Databases Exam

TDA357 (Chalmers), DIT621 (University of Gothenburg)

2020-08-27 14:00-18:00

Department of Computer Science and Engineering

Examiner: Jonas Duregård, contact via Zoom supervision. In case of urgent technical issues, contact by phone: 031 772 1028

Results: Will be published within three weeks from exam date

Maximum points: 60

Grade limits Chalmers: 24 for 3, 36 for 4, 48 for 5.
Grade limits GU: 24 for G, 42 for VG.

Instructions and submission: https://chalmers.instructure.com/courses/12048

If technical issues prevent you from accessing Canvas, send your solution to jonas.duregard@chalmers.se no later than 18:30 using this exact title:
TDA357 DIT621 exam submission

Question 1: ER-design (11 points)

a) Make an ER-diagram for the following domain:

A company is making an online fantasy role playing game. Part of the game includes collecting items (like weapons and armor). To easily create lots of unique-sounding names with minimal effort, the company decided that all items will be named on the form "<prefix> <base item> of <suffix>". For example "shining sword of the badger" (with the prefix "shining" base item "sword" and suffix "the badger") or "cursed dagger of SQL-injection" etc.

Each prefix or suffix applies a number of magical effects to the wearer. These effects all modify the attributes of the player wearing them (such as strength or intelligence). Example of an effect could be "shining" giving +3 charisma. The effects can also be attribute multipliers, e.g. "cursed" could give -10% vitality and -5 strength, and a prefix can have both kinds of modifiers for the same attribute. Prefixes can have any number of effects, suffixes can have at most one.

The company has these specific requirements in addition to the description above:

- There needs to be a table called Items where each row represents an item added to the game.
- There should be a fixed set of player attributes available, you cannot invent a new attribute by just adding an item/prefix/suffix/effect. There should also be a fixed set of base items.
- It should be possible to compute the effect of an item on a player's attributes by querying the database and doing some simple math on the result.
- There cannot be multiple items with the same name.
- Each prefix/suffix always has the same effects on attributes, regardless of which item it is applied to.

It may be difficult to enforce all these requirements in an ER-design. If so, do as close approximation as you can, and clearly specify what parts are not possible.

Note: you only have to do the item database - not players, inventories etc.!

b) Translate your diagram from a) into a relational schema. You can get full points for this even if your solution from a) is not correct, but if your solution to a) is extremely over-simplified you will not get full points even if translating it correctly.

If your ER-design had any known flaws, can the schema be patched to fix them? Indicate clearly if you make any such changes.

c) Show the contents of the relations/tables in b) needed for an item called "wonderful armor of the rainbow", with "wonderful" giving +3 wisdom and an additional +3% wisdom, and "the rainbow" giving +10 charisma.

Question 2: Functional Dependencies, Normal Forms (8 points)

Consider a relation R(A, B, C, D) with the following Functional Dependencies

A -> B
B -> A
C -> D

a) Construct a table with exactly these functional dependencies (including any derived ones) and no other functional dependencies. You can use small integers as the data, e.g. your solution could look like this:

```
A B C D
1 1 2 2
1 1 2 3
```

This solution would however be incorrect for multiple reasons: It has lots of additional dependencies including A->C (but not A->D), and it violates C->D.

Hint: It is possible to construct a correct solution with four rows (but it is OK to use more).

b) List 2 different minimal keys (key candidates) of this relation (each key is a set of attributes, write each set in alphabetical order on its own line)

Question 3: SQL Queries (10 points)

Below is the schema for a database of assignment submissions, somewhat similar to submissions done in Canvas for this exam and for lab assignments.

```
Assignment(course, name, description, deadline)

Submission(idnr, student, course, assignment, stime)
  (course,assignment) -> Assignment.(course, name)

SubmittedFile(submission, filename, filesize, contents)
  submission -> Submission.idnr
```

Assignments have names that are unique within each course, a text description and a deadline (specified as a timestamp).

