Databases

TDA357/DIT622/DIT621 (4.5 hec)

Responsible: Ana Bove, tel: 1020

Friday 21st of March 2025, 8:30–12:30

Total: 60 points
TDA357/DIT622: $\geqslant$ 27: 3, $\geqslant$ 38: 4, $\geqslant$ 49: 5
DIT621: $\geqslant$ 27: G, $\geqslant$ 45: VG

Make sure your handwriting and drawings are readable!
What we cannot read we cannot correct!

The exam has **6 questions**. Make sure to turn pages! :)

Note: As said in the course page, if you brought **one hand written double sided A4** of notes to the exam, you **must hand it in** along with your solutions.

Good luck!

Note: The exm turned out to be 60.5 pts in total since part 2 is 12 points and not 11.5. The amount of points needed for each grade will be the same as stated above.

1 SQL and Constraints (7.5 pts)

Here we work in the domain of users, channels and messages efter the following (incomplete) relational schema:

Users (id, nickname, role)
Channels (name, status)
Messages (nr, sender, channel, content, reply)
sender → Users.id
channel → Channels.name

Users have an id number which identifies them, a nickname and a *role* which at the moment can only be of two kinds: admin or student (but in the future one might allow more possible roles).

Channels have a unique name and a *status* flag indicating whether the channel is open or not. Students can only send messages to open channels, admin users can send messages to all channels.

Each message has a unique identifying number, is send by a user in an specific channel, and has a text content. Some messages are a reply to some other message, in which case *reply* will contain the identification number of the message that is being replied to.

a) (1.5pts) Without giving the complete tables, give the corresponding SQL definitions for the attributes (sometimes also called columns) *roles*, *status* and *reply* in the tables which will result from the schema above.

b) (1.5pts) Does the current schema guarantee that deleting a message will also delete all its replies?

If yes, explain why. If not, explain how this could be guaranteed using SQL (no need for a full implementation).

c) (2pts) Does the current schema guarantee that a reply is sent to the *same* channel as the message that is being replied to?

If yes, explain why. If not, explain how this could be guaranteed using SQL (no need for a full implementation).

d) (2.5pts) We want to allow a user to block another user for a certain number of days. For this purpose we add a new table, with the following (incomplete) schema, to keep track of the blocks that are *currently in place* (that is, only active blocks are kept in the table, you do not need to worry about removing expired blocks):

Blocks (blocker, blocked, start, period)

This means that the if user A (the *blocker*) blocks user B (the *blocked*) today (the *start*) and for 3 days (the *period*), then user B cannot reply to any of the messages A sends during the following 3 days.

Give the full SQL definition of the table *Blocks*.

Solution:

```
CREATE TABLE Users (
  id INT PRIMARY KEY,
  nickname TEXT NOT NULL,
  role TEXT NOT NULL CHECK (role IN ('admin','student')));

CREATE TABLE Channels (
  name TEXT PRIMARY KEY,
  open BOOLEAN NOT NULL);

CREATE TABLE Messages (
  nr INT PRIMARY KEY,
  sender INT NOT NULL REFERENCES Users,
  channel TEXT NOT NULL REFERENCES Channels,
  content TEXT NOT NULL,
  reply INT REFERENCES Messages);
```

a) See the definition of the attributes in the tables above.
It is enough to simply give the corresponding lines.
Important information is the type of the attribute, whether it can be empty or not, and any additional constraint.

I used the boolean type for `status` so I called this attribute `open` instead. Another possibility would be to give the following definition for `status`:

```
status TEXT NOT NULL CHECK (status IN ('open', 'close'))
```

b) No, `reply` would need to be defined as
```
reply INT REFERENCES Messages ON DELETE CASCADE
```

c) No, the is nothing in the defintion of the tables guaranteeing this.

A foreign key reference
```
FOREIGN KEY (reply, channel) REFERENCES Messages.(nr, channel)
```
in `Messages` will do this.

Recall that `FOREIGN KEY` need to refer to `UNIQUE` values so we will need to even add a `UNIQUE (channel, reply)` constraint.

