CS2102: Database Systems (AY2223 – Sem 2)
Midterm

Date: 27 February 2023, Time: 12:30–13:30

Submission Instructions

1. Please read **ALL** instructions carefully.
2. All the assessment is be done using Examplify; the assessment contains
   (a) Multiple Response Questions (MRQs): Question 1 - 5
   (b) Fill-in-the-Blank: Question 6
   (c) SQL Queries: Question 7 - 9
3. This is an open-book assessment
4. Your Internet connection will be blocked for the duration of the assessment
5. This assessment starts at 12:30 and ends at 13:30.
   - Submit your answers by 13:30
   - No additional time will be given to submit.
6. 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` (*or other program of your choice*)
   - 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.
7. Failure to follow each of the instructions above may result in deduction of your marks.

Good Luck!

1 ER Model & Relational Algebra (*14 Marks*)

For this section, we will consider the following ER diagram for teaching assistants (*TA*).

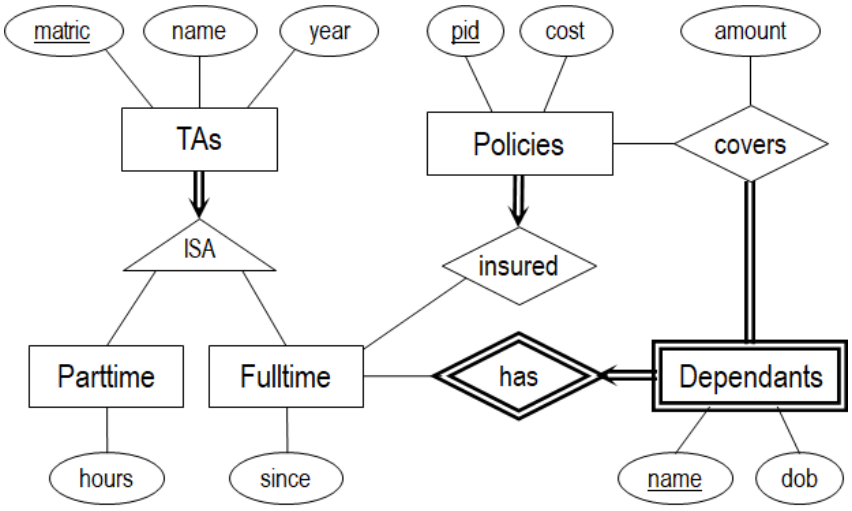


Figure 1: ER Diagram for Teaching Assistant

Based on the ER diagram, the following database schema has been created. Recap that the key attributes are underlined.

- TAs(matric, name, year)
- Parttime(matric, hours)
- Fulltime(matric, since)
- Dependants(matric, name, dob)
- Policies(pid, matric, cost)
- Covers(matric, name, pid, amount)

Additionally, the foreign key constraints are also specified.

- (Parttime.matric) $\rightsquigarrow$ (TAs.matric)
- (Fulltime.matric) $\rightsquigarrow$ (TAs.matric)
- (Dependants.matric) $\rightsquigarrow$ (Fulltime.matric)
- (Policies.matric) $\rightsquigarrow$ (Fulltime.matric)
- (Covers.matric,Covers.name) $\rightsquigarrow$ (Dependants.matric,Dependants.name)
- (Covers.pid) $\rightsquigarrow$ (Policies.pid)

You may assume reasonable data type for each attributes. For the attribute `since` of full time TA, we simply want to record the year so it is an integer. Additionally, we let the ids including `matric` to be integer.

Note that there may be mistakes in the ER diagram and or schema. In such cases, we want the database to satisfy the following set of constraints.

(A) Since the university is founded only in 1990, there should be no full time TA working since before 1990.

(B) A TA cannot be both part time and full time TA.

(C) The total cost of all policies for a full time TA must be less than or equal to $2,000.

(D) A dependant is covered by at least one policy but can be covered by multiple policies.

(E) All TAs and policies can be uniquely identified.

(F) Each full time TA policies cover only this TA's dependants.

(G) Each TA must be either a part time or full time TA.

(H) Each policy is held by exactly one full time TA.

1. (3 points) **ER Diagram Constraints**

Select **ALL** the constraints above that are captured by the ER diagram.