For each assignment there is a number of submissions, each done by a student (the same student may make multiple submission for each assignment). Each submission is given a unique id number (idnr). The attribute stime is the time of submission as a timestamp.

For each submission there is a number of submitted files, each has a name, a size and a file content.

Example contents of Assignment, Submission and SubmittedFile (some data left out, and timestamps are symbolic and do not correspond to realistic dates/times):

course	name	description	deadline
TDA357	Lab 1	...	2500
TDA357	Lab 2	...	6000
TDA144	Lab 2	...	2000

idnr	student	course	Assignment	stime
0	s1	TDA357	Lab 1	1000
1	s2	TDA357	Lab 2	2000
2	s1	TDA357	Lab 1	3000
3	s3	TDA144	Lab 2	1000

submission	filename	filesize	contents
1	tables.sql	1300	...
1	comment.txt	100	...
2	views.sql	800	...
2	tables.sql	900	...

a) Compute the combined file size of all submissions for each student that has made at least one submission. The result for the data above should be (student, total size):
(s1, 1400) (s2, 900) (s3, 0)

b) Find all submissions to Lab 1 of course TDA357 that are missing one or both files 'tables.sql' and 'views.sql'.

Question 4: Relational Algebra (9 points)

Write a relational algebra expression for solving these tasks:

a) Find all submissions done after the deadline had passed for its assignment.

b) If everything works as intended, a submission with higher idnr should also have later (or equal) stime. Find all pairs of assignment (their idnr) that violate this property. For the data above, the pairs (1,3) and (2,3) violate this property.

c) Construct a copy of Submissions, but containing only the latest submission for students that have made multiple submissions for the same assignment. You do not need to include stime in the result. You may assume the latest submission will have the highest idnr. For the data above, it should be identical to submission except the submission with idnr=0 is missing (and there is no stime column).

Hints: Do not use relational algebra expressions in conditions!

Question 5: Views, constraints and triggers (12 points)

Database integrity can be improved by several techniques:

- Views: virtual tables that show useful information that would create redundancy if stored in the actual tables
- SQL Constraints: conditions on attribute values and tuples
- Triggers (and assertions): automated checks and actions performed on entire tables

As a general rule, these methods should be applied in the above order: if a view or constraint can adequately do the job, do not use a trigger.

The task in this question is to implement a small database. You may use any SQL features we have covered in the course. While the description below gives requirements for what should be in the database, you are allowed to divide it across as many tables and views as you need to. Points will be deducted if your solution uses a trigger where a constraint or view would suffice, or if your solution is drastically over-complicated. For triggers, it is enough to specify which actions and tables it applies to, and PL/(pg)SQL pseudo-code of the function it executes.

The database will be mostly the same as in Question 3 and 4:

```
Assignment(course, name, description, deadline)

Submission(idnr, student, course, assignment, stime)
  (course,assignment) -> Assignment.(course, name)

SubmittedFile(submission, filename, filesize, contents)
  submission -> Submission.idnr
```

Additionally, you should add a table Registered that keeps track of which courses each student is registered for:

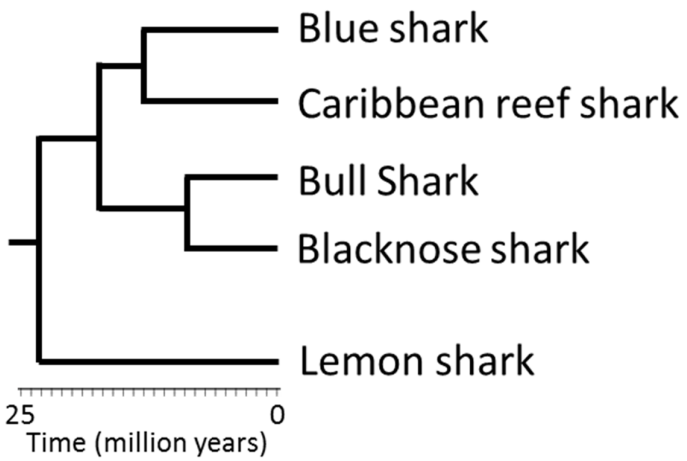
```
Registered(student, course)
```

You should also enforce the following additional constraints (write a letter in the margin of your code where you implement each constraint, the same letter may appear multiple times):

