UNIVERSITY OF GOTHENBURG

CHALMERS UNIVERSITY OF TECHNOLOGY

Exam

DIT032 / DAT335 – Data Management
Take-Home Edition

Tuesday, June 9th, 2020 08:30 - 12:30 (+ 30 minute grace period)

Examiner:
Philipp Leitner

Contact Persons During Exam:
Philipp Leitner (+46 733 05 69 14, philipp.leitner@chalmers.se)

Allowed Aides:
You may refer to all class material during the exam.
This is still an individual exam, communicating with people other than teachers during the exam period – independently of whether that's in-person or via Slack / Facebook / SMS / etc. – is strictly forbidden.

Results:
Exam results will be made available no later than in 15 working days through Ladok.

Grade Limits:
For GU students: 0 – 49 pts: U, 50 – 84 pts: G, 85+ pts: VG
For Chalmers students: 0 – 49 pts: U, 50 – 69 pts: 3, 70 – 84 pts: 4, 85+ pts: 5

Review:
Exam reviews will be handled on demand via Zoom.

Task 1 – Theory and Understanding (20 pts)

Each of the following questions requires an approximately two to four paragraphs long answer **in your own words**. Use figures or sketches if appropriate. Read the questions carefully, and make sure to answer the question that is asked. All questions ask for **examples**: these are important, and answers without a good, self-developed example will not lead to many points. Any existing examples from the lecture slides, book, or the Internet do not count here - you should develop the examples yourself.

Q1.1: Describe the basic database operations introduced in this course (the CRUD operations). Specify an example database table and provide concrete examples in SQL for all four operations. *(5 pts)*

Q1.2: Describe the process of query rewriting. Use a concrete example and illustrate the query plan in different phases of the optimization process. Also explain how we can display the final query plan in PostgreSQL. *Note: your example does not need to be optimal, i.e., don't worry if your example optimizations wouldn't really be how a database would really do the optimizations. However, the query plans should indeed be equivalent in all phases. (5 pts)*

Q1.3: Describe and compare replication and sharding. In what ways are they similar, and how do they differ? Provide an example of a scenario in which both sharding and replication would be useful. *(5 pts)*

Q1.4: In what cases are transactions in a relational database useful, and what problems do they prevent? Provide an example of an interaction with a database that makes use of transactions, and explain what problems are prevented by using transactions in your example. Also note which database isolation level your example assumes. *Note: your example does not need to contain the actual SQL statements, a low-level textual description of what happens is sufficient. For example: "User 1 opens a transaction and queries item A. User 2 opens a transaction ... User 2 commits. This prevents the following problem: ... (5 pts)*

Task 2 – EER Diagrams (24 pts)

Consider the following excerpt of the domain description for a database used in a research project (such as the bachelor thesis you will be writing in a few semesters). The goal of the database is to store meta-information about Java programs collected from a repository service such as GitHub for future analysis.

Model the described domain using an EER diagram with the notation we used in the course (find a cheat sheet in the appendix of your exam papers). Use the 1,N,M notation for describing cardinalities rather than the min-max notation. If *and only if* something is not specified, make a reasonable assumption and note it down in plain text.

Java Programs Database:
The database needs to store various Java programs. For each program, we need to store a link to the original repository (which is unique and identifies the program), as well as a human-readable name and the Java dialect that is being used. Each program contains zero to many classes, and each class contains zero to many methods. Classes are identified through fully qualified class names (that are only unique for a program), and methods are identified in the context of a specific class through a combination of name, return value, and a list of parameters. For methods, we also need to store the implementation code.