☐ **B** ☐ **D** **E** ☐ **G** **H**

2. (3 points) **Schema Constraints**

Select **ALL** the constraints above that can be captured when converting the ER diagram into the database schema using the concepts you have learnt so far. In particular, you cannot add new attributes on the schema above, change the key attributes, and you cannot use triggers.

A ☐ ☐ ☐ **E** ☐ ☐ **H**

In the next **three (3)** questions, we will use the schema as tables assuming all the constraints in the previous two questions are all satisfied. The attributes of the schema will be the columns of the tables. Recap that key attributes are underlined and we assume these will also be the key attributes of the table. You should also assume that the foreign keys are always respected.

Further recap that two queries (*can be SQL or relational algebra expressions*) are equivalent if and only if they return the same result for any database. Assume that none of the queries can produce errors. Note that SQL uses multi-set (*or bag*) semantics while relational algebra uses set semantics.

3. (2 points) **Relational Algebra Equivalence**

Select **ALL** pairs where Q_1 is equivalent to Q_2.

(A)
$$Q_1 = \pi_{[\text{pid}]}(\sigma_{[\text{cost}\leq 1000]}(\text{Policies} \bowtie \text{Covers}))$$
$$Q_2 = \pi_{[\text{pid}]}(\sigma_{[\text{cost}\leq 1000]}(\text{Policies})) \bowtie \text{Covers}$$

(✓)
$$Q_1 = \pi_{[\text{matric,name}]}(\sigma_{[\text{since}\leq 2022]}(\text{Fulltime} \bowtie \text{TAs}))$$
$$Q_2 = \pi_{[\text{matric,name}]}(\text{TAs} \bowtie \sigma_{[\text{since}\leq 2022]}(\text{Fulltime}))$$

(✓)
$$Q_1 = \pi_{[\text{matric,name}]}(\ \pi_{[\text{matric}]}(\sigma_{[\text{since}\leq 2022]}(\text{Fulltime})) \bowtie \pi_{[\text{matric,name}]}(\text{Dependants})\ )$$
$$Q_2 = \pi_{[\text{matric,name}]}(\sigma_{[\text{since}\leq 2022]}(\text{Fulltime}) \bowtie \text{Dependants})$$

(✗)
$$Q_1 = \pi_{[\text{matric,name}]}(\text{Fulltime} \bowtie_{[\text{matric}=\text{matric2}]} (\rho_{[\text{matric2}\leftarrow\text{matric}]}(\sigma_{[\text{cost}\leq 1000]}(\text{Policies}))))$$
$$Q_2 = \pi_{[\text{matric,name}]}(\sigma_{[\text{cost}\leq 1000]}(\text{Fulltime} \bowtie \text{Policies}))$$

(E) *None of the above*

4. (2 points) **RA Equivalent of SQL**

Given the following SQL query.

```
SELECT D.matric, D.name
FROM   Dependants D
WHERE  NOT EXISTS (
    SELECT 1
    FROM   Covers C, Policies P
    WHERE  C.pid = P.pid   AND  C.matric = D.matric
      AND  C.name = D.name AND  P.cost <= 1000
  );
```

Select **ALL** relational algebras that is equivalent to the SQL query above. Consider only Q_{ans}.

(A)
$$Q_1 = \pi_{[\text{matric,name,pid}]}(\text{Dependants} \bowtie \text{Covers})$$
$$Q_{ans} = \pi_{[\text{matric,name}]}(Q_1 \bowtie \sigma_{[\text{cost} \leq 1000]}(\text{Policies}))$$

(✓)
$$Q_1 = \sigma_{[\text{cost} \leq 1000]}(\text{Covers} \bowtie \pi_{[\text{pid,cost}]}(\text{Policies}))$$
$$Q_{ans} = \pi_{[\text{matric,name}]}(\sigma_{[\text{pid IS NULL}]}(\text{Dependants} \rightarrowtriangle Q_1)$$

(C)
$$Q_1 = \pi_{[\text{matric,name,pid}]}(\text{Dependants} \bowtie \text{Covers})$$
$$Q_{ans} = \pi_{[\text{matric,name}]}(\sigma_{[(\text{cost} \leq 1000) \wedge (\text{pid IS NULL})]}(Q_1 \rightarrowtriangle (\pi_{[\text{pid}]}(\text{Policies})))$$