a) The newer a submission is the higher its idnr.
b) Students can only submit solutions to assignments in courses they are registered for.
c) filesize always reflects the length of content (computable by length(content) in Postgres). A database user should not need to specify filesize when adding a new file to a submission.
d) When a submission is deleted, its files should also be deleted automatically.
e) When all files of a submission are deleted, delete the submission itself as well.
f) A student can not have two submissions for the same assignment with exactly the same time.

Question 6: Semi-structured data and other topics (10 p)

Below is a kind of phylogenetic tree (specifically a chronogram), a "tree of life" that shows the evolution of animals into various species. The branching points (splits) in the tree show when two species diverged from a common ancestor.



Your job is to represent the data needed to construct such a diagram in a suitable JSON format. A few things to note:

- The lengths of the horizontal bars (or equivalently: the horizontal positions of the splits) are relevant. In this example Blue sharks and Caribbean reef sharks diverged earlier than Bull sharks and Blacknose sharks did.
- These particular diagrams only show now living species by name, so the bars leading up to a name always extend all the way to 0.
- The vertical size and vertical position of all bars and names are automatically decided and do not need to be stored in the data set you are creating.
- The time scale shown at the bottom should be possible to automatically decide based on the data. It should not need to be specified explicitly in the JSON document.

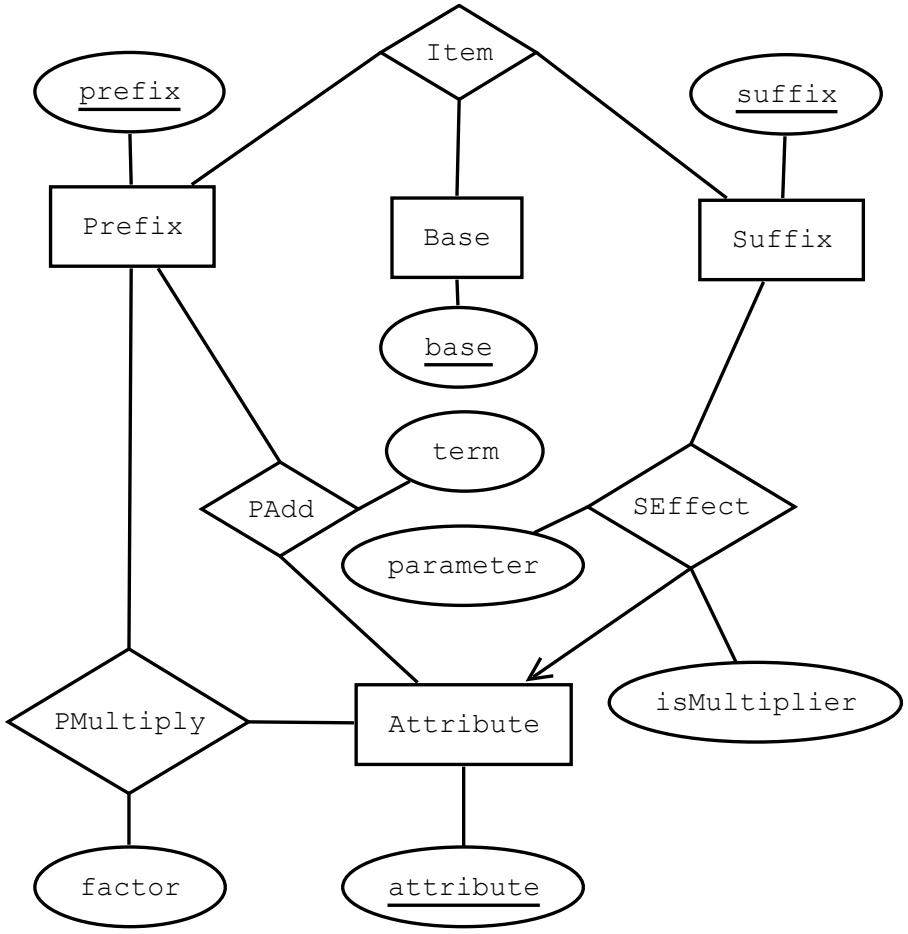
Encode the data for the image above as a JSON object. You do not have to write a JSON schema, but the format of your data should be clear and consistent so one can easily see how to create a JSON document for a different diagram.

