Databases

TDA357/DIT621 (4.5 hec)

Responsible: Ana Bove, tel: 1020

Thursday 24th of August 2023, 14:00–18:00

Total: 60 points	
CTH: $\geqslant$ 27: 3, $\geqslant$ 38: 4, $\geqslant$ 49: 5	GU: $\geqslant$ 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!

1 SQL, Constraints and Views (12 pts)

A client of a bank is either a private person or an organisation.

Private persons are identified by a personal number, have a name, a telephone number and an email address.

Organisations are identified by their organisation number, have a name and an address, and a director which is unique (meaning that there cannot be multiple client organisations with the same director). Directors are private persons even though they might not be clients of the bank themselves. Organisations also have an estimated value.

Private persons and organisations can loan money from the bank, and both can of course have more than one loan in the bank. Loans from an organisation are registered via the personal number of the director. An attribute (type) will tell if the loan is a "private" one or an "organisation" one, and these are the only two possible types of loans. Each loan can be bound for 1, 2, 3, or 5 years from a particular date which is recorded. The annual interest to be paid is set on case by case.

A (certainly incomplete) relational schema for this domain is the following:

Person (personal_nr, name, tel, email)
Organisation (org_nr, name, address, director, value)
    director → Person.personal_nr
Loan (id, client, type, amount, binding, bdate, interest)
    client → Person.personal_nr

You can assume that all years have 365 days. The SQL expression `CURRENT_DATE` gives you the date of today, `CURRENT_DATE + 1` will give you tomorrow's date and `CURRENT_DATE - 1` yesterday's. Also, subtracting a date from another date gives the number of days between them.

a) (2.5 pts) Define SQL tables for the relational schema above. Make sure to define ap-

propriate types and constraints, including possible missing ones in the schema.

b) (2.5pts) The table of loans includes loans whose binding period have passed and hence are no longer "active" loans. (Once the binding period ends the client would need to renew the loan and hence it will become a new entry in the table, or pay it back.)

Help the bank and create a view `ActiveLoans` which gives all the information of only the active loans.

Note: If you don't manage to have do this part (in a correct way), you can anyway use the `ActiveLoans` view as part of your solution to other questions.

c) (3.5pts) Write an SQL query that lists all the (active) loans whose binding period ends within the next 30 days together with the name and email address of the client.

The output should only contain the name of the client (name of the private person or of the organisation), the email address (of the private person or of the director of the organistion), and the number of days till the loan expires.

d) (3.5 pts) Write an SQL query that lists the personal number and name of all persons connected to the bank, and the kind of active loan(s) they might have. The kind of loan is "personal" when the person only has loan(s) as a private person, "org" when the person only has loan(s) in his/her role as director of an organisation, "both" in case the person has both loan(s) as a private person and as director of an organisation, or "none" if there is no loan associated to the person.

Order the output by the personal numbers.

