CS2102: Database Systems (AY2021 – Sem 1)

Midterm Exam (Makeup)

Date: 23 October 2021, Time: 12:00–13:00

Submission Instructions

1. Please read **ALL** instructions carefully.
2. All the assessment is be done using Examplify; the assessment contains
   (a) Multiple Choice Questions (MCQs): 1.1 and 1.2
   (b) Essay Questions: 2.1–2.5 (5 SQL queries)
   (c) Your Internet connection will be blocked
   (d) This is an open-book exam
3. This assessment starts at 12:00 and ends at 13:00.
   - Submit your answers by 13:00
   - No additional time will be given to submit.
4. For the SQL questions, there are additional instructions below; in a nutshell:
   - Use an IDE or text editor to write your queries and test them using `psql`
   - Prepare your final answer before submitting it to Examplify according to the instructions in Section 2
   - Allocate sufficient time to prepare your final answers and to copy-&-paste them into Examplify.
5. Failure to follow each of the instructions above may result in deduction of your marks.

Good Luck!

1 ER model & Relational Algebra (10 Marks)

For this task, we consider the following simplified company database. *Employees* are identified by a unique employee id (`eid`), and they have a name and a dob (date of birth). An employee is either a part-time or full-time employee. For part-time employees their hours, and for full-time employees their role is recorded. Full-time Employees can have *dependants* which are uniquely identified by the employee id and the dependant name; they also have a dob. Full-time employees can hold *policies* which are identified by a unique policy id (`pid`), and they have a cost attribute (values are in dollars). Each dependant is covered by one or more policies with a certain percentage (a value between 0 and 100, inclusive of both) for each policy.

We can model this application requirements using the following ER diagram:

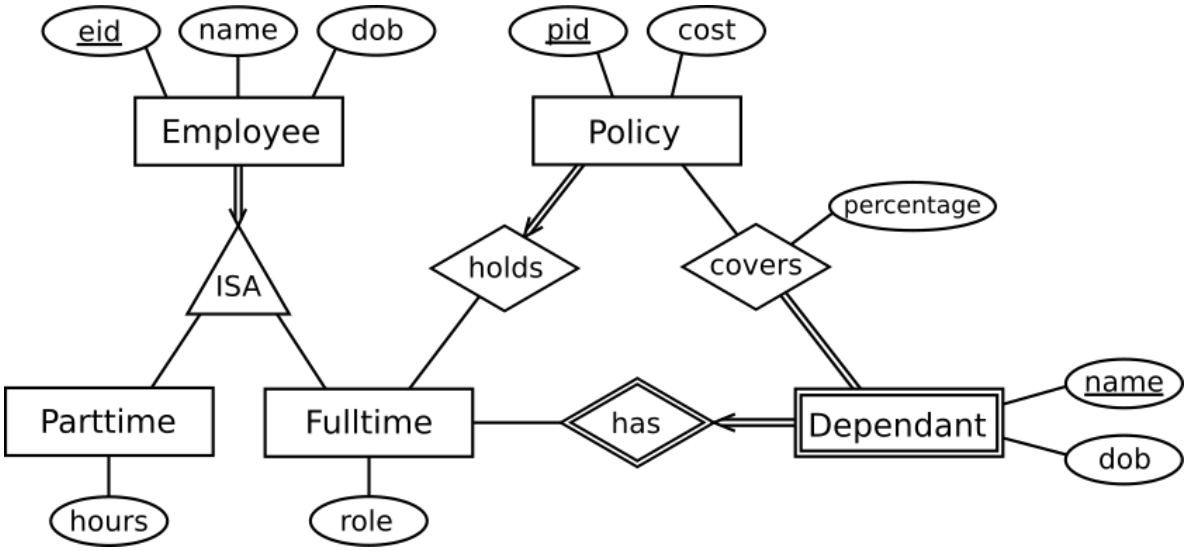


Figure 1: ER Diagram for Online Shop

Based on this ER diagram, the following database schema has been created:

- `Employee(eid, name, dob)`
- `Parttime(eid, hours)`
- `Fulltime(eid, role)`
- `Dependant(eid, name, dob)`
- `Policy(pid, cost, eid)`
- `Covers(eid, name, pid, percentage)`

All id attributes are integers; for the other attributes you can assume meaningful data types. The foreign key constraints are as follows:

- `Parttime.eid`→`Employee.eid`
- `Fulltime.eid`→`Employee.eid`
- `Dependant.eid`→`Employee.eid`
- `Policy.eid`→`Fulltime.eid`
- `Covers.(eid,name)`→`Dependant.(eid,name)`
- `Covers.pid`→`Policy.pid`

1.1 Constraints (4 Marks). Let's assume the application requirements include the following list of constraints:

- The value of `percentage` must be a value between 0 and 100 inclusive of both.
- An `Employee` can be a `Parttime` or `Fulltime`, but not both
- The total cost of all `Policy` for a `Fulltime` must be less than or equal to S$100,000
- A `Dependant` is covered by at least one `Policy`
- All `Employee` and `Policy` are uniquely identified
- A `Fulltime`'s policies cover only this employee's `Dependant`
- Each `Employee` must either be a `Parttime` or `Fulltime`
- Each `Policy` is held by exactly one `Fulltime`

Given these 8 constraints, answer the following 2 questions:

(a) Which of constraints above are **NOT** captured by the ER diagram? Select all that apply.

Solution:

- The total cost of all policies for a `Fulltime` must be than or equal to $S100,000.
- The value of percentage must be a value between 0 and 100 inclusive of both.
- A `Fulltime`'s policies cover only this employee's dependants

(b) Which constraints can **NOT** be captured when converting the ER Diagram into the database schema without using `TRIGGER`? Select all that apply.

Solution:

- A `Dependant` is covered by at least one `Policy`
- A `Fulltime`'s policies cover only this employee's `Dependant`
- An `Employee` can be a `Parttime` or `Fulltime`, but not both
- Each `Employee` must either be a `Parttime` or `Fulltime`
- The total cost of all policies for a `Fulltime` must be less than or equal to $S100,000

1.2 Equivalence of queries (6 Marks). Based on the given database schema above, the following questions are about determining if two queries – written in SQL or RA (Relational Algebra) – are *equivalent*. Two queries are considered equivalent if they return the same result for any database. Note that you will not be required to write your own SQL queries or algebra expressions.

(a) Below are 5 pairs of Relational Algebra expressions Q_1 and Q_2. Select all pairs where Q_1 is equivalent to Q_2!

A

$$Q_1 = \pi_{pid}(\sigma_{cost \geq 10000}(Policy \bowtie Covers))$$
$$Q_2 = \pi_{pid}(\sigma_{cost \geq 10000}(Policy)) \bowtie Covers$$

B

$$Q_1 = \pi_{eid,name}(\sigma_{role='sales'}(Fulltime \rtimes Employee))$$
$$Q_2 = \pi_{eid,name}(Employee \bowtie \sigma_{role='sales'}(Fulltime))$$

C

$$Q_1 = \pi_{eid,name}(\pi_{eid}(\sigma_{role='sales'}(Fulltime)) \bowtie \pi_{eid,name}(Dependant))$$
$$Q_2 = \pi_{eid,name}(\sigma_{role='sales'}(Fulltime) \bowtie Dependant)$$

D

$$Q_1 = \pi_{eid,name}(Dependant \bowtie \sigma_{percentage=100}(Covers))$$
$$Q_2 = \pi_{eid,name}(\sigma_{(percentage=100)\ \wedge\ (eid=eid2)}(Dependant \times \rho_{eid2 \leftarrow eid}(Covers)))$$

E

$$Q_1 = \pi_{eid,name}(Fulltime \bowtie_{eid=eid2} (\rho_{eid2 \leftarrow eid}(\sigma_{percentage=100}(Policy))))$$
$$Q_2 = \pi_{eid,name}(\sigma_{percentage=100}(Fulltime \bowtie Policy))$$

Solution:

- B (the Left Outer Join always collapses to the Inner Join)
- C (should be the same since Natural join does not "duplicate" shared attributes)
- E (the renaming and Inner Join simply mimics the Natural Join)

The following pairs are incorrect:

- A: Q_1 has only 1 attribute, while Q_2 has 4
- D: The join must be over `eid` **and** `name`; the latter is missing

(b) Given the following SQL query:

```
SELECT d.eid, d.name
FROM Dependant d
AND NOT EXISTS (SELECT 1
                FROM Covers c, Policy p
                WHERE c.pid = p.pid
                AND c.eid = d.eid
                AND c.name = d.name
                AND p.cost <= 10000);
```

Which of the following Relational Algebra expressions are equivalent to this SQL query? Select all that apply.