Also write a short text describing the choices you made for your format and any assumptions you make about limitations etc.

For the purpose of creating the JSON document, let's say that the splits in the diagram are at 24, 20, 17 and 10 million years ago.

Question 1:

a) One possible solution, perhaps the simplest one that satisfies all criteria:



I use “ismultiplier” as to indicate if parameter is a factor or a term. It could be split up, but then suffixes could have more than one effect.

You could have an Effect entity with multiplier and/or addition as subentities.

b)

Prefix(prefix)
Base(base)
Suffix(suffix)
Attribute(attribute)

PMultiply(prefix, attribute, factor)
  prefix -> Prefix.prefix
  attribute -> Attribute.attribute

PAdd(prefix, attribute, term)
  prefix -> Prefix.prefix
  attribute -> Attribute.attribute

SEffect(suffix, attribute, isMultiplier, parameter)
  suffix -> Suffix.suffix
  attribute -> Attribute.attribute

c)

Prefix:
(Wonderful)

Base:
(Armor)

Suffix:
(The Rainbow)

Attribute:
(Wisdom)
(Charisma)

PMultiply:
(Wonderful, Wisdom, 1.03) – or 0.03 or 3 or really any value that makes sense

PAdd:
(Wonderful, Wisdom, 3)

SEffect:
(The Rainbow, Charisma, FALSE, 10)

Question 2:

a)

A	B	C	D
1	1	2	2
1	1	3	3
4	4	2	2
5	5	5	2

In the table above I mostly use new numbers except when there's a specific reason not to. A and B doesn't always have to be identicalLi though (e.g. I could have used (1,2), (2,3) and (3,4) instead of (1,1), (4,4) and (5,5)).

There are *a lot* of other correct solutions (this one will be fun grading…). The first two lines in this solution makes sure that there are no dependencies from A and/or B to C or D. The third line does the same in the other direction and the last just makes sure that D -> C doesn't hold.

b)

a c
b c

Question 3:

```sql
-- a)

SELECT student, SUM(COALESCE(filesize,0))
FROM Submission LEFT OUTER JOIN SubmittedFile ON submission=idnr
GROUP BY student;

-- b)
-- There are lots of ways to solve this, here I use a correlated query
--   to count the number of files named tables.sql and views.sql in each
--   submission (should be two only if both files are present since
--   duplicate filenames cannot exist within submissions)
-- There are also lots of incorrect ways of solving this, like most things
--   you can do with joins
SELECT *
FROM Submission AS Sub
WHERE
  course = 'TDA357' AND assignment='Lab 1'
  AND  2 > (SELECT COUNT(*) FROM SubmittedFile
            WHERE submission = Sub.idnr
                  AND (filename = 'tables.sql' OR filename = 'views.sql'))
  ;
```

