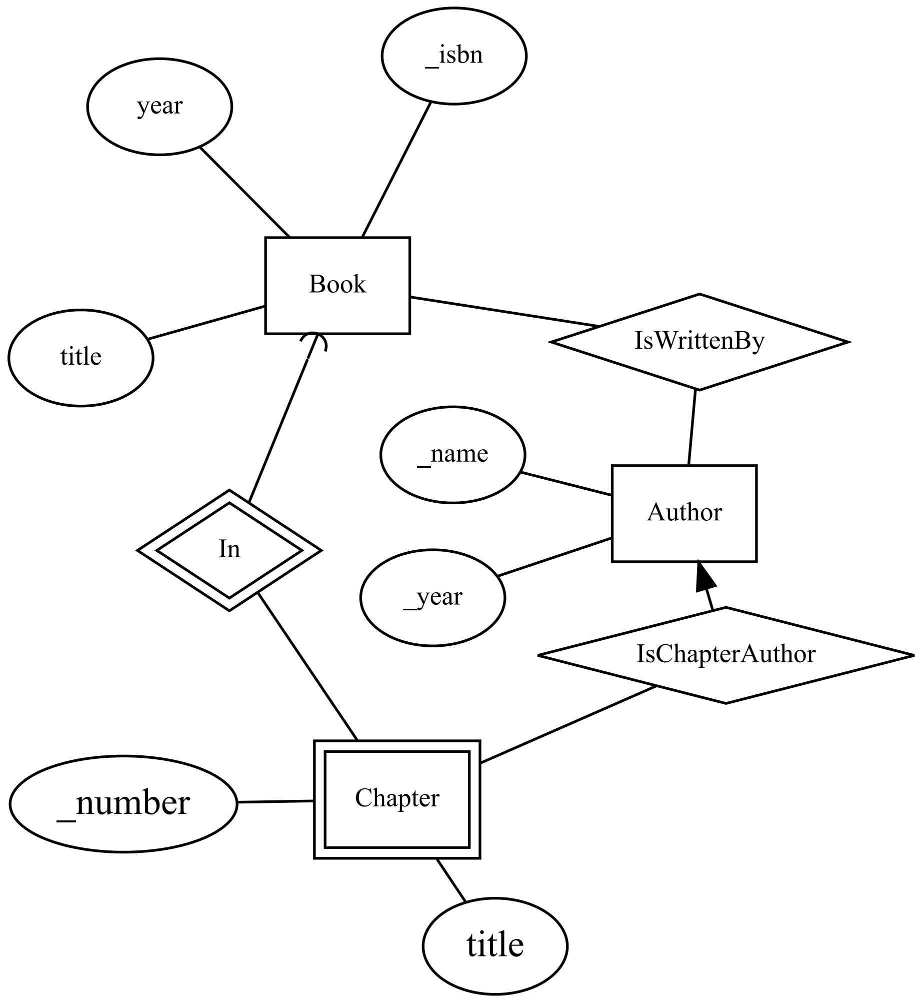
```
ENTITY Book : _isbn title year

ENTITY Author : _name _year

RELATIONSHIP IsWrittenBy Book -- Author

WEAK ENTITY Chapter Book In : _number title

RELATIONSHIP IsChapterAuthor Chapter -> Author
```



1.2 From E-R model to database schema (5p)

```
Title(_title,year)

TvSeries(_title)
  title -> Title(title)

TvEpisode(_title,_seriesTitle,season,episode)
title -> Title(title)
seriesTitle -> TvSeries(title)
```

2 Functional Dependencies (8p)

2.1 Redundancies and inconsistencies (2p)

An update the keeps the information consistent:

```
UPDATE Bookings SET seats=40 WHERE lab='ED3507'
```

An update that introduces an inconsistency:

```
UPDATE Bookings SET seats=40 WHERE lab='ED3507' AND course='TDA555';
```

