UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Exam

DIT033 / DAT335 – Data Management

Thursday, June 8th, 2023 08:30 - 12:30

Examiner:
Philipp Leitner

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

Allowed Aides:
None except English dictionary (non-electronic), pen/pencil, ruler, and eraser.

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

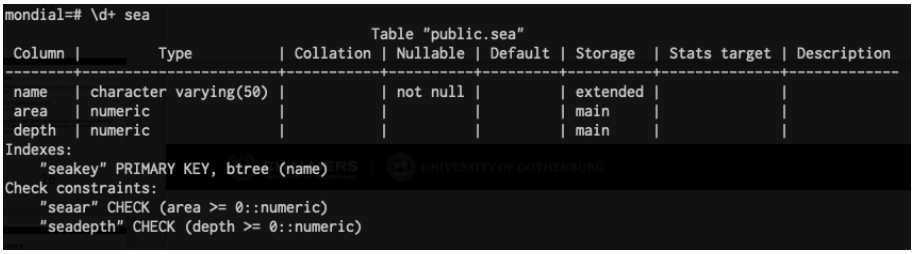
Grade Limits:
For GU students: 0 – 49 pts: U, 50 – 84 pts: G, 85+ pts: VG
For Chalmers students: 0 – 49 pts: U, 50 – 69 pts: 3, 70 – 84 pts: 4, 85+ pts: 5

Review:
Exam reviews will be announced via Canvas.

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**. Many of these questions ask for examples – these examples are important, and you should develop your own examples (simply re-using an existing example from the lecture slides or the Internet is not sufficient). Use figures or sketches if appropriate. Read the questions carefully, and make sure to answer the question that is asked.

Q1.1: Describe what a full table scan is and why it can be problematic. Use the example database table shown below (the sea table in Mondial) as basis for your discussion, and provide one example query on this table that would lead to a full table scan. Also describe what, if any, index would help with your example query. *(6 pts)*



```
mondial=# \d+ sea
                                    Table "public.sea"
 Column |         Type          | Collation | Nullable | Default | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+---------+----------+--------------+-------------
 name   | character varying(50) |           | not null |         | extended |              |
 area   | numeric               |           |          |         | main     |              |
 depth  | numeric               |           |          |         | main     |              |
Indexes:
    "seakey" PRIMARY KEY, btree (name)
Check constraints:
    "seaar" CHECK (area >= 0::numeric)
    "seadepth" CHECK (depth >= 0::numeric)
```

Q1.2: Refer to the below (buggy) SQL query. Why will this query not work, and how could it be fixed? Assume that the used tables and attributes are all available, i.e., the problem is *not* a misspelled or incorrect table or attribute name. *(4 pts)*

```
SELECT AVG(depth), name
FROM lake
GROUP BY type;
```

Q1.3: Assume a database isolaton level of `READ UNCOMMITTED`. Provide and discuss an example database interaction that suffers from a dirty read. Develop your own example – existing examples from the lecture slides, book, or the Internet do not count. *(4 pts)*

Q1.4: Describe and contrast the two ways we discussed to distribute a database (replication and sharding). What does each of these distribution strategies help with? *(6 pts)*

Task 2 – EER Diagrams (24 pts)

Consider the following excerpt of the domain description for the database of a streaming service. 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.

Streaming Service Database:
At the heart of the system are users. For users we need to save an unique username and their watching preferences. There are two types of users. Free users have a quota (their account is disabled if they watch more than their quota in a month). Paying users have no quota, but we need to store their credit card information, which consists of an unique credit card number, an address, and a validity date. Free user accounts can be linked to paying accounts, which increases their quota. Every free account can be linked to at most one paying account.