Observe that just a `UNIQUE (channel, reply)` constraint will only guarantee that all replies of a message are in the same channel, but this channel might not be the same channel as the one the original message was sent to!

d)
```
CREATE TABLE Blocks (
  blocker INT REFERENCES Users,
  blocked INT REFERENCES Users,
  start DATE NOT NULL DEFAULT CURRENT_DATE,
  period INT NOT NULL CHECK (period > 0),
  PRIMARY KEY (blocker, blocked));
```

No need for the part `DEFAULT CURRENT_DATE` but the check for `period` needs to be there.

2 SQL and RA (12 pts)

We continue with the same domain as in question 1 on SQL with the constraints you defined previously:

Users (<u>id</u>, nickname, role)
Channels (<u>name</u>, status)
Messages (<u>nr</u>, sender, channel, content, reply)
  sender → Users.id
  channel → Channels.name
Blocks (blocker, blocked, start, period)

a) (2.5+3pts) Write an SQL query and a relational algebra expression that output the list of users' nicknames and the number of messages (including replies) each of the users has sent. The output has to be ordered in decreasing order with respect to the numbers of sent messages.

b) (3pts) Write a relational algebra expression that for *every user* in the domain, outputs the id and nickname of the user together the with sum of the lengths of the contents in

the messages that the user has sent. To compute the length of a text t simply use the function LENGTH(t).

c) (3.5pts) Write an SQL query that for *each message*, outputs the nickname of the sender. If the message is a reply, your query should also output the nickname of the user that sent the message the current message is replying to. Your output should then have 3 columns; order it by the message's number.

Solution:

a)
```
SELECT nickname, COUNT(*) AS cnt
FROM Users JOIN Messages ON id = sender
GROUP BY id, nickname
ORDER BY cnt DESC;
```

(recent POSTGRES version also allow just `GROUP BY id`).

This answer only outputs the users who HAVE sent messages. If your answer outputs ALL users, then you need to make sure not to use `COUNT(*)` since this will give 1 even for those users who have not sent messages. In this case you need to use for example `COUNT(nr)`, this will give 0 is no message number for that user. Same comment applies to the corresponding RA.

RA:

$$\tau_{-\mathsf{cnt}}(\pi_{\mathsf{nickname,cnt}}(\gamma_{\mathsf{id,nickname,COUNT}(*)\rightarrow\mathsf{cnt}}(\mathsf{Users} \bowtie_{\mathsf{id=sender}} \mathsf{Messages})))$$

Observe that in recent POSTGRES versions, it is ok to simply group by `id` and output `nickname` because `id` is the primary key of the table with the attribute `nickname`.

However, in the relational algebra, if we don't have nickname in the γ operator then we are not able to output it since the γ operator also performs a projection.

b) Here it was emphasised thay every user should be part of the output so only an OUTER JOIN is possible.

OK to do the COALESCE after the SUM instead.

$$\tau_{\mathsf{len\_sum}}(\gamma_{\mathsf{id,nickname,SUM(COALESCE(LENGTH(content),0))}\rightarrow\mathsf{len\_sum},}(\mathsf{Users} \bowtie^{\mathsf{OL}}_{\mathsf{id=sender}} \mathsf{Messages}))$$

c)
```
WITH MsgSenders AS
 (SELECT nr, nickname AS sender, reply
  FROM Users JOIN Messages ON id = sender)
SELECT Msg.nr AS MsgNr, Msg.sender, Reply.sender AS replyingto
FROM MsgSenders AS Msg LEFT OUTER JOIN MsgSenders AS Reply
     ON Msg.reply = Reply.nr
ORDER BY MsgNr;
```

3 Views and Triggers (10 pts)

We continue with the same domain as in question 1 on SQL with the constraints you defined previously:

Users (id, nickname, role)
Channels (name, status)
Messages (nr, sender, channel, content, reply)
    sender → Users.id
    channel → Channels.name
Blocks (blocker, blocked, start, period)

Propose a solution (meaning, the corresponding full SQL code) to the following tasks that need to be performed:

a) (4pts) Keep track of how much every user has blocked others and has been blocked by others, ordered by the user id.