2.2 Dependencies and keys (3p)

Functional dependencies:

```
course -> name
lab -> seats
lab timeslot -> course
```

There is only one possible key: `lab timeslot`

2.3 Normalization (3p)

BCNF violations

```
course->name
lab->seats
```

Relations after BCNF decomposition

```
Bookings(course,_timeslot,_lab)
  course -> Courses(course)
  lab -> Labs(lab)

Courses(_course,name)

Labs(_lab,seats)
```

Tables

Courses

course	name
TDA357	Databases
TDA555	Intro to FP

Labs

lab	seats
ED3507	30
ED3354	16
ED3358	16

Bookings

course	timeslot	lab

course	timeslot	lab
TDA357	Monday 10-12	ED3507
TDA357	Friday 13-15	ED3507
TDA555	Monday 10-12	ED3354
TDA555	Monday 10-12	ED3358
TDA555	Friday 15-17	ED3507

3 SQL queries (12p)

3.1 (2p)

```
SELECT title
FROM Ratings
WHERE averageRating>=8.5 AND numberOfVotes>=10000;
```

3.2 (3p)

```
    SELECT title,releaseYear
    FROM AppearsIn NATURAL JOIN Movies
    WHERE name='Max von Sydow'
INTERSECT
    SELECT title,releaseYear
    FROM AppearsIn NATURAL JOIN Movies
    WHERE name='Angela Lansbury'
```

Another solution:

```
WITH MaxMovies AS (
  SELECT title
  FROM AppearsIn
  WHERE name = 'Max von Sydow'
),
AngelaMovies AS (
  SELECT title
  FROM AppearsIn
  WHERE name = 'Angela Lansbury'
)
SELECT Movies.title, releaseYear
FROM Movies, MaxMovies, AngelaMovies
WHERE Movies.title = MaxMovies.title
AND   Movies.title = AngelaMovies.title;
```

Yet another solution:

```
SELECT M.title,releaseYear
FROM Movies AS M
     INNER JOIN AppearsIn AS A1
             ON A1.title=M.title AND A1.name='Angela Lansbury'
     INNER JOIN AppearsIn AS A2
             ON A2.title=M.title AND A2.name='Max von Sydow';
```

And yet another solution:

```
SELECT title,releaseYear
FROM Movies NATURAL JOIN AppearsIn
WHERE name='Max von Sydow'
  AND title IN (SELECT title FROM AppearsIn WHERE name='Angela Lansbury');
```

Another variant:

```
SELECT * FROM Movies
WHERE 'Max von Sydow' IN (SELECT name FROM AppearsIn
                          WHERE AppearsIn.title = Movies.title)
AND 'Angela Lansbury' IN (SELECT name FROM AppearsIn
                          WHERE AppearsIn.title = Movies.title);
```

Even more ways:

```
SELECT title, releaseYear
FROM Movies
WHERE (SELECT COUNT(name)
       FROM AppearsIn
       WHERE title=Movies.title AND
             name IN ('Max von Sydow', 'Angela Lansbury')) = 2;
```

3.3 (3p)

```
SELECT DISTINCT name
FROM Movies NATURAL JOIN AppearsIn NATURAL JOIN MovieStars
WHERE releaseYear-birthYear<16
```

Alternative solution:

```
WITH FirstAppearance AS (
    SELECT name,MIN(releaseYear) AS firstYear
    FROM Movies NATURAL JOIN AppearsIn
    GROUP BY name)
SELECT name
FROM FirstAppearance NATURAL JOIN MovieStars
WHERE firstYear-birthYear<16
```

Yet another solution:

```
WITH AppearancesWithAge AS (
    SELECT name,title,releaseYear-birthYear AS age
    FROM Movies NATURAL JOIN AppearsIn NATURAL JOIN MovieStars
)
SELECT DISTINCT name
FROM AppearancesWithAge
WHERE age<16
```