A

$$Q_1 = \pi_{eid,name,pid}(Dependant \bowtie Covers)$$
$$Q_{ans} = \pi_{eid,name}(Q_1 \bowtie \sigma_{cost \leq 10000}(Policy))$$

B

$$Q_1 = \sigma_{cost \leq 10000}(Covers \bowtie \pi_{pid,cost}(Policy))$$
$$Q_{ans} = \pi_{eid,name}(\sigma_{pid\ IS\ NULL}(Dependant \rtimes Q_1))$$

C

$$Q_1 = \pi_{eid,name,pid}(Dependant \bowtie Covers)$$
$$Q_{ans} = \pi_{eid,name}(\sigma_{(cost \leq 10000)\ \wedge\ (pid\ IS\ NULL)}(Q_1 \rtimes (\pi_{pid}(Policy))))$$

D

$$Q_1 = Covers \bowtie \pi_{pid}(\sigma_{cost \leq 10000}(Policy))$$
$$Q_{ans} = \pi_{eid,name}(Dependant) - \pi_{eid,name}(Q1)$$

E None of the others.

Solution: The query find all dependants with at least 1 policy larger than 10k SGD.

- B, D

The following are not equivalent:

- A: This query finds all dependants with policies less the 10k, i.e., the "opposite"
- C: `cost` is no longer available after the projection in Q_1

(c) Given the following Relational Algebra expression:

$$Q = \pi_{eid}(\sigma_{percentage=100}(Fulltime \bowtie Policy \bowtie \pi_{pid,percentage}(Covers)))$$

Which of the following SQL queries are equivalent to this Relational Algebra expression? Select all that apply.

A:

```
SELECT f.eid
FROM Fulltime f, Policy p, Covers c
WHERE f.eid = p.eid
AND p.pid = c.pid
AND c.percentage = 100;
```

B:

```
SELECT eid
FROM Fulltime f
WHERE eid = ANY (SELECT eid
                 FROM Policy p, Covers c
                 WHERE p.pid = c.pid
                 AND c.percentage = 100);
```

C:

```
SELECT eid
FROM Fulltime f
WHERE eid IN (SELECT eid
              FROM Policy
              WHERE pid IN (SELECT pid
                            FROM Covers
                            WHERE percentage = 100));
```

D:

```
SELECT eid
FROM Fulltime f
WHERE EXISTS (SELECT 1
              FROM Policy p, Covers c
              WHERE p.pid = c.pid
              AND c.percentage = 100);
```

E: None of the others

Solution: The query finds all full-time employees who hold at least 1 policy that covers at least 1 dependant with 100%.

- B, C

The following are not equivalent

- A: Requires DISTINCT since SQL is multiset-based and RA is set-based
- D: The EXIST subquery is not correlated with the outer query

2 Formulating SQL Queries (10 Marks)

In this part of the exam your task is to formulate queries in SQL. We use the same database that we introduced in Lectures 5 & 6. This means you can directly run and test your queries using `psql`. If needed, we provide the ER diagram and the `CREATE TABLE` statements for the database schema in the Appendix.

Instructions. To ensure successful submission and grading of your SQL queries adhere to the following instructions:

- For each question, you are to write a **single CREATE OR REPLACE VIEW SQL statement** to answer the question. You **MUST** use the view schema provided for each question without changing any part of the view schema (i.e., view name, column names, or the order of the columns). For example, the provided view schema for the question is `"q1 (airport_name)"`, and if your answer to this question is the query `"SELECT name FROM airports;"`, then you must enter your answer in the answer box as follows:

```
CREATE OR REPLACE VIEW q1 (airport_name) AS
SELECT name
FROM airports
;
```

- Each `CREATE OR REPLACE VIEW` statement must terminate with a semicolon.
- Each answer must be a syntactically valid SQL query. Test with `psql`!
- Each question must be answered independently of other questions, i.e., the answer for a question must not refer to any other view that is created for another question.
- Your answers must not use SQL constructs that are not covered in Lectures 1-6.
- Your answers must be executable on PostgreSQL. Test with `psql`!
- You must not enter any extraneous text for your answer (e.g., "My answer is: ...), or enter multiple answers (i.e., submit multiple SQL statements). Your answer should not contain any comments.