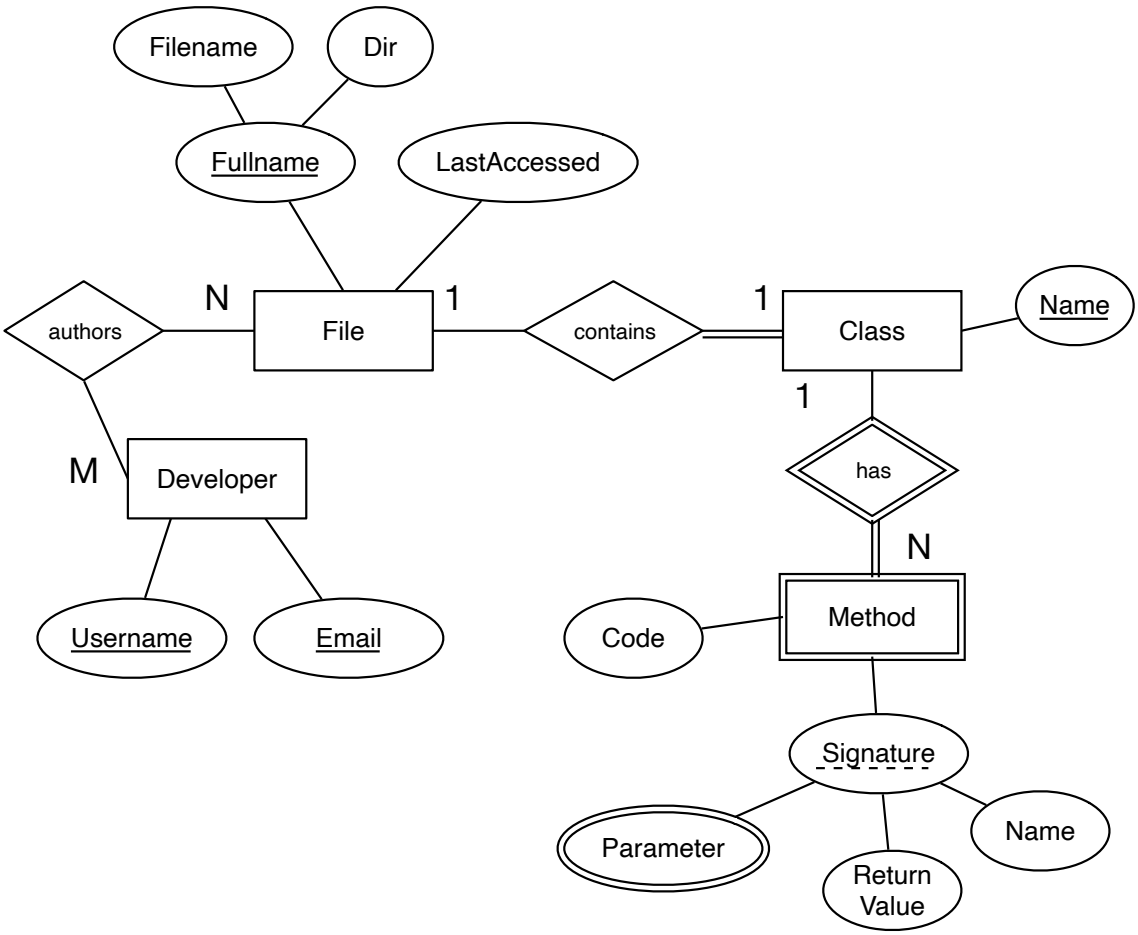
Users stream content, i.e., movies or series episodes. Whenever a user streams content, we need to save the actual duration of how long they watched. For simplicity, you can assume that users are unable to stream the same content multiple times, but of course the same content can be streamed to multiple users, and a user can stream multiple different content items. For all types of content, we need to save the the length and the release date. Further, we need to save the actual movie data. For movies, we also need to store an unique title. Episodes are collected in series. For each series, we need to store its unique title. A series can have multiple episodes, each of which has a title (which is optional and does not have to be unique). Episodes are identified through an episode key consisting of the combination of season and episode number (e.g., Season 1 - Episode 5). Episode keys are only unique in combination with the title of the series.

For each user, the system generates exactly one unique personalized recommendation list. Each list is identified through an unique id, and contains an arbitrary number of items (but always more than one). Items on the recommendation list are a combination of movies and series. Movies and series can be contained in multiple recommendation lists, but don't have to be in any.

Finally, we need to keep track of actors in our database. We need to store the (unique) names of actors, but we need to keep track of which episodes or movies they appeared in and which role they played. An actor has to have appeared in at least one episode or movie, but they could have been in many. Most streaming content has many actors appearing in them, but some also make due without any actors (e.g., animation movies). As a special feature, paying users can follow any number of actors and get informed whenever new content featuring them is released. Evidently, the same actor can be followed by many paying users.

Task 3 – Mapping an EER model (16 pts)

Consider the EER model below. Construct a relational model that represents this domain as precisely as possible. Use the notation that we used in the course to indicate relations, attributes, primary keys, and foreign keys (see Task 4 for an example of the required notation).