Solution:

```
CREATE TABLE Persons (
  personal_nr INT PRIMARY KEY,
  name text NOT NULL,
  tel INT NOT NULL CHECK (tel > 0),
  email TEXT NOT NULL );

CREATE TABLE Organisations (
  org_nr INT PRIMARY KEY,
  name text NOT NULL,
  address TEXT NOT NULL,
  director INT NOT NULL UNIQUE REFERENCES Persons,
  value INT NOT NULL );
  -- Note that value could in principle be negative

CREATE TABLE Loans (
  id INT PRIMARY KEY,
  client INT NOT NULL REFERENCES Persons,
  type TEXT NOT NULL CHECK (type IN ('Private', 'Org')),
  amount INT NOT NULL CHECK (amount > 0),
```

```
  binding INT NOT NULL CHECK (binding IN (1,2,3,5)),
  bdate DATE NOT NULL,
  interest FLOAT NOT NULL CHECK (interest > 0 AND interest <= 100) );
  -- Alternatively the interest could be between 0 and 1


-- Active loans
CREATE OR REPLACE VIEW ActiveLoans As
SELECT *
FROM Loans
WHERE bdate >= CURRENT_DATE - 365 * binding;


-- Loans expiring within 30 days
-- Will not work if AboutToExpire uses DaysToExpire in WHERE
WITH AboutToExpire AS
  (SELECT *, bdate + binding * 365 - CURRENT_DATE AS DaysToExpire
   FROM ActiveLoans
   WHERE (bdate + binding * 365 - CURRENT_DATE) < 30)
(SELECT name, email, daystoexpire
FROM Persons JOIN AboutToExpire ON personal_nr = client
WHERE type = 'Private')
UNION
(SELECT Organisations.name, email, daystoexpire
FROM (Persons JOIN AboutToExpire ON personal_nr = client)
     JOIN Organisations ON personal_nr = director
WHERE type = 'Org');


-- Persons and their kind of loans
WITH MaxOneTypeLoan AS
  ((SELECT personal_nr, 'none' AS kind FROM Persons
    EXCEPT
    SELECT client, 'none' FROM ActiveLoans)
   UNION
   (SELECT client AS personal_nr, 'private' AS kind
    FROM ActiveLoans WHERE type = 'Private'
    EXCEPT
    SELECT client, 'private' FROM ActiveLoans WHERE type = 'Org')
   UNION
   (SELECT client AS personal_nr, 'org' AS kind
    FROM ActiveLoans WHERE type = 'Org'
    EXCEPT
    SELECT client, 'org' FROM ActiveLoans WHERE type = 'Private'))
SELECT personal_nr, name, COALESCE(kind,'both') AS kind
FROM MaxOneTypeLoan RIGHT JOIN Persons USING (personal_nr)
ORDER BY personal_nr;
```

2 More SQL and Relational Algebra (10 pts)

We continue with the same domain as in question 1 on SQL:

Person (personal_nr, name, tel, email)
Organisation (org_nr, name, address, director, value)
director → Person.personal_nr
Loan (id, client, type, amount, binding, bdate, interest)
client → Person.personal_nr

Note: Here you can assume that you have a working implementation of the AciveLoans view even if you have not solved that particular question (in a correct way).

a) (2.5+2.5pts) Write an SQL query and a relational algebra expression that lists the number and name of all the organisations that have at least one active loan which is higher than 3 times the estimated value of the organisation.

The output should only contain the number and name of the organisation, and be in descending order after the organisation number. Each organisation should appear at most once.

Solution:

```
SELECT DISTINCT org_nr, name
FROM Organisations JOIN ActiveLoans ON director = client
WHERE type = 'Org' AND amount > value * 3
ORDER BY org_nr DESC;
```

$$\tau_{-\mathsf{org\_nr}}(\delta(\pi_{\mathsf{org\_nr,name}}(\sigma_{\mathsf{type}='\mathsf{Org}'\wedge\mathsf{amount}>\mathsf{value}*3}(\mathsf{Organisations} \bowtie_{\mathsf{director}=\mathsf{client}} \mathsf{ActiveLoans}))))$$

b) (2.5+2.5pts) Write an SQL query and a relational algebra expression that lists the personal numbers and names of all persons that have (active) loans (counting both the private loans and those done as director of an organisation) for more than 10M SEK.

In addition to the personal number and name, the output should contain the total amount the person has loaned (counting both private and organisation loans) and be ordered by this total amount.

Solution:

```
-- Total loans
-- Will not work if TotalLoaned is used in HAVING
SELECT client, name, SUM(amount) AS TotalLoaned
FROM Persons JOIN ActiveLoans ON personal_nr = client
GROUP BY client, name
HAVING SUM(amount) > 10M  -- right figure needs to be given here :)
ORDER BY TotalLoaned;
```

$$\tau_{\mathsf{TotalLoaned}}(\sigma_{\mathsf{TotalLoaned}>10\mathsf{M}} (\gamma_{\mathsf{client,name,SUM(amount)}\rightarrow\mathsf{TotalLoaned}}(\mathsf{Persons} \bowtie_{\mathsf{personal\_nr}=\mathsf{client}} \mathsf{ActiveLoans})))$$

Recall γ combines select and aggregation.