3.4 (4p)

```
WITH
  RatedMovies AS (
    SELECT releaseYear AS year,title,averageRating
    FROM Movies NATURAL JOIN Ratings
    WHERE releaseYear>=1970 and numberOfVotes>=10000),
  TopRatings AS (
    SELECT year,MAX(averageRating) AS topRating
    FROM RatedMovies
    GROUP BY year)
SELECT year,topRating AS rating,title
FROM RatedMovies NATURAL JOIN TopRatings
WHERE averageRating=topRating
ORDER BY year;
```

This should also work (in principle, it took too long in my test):

```
SELECT releaseYear,averageRating,title
FROM Movies AS M NATURAL JOIN Ratings
WHERE releaseYear>=1970 AND numberOfVotes>=10000
  AND averageRating = (SELECT MAX(averageRating)
                       FROM Movies NATURAL JOIN Ratings
                       WHERE numberOfVotes>=10000
                         AND releaseYear = M.releaseYear);
```

4 Algebra and theory (8p)

4.1 (4p)

$\pi_{\text{averageRating,title}}$ ($\sigma_{\text{name='Tom Cruise'}}$ (AppearsIn ⋈ Rating))

Or:

$\pi_{\text{averageRating,title}}$ (($\sigma_{\text{name='Tom Cruise'}}$ AppearsIn) ⋈ Rating)

4.2 (4p)

```
SELECT name
FROM Countries
WHERE name IN (SELECT capital FROM countries);
```

This would be translated to π_{name} ($\sigma_{\mathit{condition}}$ Countries) in relational algebra, where *condition* comes from WHERE clause in the SQL query, which in this example contains another query. The problem is that the conditions in the sigma operator are not allowed to contain relational algebra operators.

5 Views, Constraints and Triggers (12p)

5.1 (4p)

```
CREATE TABLE Votes(
    title    TEXT REFERENCES Movies(title),
    username TEXT REFERENCES Users(username),
    vote     INT  NOT NULL CHECK (0<=vote AND vote<=10),
                            -- (vote BETWEEN 0 AND 10)
    PRIMARY KEY (title,username)
);
```

5.2 (4p)

```
CREATE VIEW Ratings AS (
  SELECT title,AVG(vote) AS averageRating,COUNT(vote) AS numberOfVotes
  FROM Votes
  GROUP BY title
);
```

5.3 (4p)

```
CREATE OR REPLACE FUNCTION propagate_title() RETURNS TRIGGER AS $$
BEGIN
  IF NEW.title<>OLD.title THEN
    UPDATE AppearsIn SET title=NEW.title WHERE title=OLD.title;
    UPDATE Votes     SET title=NEW.title WHERE title=OLD.title;
  END IF;
  RETURN NULL;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER propagate_title AFTER UPDATE ON Movies
  FOR EACH ROW EXECUTE FUNCTION propagate_title();
```

6 Semi-structured data (10p)

6.1 (5p)

```
WITH Filmographies AS (
    SELECT jsonb_build_object(
                'name',name,
                'birthYear',MAX(birthYear),
                'movies',jsonb_agg(jsonb_build_object(
                                       'title',title,
                                       'year',releaseYear,
                                       'rating',averageRating))) AS filmography
    FROM Movies NATURAL JOIN MovieStarts
                NATURAL JOIN AppearsIn
                NATURAL JOIN Ratings
    GROUP BY name)
SELECT jsonb_agg(filmography) AS filmographies
FROM Filmographies;
```

6.2 (5p)

```
{"type":"array",
 "items":{
     "type":"object",
     "required":["name","birthYear","movies"],
     "properties":{
         "name":{"type":"string"},
         "birthYear":{"type":"number"},
         "movies":{"type":"array",
                   "items":{
                       "type":"object",
                       "required":["title","year","rating"],
                       "properties":{
                           "title":{"type":"string"},
                           "year":{"type":"number"},
                           "rating":{"type":"number"}
                       }
                   }
                  }
     }
 }
}
```