Recommendations. We recommend the following steps to answer each question:

- Write the answer query using your IDE or text editor of choice, and test it on your machine with `psql`.
- Once you are happy with your query, add the `CREATE OR REPLACE VIEW` statement for the corresponding question in front of the query. You can test the complete statement again with `psql` and create the view.
- Copy the complete `CREATE OR REPLACE VIEW` statement into the answer field in Examplify (don't forget to the semicolon at the end!)

2.1 Query 1 (1 Mark): Find all Australian airports that contain `'International'` in their name! You can use the knowledge that the country code for Australia is `'AU'`.

```
CREATE OR REPLACE VIEW q1 (airport_name) AS
-- your query goes here
;
```

Solution:

```
CREATE OR REPLACE VIEW q1 (airport_name) AS
SELECT name AS airport_name
FROM airports
WHERE name LIKE '%International%'
AND country_iso2  = 'AU'
;
```

2.2 Query 2 (2 Marks): Find the names of all cities with 4 or more airports! Include the country code and the number of airports in the result! (Note: we need the country code since cities in different countries can have the same name.)

```
CREATE OR REPLACE VIEW q2 (country_iso2, city_name, airport_count)
   AS
-- your query goes here
;
```

Solution:

```
CREATE OR REPLACE VIEW q2 (country_iso2, city_name, airport_count
   ) AS
SELECT c.country_iso2, c.name AS city_name, COUNT(*) AS
   airport_count
FROM cities c, airports a
WHERE c.name = a.city
AND c.country_iso2 = a.country_iso2
GROUP BY c.country_iso2, c.name
HAVING COUNT(*) >= 4;
```

2.3 Query 3 (2 Marks): Find the names and population sizes of all cities with population larger than every individual country in the continent of `'Oceania'`! Sort the result by population size from the largest to the smallest countries!

```
CREATE OR REPLACE VIEW q3 (city_name, population) AS
-- your query goes here
;
```

Solution:

```
CREATE OR REPLACE VIEW q3 (city_name, population) AS
SELECT name AS city_name, population
FROM cities
WHERE population > ALL (SELECT population
                        FROM countries
                        WHERE continent = 'Oceania')
ORDER BY population DESC
;
```

2.4 Query 4 (2 Marks): Find the names of top-20 countries with the most airports per capita! For example, a country with 2 airports and a population of 1,000,000 would have 0.000002 airports per capita.

```
CREATE OR REPLACE VIEW q4 (country_name) AS
-- your query goes here
;
```

Solution:

```
CREATE OR REPLACE VIEW q4 (country_name) AS
SELECT c.name AS country_name
FROM countries c, airports a
WHERE c.iso2 = a.country_iso2
GROUP BY c.iso2
ORDER BY COUNT(*) / CAST(c.population AS DECIMAL) DESC
LIMIT 20
;
```

2.5 Query 5 (3 Marks): Find the names of all cities that are (a) the capital of a country (*i.e.*, `capital='primary'`) with no land border and (b) are a city with no airport. For example, Valetta is the capital of the island of Malta, and Valetta has no airport. Therefore, Valetta should be in the result sets. (Recall that a country without a land border is in Table `borders` with `country2_iso2 IS NULL`.)