3 Views and Triggers (10 pts)

We continue with the same domain as in question 1 on SQL:

Person (personal_nr, name, tel, email)
Organisation (org_nr, name, address, director, value)
director → Person.personal_nr
Loan (id, client, type, amount, binding, bdate, interest)
client → Person.personal_nr

Propose a solution (meaning, the corresponding full SQL code unless otherwise stated) to the following tasks that need to be performed.

Note: Here you can assume that you have a solution that gives you the active loans even if you have not solved that particular question (in a correct way).

a) (2pts) When an organisation changes the director, active loans from the organisation need now to have the new director (instead of the old one) connected to the loans.

   In this part you DO NOT need to write the full SQL code with the solution, simply explain with words how you would solve the task.

b) (4pts) Produce the billing for the monthly payment of the interest of the (active) loans. The output should contain the id of the loan, the email address of the private person or of the director of the organisation, and the amount to be paid.
   If the loan of an organisation is higher than 10M SEK, then there is a 0.05% discount to apply to the current interest of the loan.
   Recall that the interest in Loans is an **annual** interest and here you need to produce the monthly fee!

   Observe that if a person or an organisation have more than one active loan, there will be an entry in the output for each of those loans.

c) (4pts) The current database design allows you to add organisation-type loans connected to a person (client) who is not a director from any organisation.

Make sure to prevent this problem.

Solution:

For part a) one solution could be to define a trigger BEFORE/AFTER UPDATE ON Organisations that if the NEW.director is not the same as the OLD.director, then it also changes the client of all the active loans from the Organisation to the new director.

Another solution could be to redesign the database and separate the private loans from the loans from an organisation and in this later, have a foreign key from the client of the loan to the director of the organisations. Then we could have an ON UPDATE CASCADE on the directors on the Organisations table.

```
CREATE OR REPLACE VIEW MonthlyBill AS
WITH
  ToBePaid AS (
    (SELECT id, client, type,
            (amount * interest /1200)::NUMERIC(5,2) AS payment
     FROM ActiveLoans
     WHERE type = 'Private' OR amount < 1000)
    UNION
    (SELECT id, client, type,
            (amount * (interest-0.05) /1200)::NUMERIC(5,2) AS payment
     FROM ActiveLoans
     WHERE type = 'Org' AND amount >= 1000))
(SELECT id, email, payment
 FROM Persons JOIN ToBePaid ON personal_nr = client)
ORDER BY id;


-- Checking that Org loans have a client which is a director
CREATE OR REPLACE FUNCTION check_director() RETURNS TRIGGER AS $$
BEGIN
    IF (NEW.type = 'Org') AND
       NOT EXISTS (SELECT director FROM Organisations
                   WHERE director = NEW.client)
    THEN RAISE EXCEPTION 'Director not allowed!';
    END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS NewOrgLoan ON Loans;

CREATE TRIGGER NewOrgLoan
  BEFORE INSERT ON Loans
  FOR EACH ROW
  EXECUTE FUNCTION check_director();
```

4 ER Modelling (11 pts)

a) (6.5pts) You are designing a database for handling tech support issues. Draw an ER-diagram for the database based on this list of requirements:

- Handlers and users are people. Both are registered with a unique email and a (real) name.
  - Only users are allowed to report issues.
  - Only handlers can be assigned to solve issues.
- Issues are in one of three states: Reported, Active, and Resolved. This corresponds to the various steps in handling an issue: first it is reported but not active, eventually it becomes active and finally the issue is resolved (closed).
  - All issues are identified by a unique issue ID. In addition, they have a reporting user, a title, a description and a time of reporting.
  - Active issues are assigned to a primary handler. Sometimes, they are also assigned to a secondary handler.
  - A resolved issue has all the attributes of an active issue, as well as a time of closing.
- Each active (and hence resolved) issue have an event log attached to it. The log consists of a set of log entries with a time and a text.
  - Each log entry is identified by a combination of the ID of the issue it concerns and the time of the log entry.
  - A log entry can "tag" any number of people (these can be users and/or handlers)