   That is, for *each user* A in the domain we want to keep track of the `id` and `nickname` of the user, the number of other users A is currently blocking, and the number of users that are currently blocking A.

b) (6pts) As mentioned before, students can only send messages in open channels while users with admin roles can send messages to all channels. Also, if user A is currently blocking user B, then B cannot reply to a message that A has sent.

   Make sure these restrictions are taken care of in a proper way every time a user is sending a message in a channel.

Solution:

a) Each user should be part of the solution so an `OUTER JOIN` is needed.

```
CREATE OR REPLACE VIEW BlocksTable AS
WITH
  NrBlocksPerUser AS
  (SELECT id, nickname, COUNT(blocker) AS nrblocks
   FROM Users LEFT OUTER JOIN Blocks ON id = blocker
   GROUP BY id),
  NrBlockedPerUser AS
  (SELECT id, nickname, COUNT(blocked) AS nrblocked
   FROM Users LEFT OUTER JOIN Blocks ON id = blocked
   GROUP BY id)
SELECT id, nickname, nrblocks, nrblocked
FROM NrBlocksPerUser NATURAL JOIN NrBlockedPerUser
ORDER BY id;
```

Observe that in this solution, using `COUNT(*)` instead of `COUNT(blocker)` (`COUNT(blocked)`) will count the number of rows per user and for those with no blocks or not being blocked the count will be 1 instead of 0!

b)

```
CREATE OR REPLACE FUNCTION send_message() RETURNS TRIGGER AS $$
DECLARE
  replyto INT;
BEGIN
  IF NOT (SELECT open FROM Channels WHERE name = NEW.channel)
     AND (SELECT role FROM Users WHERE id = NEW.sender) = 'student'
  THEN RAISE EXCEPTION 'Students cannot send messages to closed channels';
  ELSIF NEW.reply IS NOT NULL
     THEN replyto = (SELECT sender FROM Messages WHERE nr = NEW.reply);
          IF EXISTS (SELECT * FROM Blocks
                     WHERE blocker = replyto AND blocked = NEW.sender)
          THEN RAISE EXCEPTION 'You are being blocked by the user you reply to';
          END IF;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER SendMessage
  BEFORE INSERT ON Messages
  FOR EACH ROW
  EXECUTE FUNCTION send_message();
```

Defining the trigger `AFTER INSERT` will be incorrect.

4 ER Modelling (10 pts)

a) (6pts) Make an ER-diagram for the following domain of books and writers. The database should satisfy these conditions:

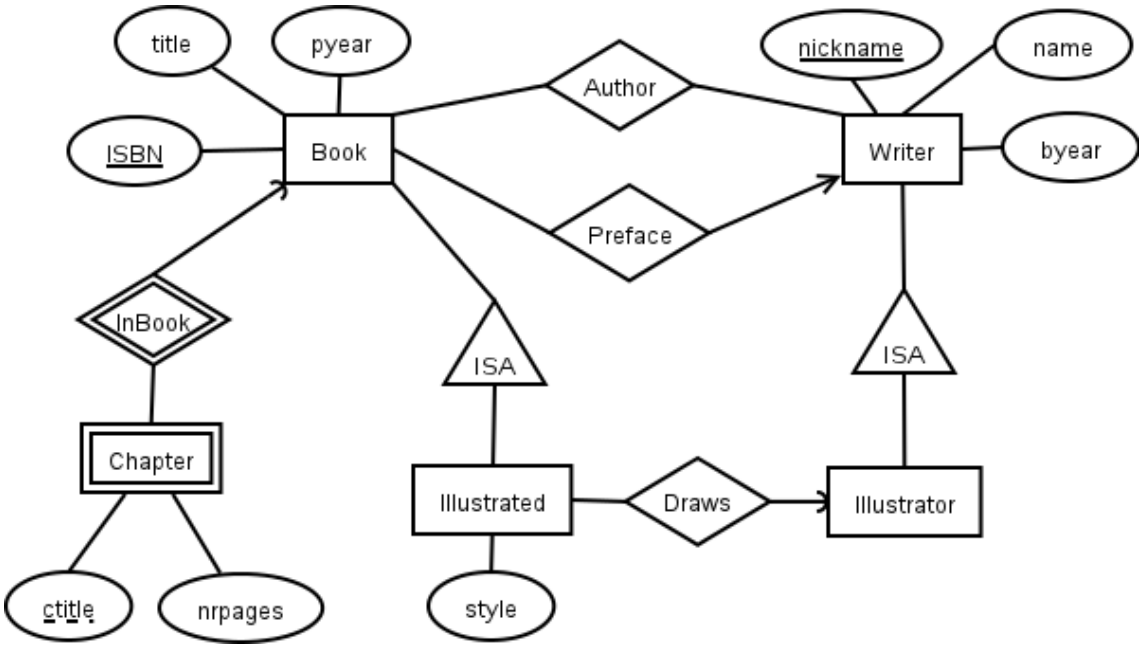
- Writers have a unique nickname, a name, and a year of birth.
- Some writers are also illustrators.
- Books are identified by their ISBN number. They have a title, a year of publication and one or more writers as authors of the book.
- Some books are illustrated books. In that case, there is one illustrator that has done all the drawings in the book. Illustrated books have also information about the style of the drawings (for example aquarelle, oil, etc.)