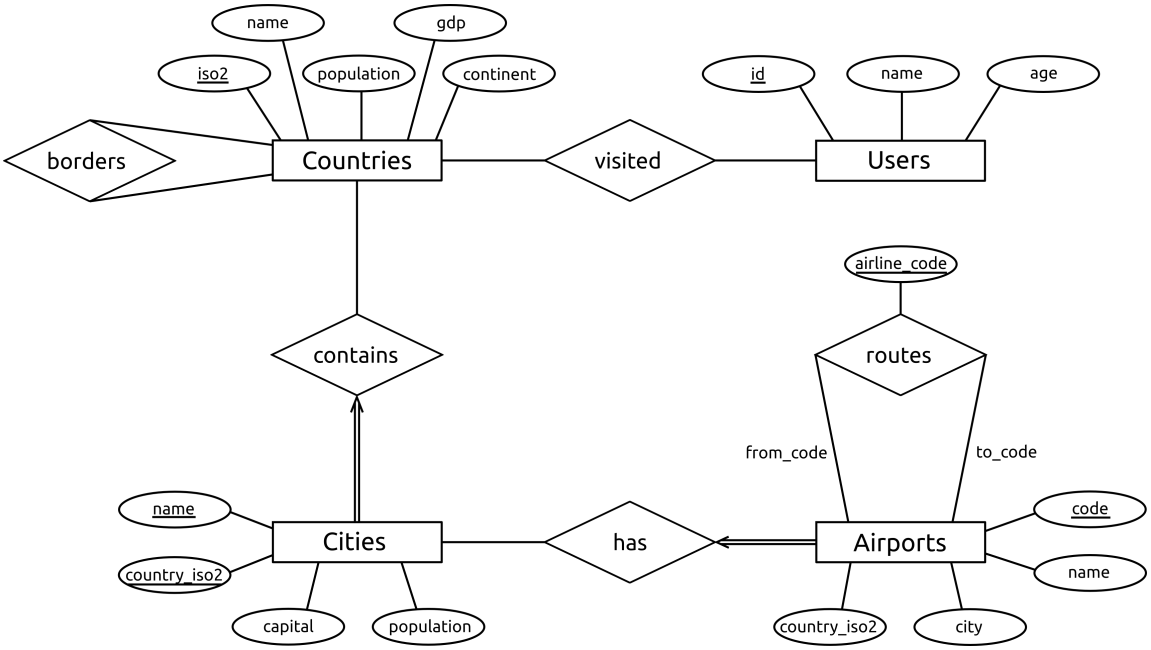
```
CREATE OR REPLACE VIEW q5 (city_name) AS
-- your query goes here
;
```

Solution:

```
CREATE OR REPLACE VIEW q5 (city) AS
SELECT c.name AS city_name
FROM cities c, borders b
WHERE capital = 'primary'
AND c.country_iso2 = b.country1_iso2
and b.country2_iso2 IS NULL
AND NOT EXISTS (SELECT 1
                FROM airports a
                WHERE a.city = c.name
                AND a.country_iso2 = c.country_iso2);
```

A Appendix

A.1 ER Diagram of Database for Task 2



A.2 CREATE TABLE statements of Database for Task 2

```
CREATE TABLE countries (
 iso2          CHAR(2) PRIMARY KEY,
 name          VARCHAR(255) UNIQUE,
 population    INTEGER CHECK (population >= 0),
 gdp           BIGINT CHECK (gdp >= 0),
 continent     VARCHAR(255) NOT NULL
);

CREATE TABLE cities (
 name           VARCHAR(255),
 country_iso2   CHAR(2),
 capital        VARCHAR(255),
 population     INTEGER CHECK (population >= 0),
 PRIMARY KEY (name, country_iso2),
 FOREIGN KEY (country_iso2) REFERENCES countries (iso2)
);

CREATE TABLE borders (
 country1_iso2  CHAR(2),
 country2_iso2  CHAR(2),
 PRIMARY KEY (country1_iso2, country2_iso2),
 FOREIGN KEY (country1_iso2) REFERENCES countries (iso2),
 FOREIGN KEY (country2_iso2) REFERENCES countries (iso2)
);

CREATE TABLE airports (
 code CHAR(3)       PRIMARY KEY,
```

```
 name VARCHAR(255)  NOT NULL,
 city VARCHAR(255)  NOT NULL,
 country_iso2       CHAR(2),
 FOREIGN KEY (city,country_iso2) REFERENCES cities (name,country_iso2)
);

CREATE TABLE routes (
 from_code     CHAR(3),
 to_code       CHAR(3),
 airline_code  CHAR(3),
 PRIMARY KEY (from_code, to_code, airline_code),
 FOREIGN KEY (from_code) REFERENCES airports (code),
 FOREIGN KEY (to_code) REFERENCES airports (code)
);

CREATE TABLE users (
 user_id  INTEGER PRIMARY KEY,
 name     VARCHAR(255) NOT NULL,
 age      INTEGER
);

CREATE TABLE visited (
 user_id  INTEGER,
 iso2     CHAR(2),
 PRIMARY KEY (user_id, iso2)
);

CREATE TABLE connections AS
SELECT DISTINCT from_code, to_code
FROM routes;
```