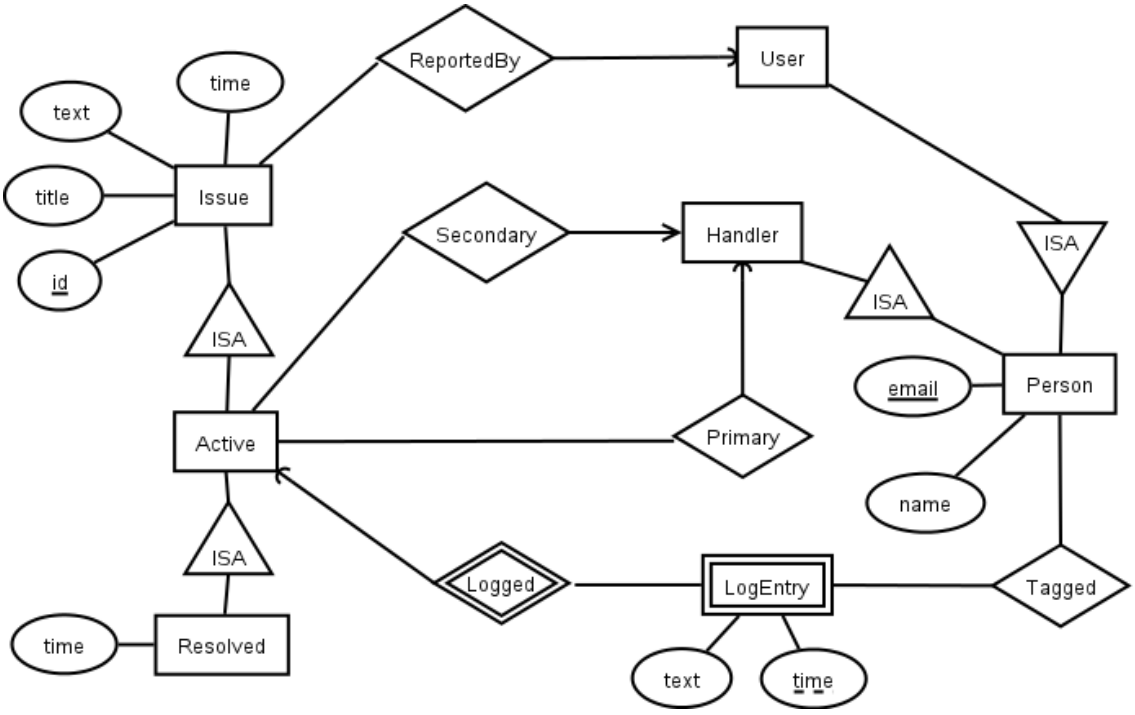
Note: You may assume time values are stored as single attributes.
Hint: Look at the example below if anything is unclear about the domain, or ask the teacher!

b) (4.5pts) Translate your diagram into a relational schema, and for each relation list the content of the relation that is needed to encode the example below. To keep the amount of writing needed manageable, you can abbreviate sentences to just the first word and times to just the time of day (e.g. (Failed, 9:00)). Also you can abbreviate Alice/Bob/Charlie to A/B/C, and use a@/b@/c@ for their emails.

Example: (A resolved issue with a single handler and four logged events)
ID: 123
Title: Computer says no
Description: The thing that should work doesn't work.
Reported at: 2023-08-01 16:30
Resolved at: 2023-08-04 15:01
Reported by: Alice `<alice@example.com>`

Primary Handler: Bob <bob@example.com>
Secondary Handler: -
Event log:
2023-08-02 8:25 - We are looking into the issue.
2023-08-02 9:00 - Failed to replicate, awaiting additional information.
Tagged: Alice <alice@example.com>, Charlie <charlie@example.com>
2023-08-04 14:00 - Received additional information.
2023-08-04 15:00 - Issue resolved, working as intended.

Solution:



It's fine to have the time an issue was reported as an attribute of the relationship ReportedBy.