- Books consist of chapters. Chapters have a title, but different books can have chapters with the same name. The database also stores the number of pages each chapter has.
- Some books may also have a preface written by a writer (though not necessarily one of the writers of the book).

b) (4pts) Translate your diagram into a relational schema that doesn't allow any `NULL` values.

Solution:

a)



Illustrated book could instead be modelled as a many-to-at-most-one relationship to Illustrators. The style should then be an attribute of the relationship. Even in a solution like the one above one could think of style as an attribute to the relationship.

b)

Books (ISBN, title, pyear)
Chapters (ISBN, ctitle, nrpages)
    ISBN → Books.ISBN
Illustrated (ISBN, nickname, style)
    ISBN → Books.ISBN
    nickname → Illustrators.nickname
Writers (nickname, name, byear)
Illustrators (nickname)
    nickname → Writers.nickname
Authors (ISBN, nickname)
    ISBN → Books.ISBN
    nickname → Writers.nickname
Prefaces (ISBN, nickname)
    ISBN → Books.ISBN
    nickname → Writers.nickname

5 Functional and Multivalued Dependencies (11pts)

A company has a web shop with a poorly designed database that currently has two tables:

Customers (cid, email, street, city, name, category, catTag)
LogTable (cid, prodId, quantity, price, prodName)
    cid → Customer.cid

The meaning of the attributes are as follows:

- cid: a customer ID (just a unique number);
- email: a customer email, used for things like logging into the web shop;
- street: part of a customer address, note that each customer can have multiple addresses;
- city: also part of a customer address;
- name: the name of a customer;
- category: a category a user has shown interest in; users may be interested in several categories;
- catTag: this is a tag for a category (a short text); each category can have multiple tags;
- prodId: unique identifier for a product;
- quantity: how many units of the product a user has purchased since the user became a customer in the web shop;

- price: this is the current price of a product;
- prodName: the name of a product; different products can have the same name.

When using the database, the company finds themselves repeating a lot of information. For instance, having two customers, each with two addresses and interested in two categories that each have two tags seems to require 16 rows in Customers!

The company then hires you to normalise the database and get rid of the redundancy. Give reasonable names to the relations in the process.

a) (3pts) List all functional dependencies (that cannot be derived from each other) that are BCNF violations in either Customers or LogTable.

b) (4pts) Normalise the database to BCNF. It is sufficient to give the final schema, not the intermediate steps. Mark primary keys and any secondary keys (as UNIQUE constraints) in the schema.

c) (4pts) List the MVDs and further normalise the database schema to 4NF. Again mark keys in the new relations.

Hint: This requires more than one step of decomposition.