Task 4 – Relational Algebra (20 pts)

Relational Model:
USER(<u>username</u>, quota, reg_date)

MOVIE(<u>title</u>, genre, duration)

STREAM(<u>id</u>, user, movie, date, completion)
user → USER.username
movie → MOVIE.title

ACTOR(<u>name</u>, <u>movie</u>)
movie → MOVIE.title

(STREAM.completion is a percentage – e.g., 0.75 represents 75% – indicating how much of the movie the user streamed before stopping the stream)

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. *(5 pts per query)*

Q4.1: Return all unique movie genres.

Q4.2: Report the length of each stream. The length is defined as the percent the stream finished the movie (completion) multiplied by the duration of the movie. Return only stream id, movie title, and length.

Q4.3: Return a list of usernames along with a count how many movies they have streamed to completion (i.e., where percent is equal to 1). You do not have to consider the case of a user streaming the same movie multiple times.

Q4.4: Return all titles of movies that have either been streamed by the user “philipp” or which feature the actress “Jennifer Lawrence”.

Task 5 – SQL (20 pts)

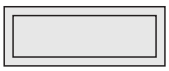
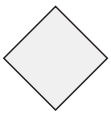
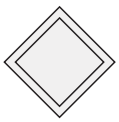
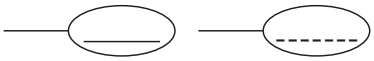
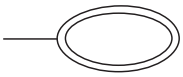
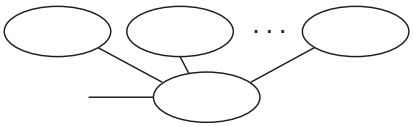
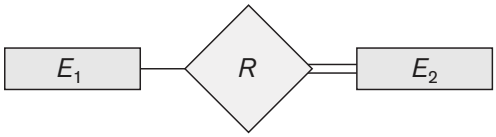
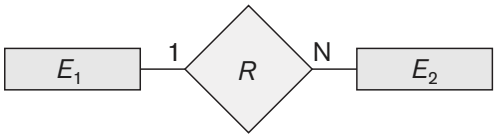
Assume that the relational in Task 4 has been implemented in a relational database. Write the following SQL queries against this database. *(5 pts per query)*

Q5.1: Provide an alphabetically sorted list of all movie titles in the genre “Thriller”.

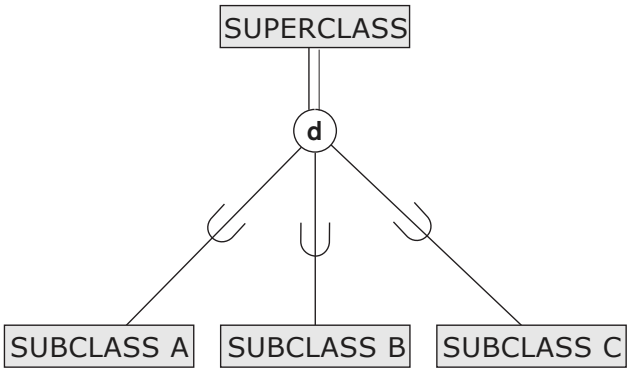
Q5.2: Return a list of all completed streams (a stream counts as completed if the ”completion” percentage is 0.98 or higher). For each completed stream return stream id, username, movie title, date, and duration of the movie. **Q5.3:** Provide a list of all unique movie genres, and how many movies there are in each genre. Call this attribute ”MovieCount”. Sort the list by ”MovieCount” in descending order.

Q5.4: Find all actors that appeared in more movies than “Bill Murray”.

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

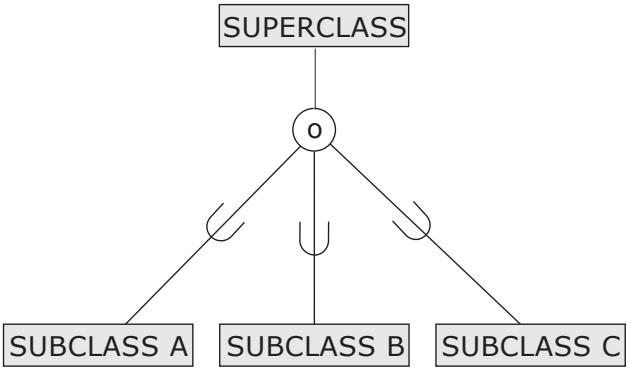


Table 8.1 Operations of Relational Algebra

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>