The content of each relation is listed directly underneath them.
Note: Making Charlie a user or a handler is fine, but not required.
Note: The Secondary relation is empty in this particular example.

Persons (name, email)
(A, a@)
(B, b@)
(C, c@)

Users (email)
email → Persons.email
(a@)

Handlers (email)
email → Persons.email
(b@)

Issues (id, title, text, time, reporter)
reporter → Users.email
(123, Computer..., The..., 16:30, a@)

Active (id, primary)
id → Issue.id
primary → Handlers.email
(123, b@)

Resolved (id, time)
id → Active.id
(123, 15:01)

Secondaries (id, handler)
id → Active.id
handler → Handler.email

Logs (id, time, text)
id → Active.id
(123, 8:25, We...)
(123, 9:00, Failed...)
(123, 14:00, Received...)
(123, 15:00, Issue...)

Tagged (id, time, person)
(id, time) → Logs.(id, time)
person → Persons.email
(123, 8:25, a@)
(123, 8:25, c@)

5 Functional and Multivalued Dependencies (8pts)

You are designing a database for an online chat system. The entire domain has 11 attributes, presented here in alphabetical order:

R(attachment, blocked_user, channel_id, channel_name, message_id, reacting_user,

reaction, text, time, timeout, user)

The attributes are to be understood as follows:

attachment: This is the URL (a unique identifier) of a file attached to a message.
blocked_user: This is the username of a user that has been temporarily blocked by another user.
channel_id: This is the globally unique ID number of a channel, in which messages can be posted.
channel_name: This is the name of a channel.
message_id: This is an id number of a message.
reacting_user: The username of a user who has reacted to a message.
reaction: The emoji some reacting_user has used to react to a message.
text: This is the content of a message.
time: This is the time at which a message was posted.
timeout: This is the time at which a temporary block expires.
user: The username of a user that has posted messages, blocked other users etc.

Some important notes (all of these need to be taken into account for a correct solution):

- Each message is posted in a specific channel.
- Message ID's are only unique for their specific users, multiple posts can have the same ID if they are all from different users.
- A message can have any number of attached files.
- The same file can be attached to multiple messages, even by different users.
- A message can receive any number of reactions, including multiple different reactions from the same user.
- There cannot be two channels with the same name.
- A user can block any number of other users but can't block the same user multiple times

a) (5.5pts) Identify some FDs and normalize R into BCNF. In each step, clarify which relation is being decomposed and which violating FDs you are using. Mark primary keys in the relations that are part of the final schema.

b) (2.5pts) Identify some MVDs and further normalize your result from a) to 4NF. You do not need to repeat the schema from a). In each step, make it clear which relation (from your BCNF schema) is being decomposed and what violating MVD you are using.

Solution:

a) user message_id → channel_id channel_name time text
channel_id → channel_name
channel_name → channel_id
user blocked_user → timeout

We decompose R with the FD user message_id → channel_id channel_name time text
R1(user, message_id, channel_id, channel_name, time, text)
R2(user, message_id, reaction, attachment, reacting_user, blocked_user, timeout)

R1 can be further decomposed with the FD channel_id → channel_name (and channel_name → channel_id)
R11(user, message_id, channel_id, time, text)
R12(channel_id, channel_name)

R2 can be further decomposed with the FD user blocked_user → timeout
R21(user, blocked_user, timeout)
R22(user, blocked_user, message_id, reaction, attachment, reacting_user)

We set now the PKs and secondary keys:
R11(<u>user</u>, <u>message_id</u>, channel_id, time, text)
channel_id → R12.channel_id
R12(<u>channel_id</u>, channel_name)
R21(<u>user</u>, <u>blocked_user</u>, timeout)
R22(<u>user</u>, <u>blocked_user</u>, <u>message_id</u>, <u>reaction</u>, <u>attachment</u>, <u>reacting_user</u>)
(user, message_id) → R11.(user, message_id)
(user, blocked_user) → R21.(user, blocked_user)