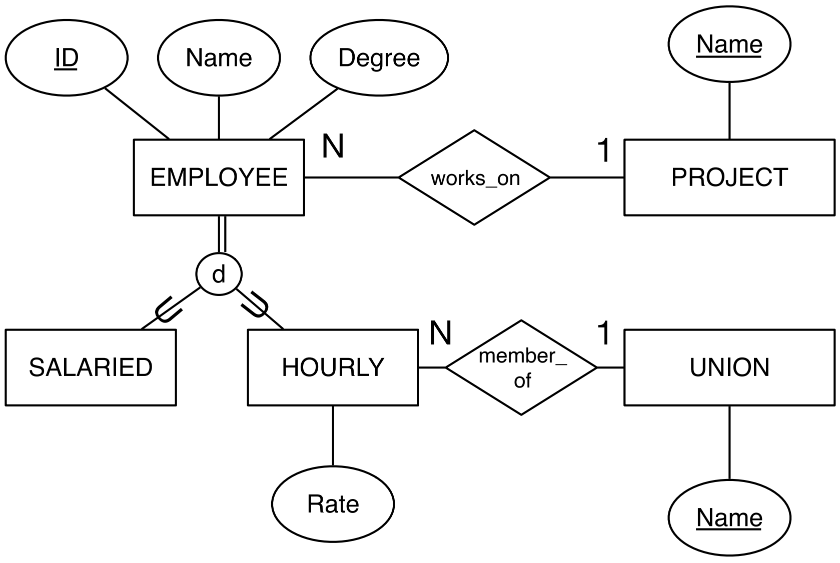
Classes are stored in files. Every class is in exactly one file, and a file may not contain more than one class. Files are identified through a combination of path, filename, and file extension.

In addition to programs, our database also needs to store developers. Developers have unique usernames and unique emails, and we also need to save when the developer has last logged in. Developers write arbitrarily many methods, and each method may have been authored by many developers.

There is a special type of developer, which we call committers. Committers have "commit rights" in one or more programs (i.e., they can confirm changes to the program). For each combination of committer and program, we need to also save what role a committer has in the program. Further, we need a way to derive how many committers every program has. Committers belong to organizations, which are identified through unique names and also store an address. Every committer belongs to exactly one organization, but an organization can have multiple committers. Lastly, every program has exactly one owner. An owner is a special committer. Every owner owns at least one program, but they can also own multiple programs.

Task 3 – Mapping Inheritance (9 pts)

Consider the EER snippet below. Apply *all three different* strategies for mapping inheritance we learned in the course. Report the relational model that results from applying all three strategies to this domain, and discuss concrete advantages and disadvantages of each mapping strategy.



Task 4 – Relational Algebra (20 pts)

Relational Model:
COURSE(shortname, fullname, syllabus)

TEACHER(id, name, salary, course)
course $\rightarrow$ COURSE.shortname

COURSE_INSTANCE(course, year, start, end)
course $\rightarrow$ COURSE.shortname

STUDENT(id, name, semester)

STUDENT_COURSEINSTANCE(course, student, year, grade)
{course,year} $\rightarrow$ {COURSE_INSTANCE.course, COURSE_INSTANCE.year}
student $\rightarrow$ STUDENT.id

(course is a foreign key pointing at COURSE.shortname, student is a foreign key pointing at STUDENT.id, and the combination of course and year in STUDENT_COURSEINSTANCE is a foreign key pointing at COURSE_INSTANCE)

Given this relational model, write relational algebra statements that exactly represent the following queries. Use the mathematical notation from the course (for the correct notation you can again refer to the appendix).

Queries:

Q4.1: Return the course short names and durations (defined as the difference between end and start date) of all course instances running in 2019.

Q4.2: Return the names of all students registered to the course with the fullname "Data Management".

Q4.3: Find the highest salary of a teacher not teaching any courses.

Q4.4: Return a list of courses and the years they have been given in. Each tuple in your result should contain shortname, fullname, syllabus, and the year. If a course has never been given it should still be contained in your result (with a NULL value for the year).

Task 5 – SQL (20 pts)

Given the same relational model as for Task 4, write the following SQL queries.

Queries:

Q5.1: Create the table `STUDENT_COURSEINSTANCE`. Assume that the `STUDENT` and `COURSE_INSTANCE` tables already exist. Select suitable data types. Grades should be integer values smaller than 6. If a student is deleted, all their records in `STUDENT_COURSEINSTANCE` should automatically also be deleted.

Q5.2: Return the short and full names of all courses where the syllabus contains the word "database".

Q5.3: Return the names and salaries of all teachers of the student with the name "Anna Berg". Avoid returning duplicate teachers, and order results from high to low salaries.