Solution:

a) CustomersList:
cid → email name
ProdAndSalesLog:
prodId → price prodName
cid prodId → quantity

b) Customers (<u>cid</u>, email, name)
UNIQUE email

Products (<u>prodId</u>, price, prodName)

SalesLog (<u>cid</u>, <u>prodId</u>, quantity)

R2 (<u>cid</u>, <u>street</u>, <u>city</u>, <u>category</u>, <u>catTag</u>)

c) Decompose R2 on
cid ↠ street city
(or equivalently on cid ↠ category catTag)

CustomerAddresses (<u>cid</u>, <u>street</u>, <u>city</u>)

R3 (<u>cid</u>, <u>category</u>, <u>catTag</u>)

Further decompose on category ↠ catTag to split R3 into

CategTags (<u>category</u>, <u>catTag</u>)
CustomerCategs (<u>cid</u>, <u>category</u>)

Note that doing these in the opposite order is also fine, and is perhaps a bit easier.

6 JSON (10 pts)

In this exercise you will use semi-structured documents to model the domain of universities with the following characteristics:

- Two universities cannot have the same name.
- Universities are divided into departments (for example Computer science, Mathematics, etc). There is always at least one department in every university.
- Universities have a rector, of whom we *must* know their name.
- The information that *must* be known from a department is its code (max. 5 characters!), the name of the head of the department, and the number of employees. Sometimes (but not always), information about the number of divisions in the department is also known.

This table shows a small example with two universities: Chalmers with the departments CSE and MV, and KTH with a CS department:

UnivName	**Rector**	**DCode**	**HoD**	**NrEmp**	**NrDiv**
Chalmers	MNJ	CSE	RT	300	15
		MV	MA	275	
KTH	MB	CS	DS	330	20

a) (3pts) Create **one** JSON **object** (that is, not an array) containing all the information about Chalmers and KTH which is included in the table above.

b) (5pts) Define a JSON schema that validates the document you created. The schema should ensure the following:

- All types are correct (for example, changing any string into an integer or vice versa should invalidate the document).
- Empty strings shouldn't be allowed and numbers of employees and of divisions need to be at least 1.
- Any constraint expressed in the description above needs to be satisfied.
- The "must know" information should always be present.

- No other information other than those mentioned above should be allowed.
- The schema should be general enough to allow more than the two universities in the table, and also any number of departments for those universities (though at least one).

c) (2pts) Write a JSON Path expression for finding the codes of all departments that have more employees than the CSE department at Chalmers, which you can assume is listed as the first department in Chalmers.

Solution:

a)
```
{"Chalmers":
     {"rector": "MNJ",
      "departments":[{"dcode":"CSE",
                      "hod":"RT",
                      "nremp":300,
                      "nrdiv":15},
                     {"dcode":"MV",
                      "hod":"MA",
                      "nremp":275}]},
  "KTH":
     {"rector": "MB",
      "departments":[{"dcode":"CSE",
                      "hod":"DS",
                      "nremp":330,
                      "nrdiv": 20}]}
}
```

b)
```
{"type": "object",
  "additionalProperties": {
    "type":"object",
    "properties": {
       "rector": {"$ref":"#/definitions/nonEmpStr"},
       "departments" :{"type":"array",
                       "minItems":1,
                       "items": {"$ref":"#/definitions/department"}}
    },
    "required": ["rector", "departments"],
    "additionalProperties": false
  },
  "definitions": {
    "nonEmpStr": {"type":"string", "minLength": 1},
    "department": {
       "type": "object",
       "properties":{
```

```
            "dcode": {"$ref":"#/definitions/nonEmpStr",
                      "maxLength": 5},
            "hod": {"$ref":"#/definitions/nonEmpStr"},
            "nremp": {"type":"integer", "minimum":1},
            "nrdiv": {"type":"integer", "minimum":1},
          },
          "required": ["dcode","hod","nremp"],
          "additionalProperties": false
     }
    }
   }
```

c) `'strict $.**?(@.nremp > $.Chalmers.departments[0].nremp).dcode'`