Note: Using channel_name in R11 instead of channel_id is OK. Reversing the order of the steps has the same result and is a bit less error prone.

b) user message_id ↠ attachment
user message_id ↠ reaction, reacting_user

Here is one possible decomposition (a different order can give another correct solution):

We decompose R22 with MVD: user, message_id ↠ attachment
R221(<u>user</u>, <u>message_id</u>, <u>attachment</u>)
R222(<u>user</u>, <u>message_id</u>, <u>reaction</u>, <u>reacting_user</u>)

6 JSON (9 pts)

Consider the domain of tech support issues of question 4. Below is a JSON document encoding a set of two issues (some properties are left out for brevity):

```
[{"id": 123,
  "status": "resolved",
  "reportedBy": {"name": "Alice","email": "alice@example.com"},
  "handlers": [ {"name": "Bob", "email": "bob@example.com"} ],
  "log": [
    { "text": "We are looking into the issue."},
```

```
    { "text": "Failed to replicate, awaiting additional information.",
      "mentions": [
        {"name": "Alice", "email": "alice@example.com"},
        {"name": "Charlie", "email": "charlie@example.com"}
      ]
    }]
},
{ "id": 124,
  "status":"reported",
  "reportedBy": {"name": "Charlie","email": "charlie@example.com"}
}]
```

a) (4pts) Make a JSON schema for documents like this one. The schema should ensure the following:

- All types are correct (changing any string into an integer or vice versa should invalidate the document).
- The status property is one of "reported", "active" or "resolved".
- All properties are required unless there is a reason for them not to be (looking at the example given).
- There are at most two handlers for an issue.

Hint: Use a definition and $ref for the frequently recurring person-objects.
Hint: minItems and maxItems are two keywords in JSON schemas.

b) (3pts) Briefly describe how you would modify the schema to ensures the following:

- If the status of an issue is "reported", it has no "handlers" property.
- If the status of an issue is "active" or "resolved", it has at least one handler.
- Only issues with the status "resolved" have a "resolvedAt" property, and all resolved issues have that property.

Your description needs to be clear enough to demonstrate that you know how to perform this modification (mention which keywords you use for what purpose etc.), but it does not need to be a complete schema.

c) (2pts) Write a JSON Path expression for finding the id numbers of all resolved issues with Bob as the primary handler.

Solution:

a) *Note:* Specifying type for status is fine but not necessary.
Note: If you used oneOf instead of enum, we'll accept it (but it wouldn't work).

```
{ "definitions":
    {"person":
      {"type": "object",
       "properties":
         {"name": {"type": "string"},
          "email": {"type": "string"}},
       "required": ["name","email"]}
     },
  "type": "array",
  "items":
    { "type": "object",
      "properties":
       {"id": {"type": "integer"},
        "status": {"enum" : ["reported","active","resolved"]},
        "reportedBy": {"$ref":"#/definitions/person"},
        "handlers":
          {"type": "array",
           "items": {"$ref":"#/definitions/person"},
           "maxItems":2},
        "log":
          {"type": "array",
           "items":
            {"type": "object",
             "properties":
              {"text": {"type": "string"},
               "mentions":
                  {"type": "array",
                  "items": {"$ref":"#/definitions/person"}}
              },
              "required": ["text"]
              } }
      },
      "required": ["id","status","reportedBy"]
    }
}
```

b) The most fundamental thing is using `oneOf` to write separate schemas for reported, active and resolved issues.

Here is a non-exhaustive list of additional modifications that give points:

- Reported issues have `"status": {"enum" : ["reported"]}`, and similarly for the other statuses.
- Reported issues have `"handlers":false` in its properties.
- Active and resolved issues require the handlers property.
- Add `"minItems":1` to the handlers property (to ensure there is at least one handler).

- Resolved issues require resolvedAt, and other issues have `"resolvedAt":false`.

c) `$[*]?(@.status=="resolved" && @.handlers[0].name=="Bob").id`
or equivalently

```
$[?(@.status=="resolved" && @.handlers[0].name=="Bob")].id
```