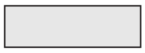
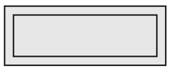
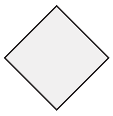
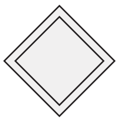
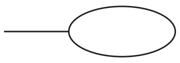
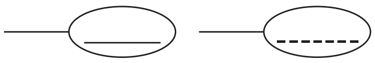
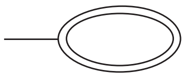
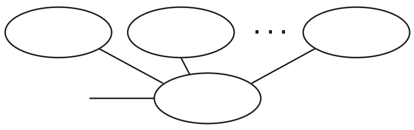
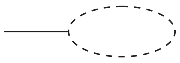
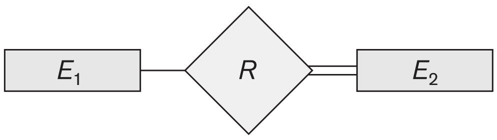
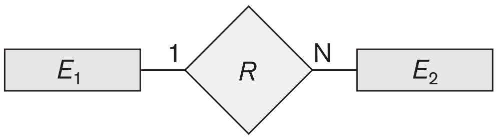
Q5.4: Return the names of all teachers whose salary is above the average teacher salary.

Task 6 – XML and JSON (7 pts)

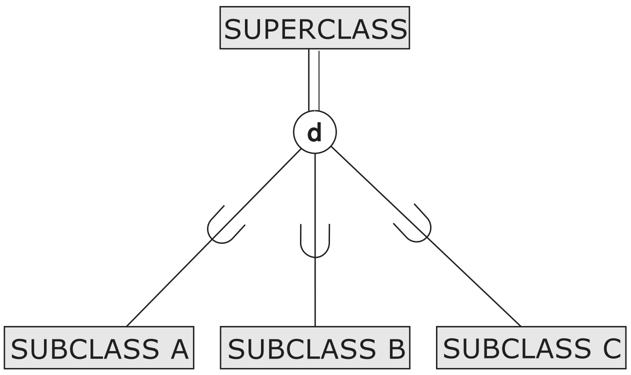
Consider the JSON document below. Provide a well-formed and logically structured XML file that captures the same information. Make use of all of the following: attributes, subelements, and text nodes.

```
{
  "id" : 146,
  "name" : "Philipp Leitner",
  "departments" : ["CSE"],
  "projects" : [
    {"id" : 14, "name" : "TRAF-C"},
    {"id" : 21, "name" : "Immersed"}
  ],
  "courses" : ["DIT032", "DAT335", "DIT341"],
  "started_at" : "01.08.2017"
}
```

Appendix: Notation Guidelines for EER and RA

Symbol	**Meaning**
	Entity
	Weak Entity
	Relationship
	Indentifying Relationship
	Attribute
	Key Attribute / Dashed Underline for Partial Key
	Multivalued Attribute
	Composite Attribute
	Derived Attribute
	Total Participation of E_2 in R
	Cardinality Ratio 1: N for E_1 : E_2 in R

Total Disjoint Specialization



Partial Overlapping Specialization

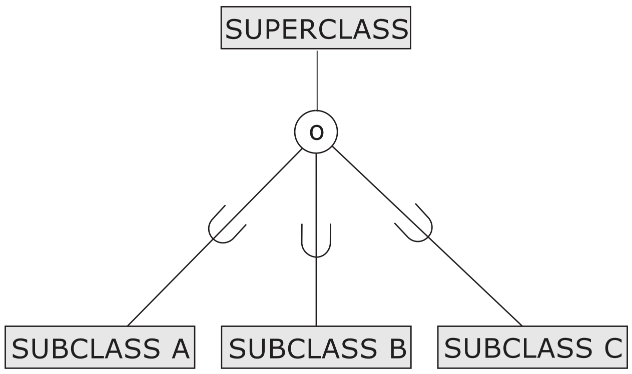


Table 8.1 Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\text{<selection condition>}}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\text{<attribute list>}}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\text{<join condition>}} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\text{<join condition>}} R_2$, OR $R_1 \bowtie_{\text{(<join attributes 1>), (<join attributes 2>)}} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{\text{<join condition>}} R_2$, OR $R_1 *_{\text{(<join attributes 1>), (<join attributes 2>)}} R_2$ OR $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

`<grouping>`$\mathscr{F}$`<functions>(R)`

whereas `<functions>` is a list of

```
[MIN|MAX|AVERAGE|SUM|COUNT] <attribute>
```