Test data for your own solution:

```sql
CREATE TABLE Assignment
    ( course TEXT
    , name TEXT
    , description TEXT
    , deadline INT
    , PRIMARY KEY (course, name)
    );

CREATE TABLE Submission
    ( idnr INT PRIMARY KEY
    , student TEXT
    , course TEXT
    , assignment TEXT
    , stime INT
    , FOREIGN KEY (course, assignment) REFERENCES Assignment(course, name)
    );

CREATE TABLE SubmittedFile
    ( submission INT
    , filename TEXT
    , filesize INT
    , contents TEXT
    , FOREIGN KEY (submission) REFERENCES Submission
    , PRIMARY KEY (submission, filename)
    );

INSERT INTO Assignment VALUES ('TDA357', 'Lab 1', '...', 2500);
INSERT INTO Assignment VALUES ('TDA357', 'Lab 2', '...', 6000);
INSERT INTO Assignment VALUES ('TDA144', 'Lab 2', '...', 2000);

INSERT INTO Submission VALUES (0, 's1', 'TDA357', 'Lab 1', 1000);
INSERT INTO Submission VALUES (1, 's2', 'TDA357', 'Lab 2', 2000);
INSERT INTO Submission VALUES (2, 's1', 'TDA357', 'Lab 1', 3000);
INSERT INTO Submission VALUES (3, 's3', 'TDA144', 'Lab 2', 1000);

INSERT INTO SubmittedFile VALUES (1, 'tables.sql',  1300, '...');
INSERT INTO SubmittedFile VALUES (1, 'comment.txt', 100,  '...');
INSERT INTO SubmittedFile VALUES (2, 'views.sql',   800,  '...');
INSERT INTO SubmittedFile VALUES (2, 'tables.sql',  900,  '...');
```

Question 4:

a)

$$\sigma_{\text{stime} > \text{deadline AND Submissions.course} = \text{Assignment.course AND assignment} = \text{name}} (\text{Submissions} \times \text{Assignment})$$

b)

$$\sigma_{\text{A1.idnr} > \text{A2.idnr AND A1.stime} < \text{A2.stime}} (\rho_{A1}(\text{Assignment}) \times \rho_{A2}(\text{Assignment}))$$

c)

$$\gamma_{\text{student, course, assignment, MAX(idnr) -> idnr}}(\text{Submissions})$$

Question 5:

```
CREATE TABLE Registration
  ( student TEXT
  , course TEXT
  , PRIMARY KEY (student, course)
  );

CREATE TABLE Assignment
    ( course TEXT
    , name TEXT
    , description TEXT
    , deadline INT -- Should probably be a TIMESTAMP or similar
    , PRIMARY KEY (course, name)
    );

CREATE TABLE Submission
    ( idnr INT PRIMARY KEY -- Should probably be a SERIAL
    , student TEXT
    , course TEXT
    , assignment TEXT
    , stime INT
    , FOREIGN KEY (course, assignment) REFERENCES Assignment(course, name)
    , FOREIGN KEY (course, student) REFERENCES Registration(course,
student) -- b)
    , UNIQUE (student, stime) -- f)
    );

CREATE TABLE SubmittedFileTable
    ( submission INT
    , filename TEXT
    , contents TEXT
    , FOREIGN KEY (submission) REFERENCES Submission ON DELETE CASCADE-- d)
    , PRIMARY KEY (submission, filename)
    );


-- c) Using a view, an alternative is to use DEFAULT + a check constraint
or something like that.
CREATE VIEW SubmittedFile AS
  SELECT *, length(contents) AS filesize FROM SubmittedFileTable;

-- a) needs a trigger on INSERT OR UPDATE on Submission that either
--    automatically adjusts the idnr or rejects updates that violate the
--    rule. Alternatively, a) can be solved using an assertion with a
--    NOT EXISTS query corresponding to the RA expression in 4b
-- e) needs to be done using a trigger AFTER DELETE for
--    SubmittedFile(Table) that checks if the last file was deleted and if
--    so run DELETE FROM Submission WHERE idnr = OLD.submission
--    to be completely safe it probably needs to be AFTER DELETE OR UPDATE
```

Question 6:

```
{
    "splitAt": 24,
    "branch1": {
        "splitAt": 20,
        "branch1": {
            "splitAt": 17,
            "branch1": "Blue shark",
            "branch2": "Caribbean reef shark"
        },
        "branch2": {
            "splitAt": 10,
            "branch1": "Bull shark",
            "branch2": "Blacknose shark"
        }
    },
    "branch2": "Lemon shark"
}
```

This solution only includes positions for splits, thus assuming the length of the bar for the root of the tree does not need to be specified. Every branch is either just a String or or another split. This one requires splitAt to be decreasing further down the tree.