(✓)
$$Q_1 = \text{Covers} \bowtie \pi_{[\text{pid}]}(\sigma_{[\text{cost} \leq 1000]}(\text{Policies}))$$
$$Q_{ans} = \pi_{[\text{matric,name}]}(\text{Dependants}) - \pi_{[\text{matric,name}]}(Q1)$$

(E) *None of the above*

5. (2 points) **SQL Equivalent of RA**

Given the following Relational Algebra expression:

$$Q = \pi_{[\text{matric}]}(\sigma_{[\text{amount}=50000]}(\text{Fulltime} \bowtie \text{Policies} \bowtie \pi_{[\text{pid,amount}]}(\text{Covers})))$$

Select**ALL** SQL queries that are equivalent to the relational algebra expression above.

(A) Query A

```
SELECT F.matric
FROM   Fulltime F, Policies P, Covers C
WHERE  F.matric = P.matric
  AND  P.pid = C.pid
  AND  C.amount = 50000;
```

(✗) Query B

```
SELECT matric
FROM   Fulltime F
WHERE  matric = ANY (
    SELECT matric
    FROM   Policies P, Covers C
    WHERE  P.pid = C.pid
      AND  C.amount = 50000
  );
```

(✓) Query C

```
SELECT matric
FROM   Fulltime F
WHERE  matric IN (
    SELECT matric
    FROM   Policies
    WHERE  pid IN (
        SELECT pid
        FROM Covers
        WHERE amount = 50000
      )
  );
```

(D) Query D

```
SELECT matric
FROM   Fulltime F
WHERE  EXISTS (
    SELECT 1
    FROM   Policies P, Covers C
    WHERE  P.pid = C.pid
      AND  C.amount = 50000
  );
```

(E) *None of the above*

6. (2 points) **Cardinality**

Consider the same ER diagram below. The numbers besides TAs and Policies indicate the number of entries in TAs and Policies entitiy sets. So, in the database there are 100 TAs and 20 Policies.

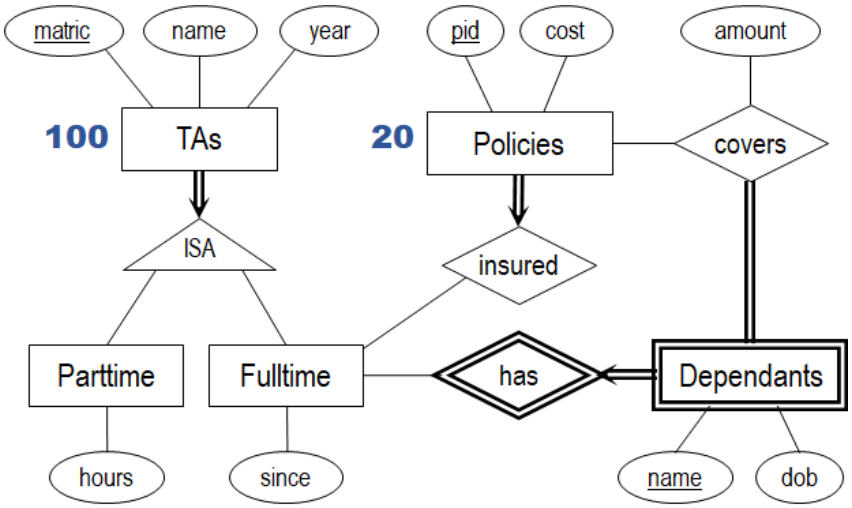


Figure 2: Numbers in the ER Diagram Indicating the Number of Entries

Determine the minimum and maximum numbers of entries in Parttime and Fulltime entity sets.

(a) (1 point) `Parttime`:

- `Minimum:` 0
- `Maximum:` 99

(b) (1 point) `Fulltime`:

- `Minimum:` 1
- `Maximum:` 100

2 SQL Queries (*6 Marks*)

In this part of the exam your task is to formulate queries in SQL. We use the same database that we introduced in Assignment 1. 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, if the provided view schema for the question is `"q7 (code, name)"`, and if your answer to this question is the query `"SELECT 'dummy' AS code, 'answer' AS name"`, then you must enter your answer in the answer box as follows:

```
CREATE OR REPLACE VIEW q7 (code, name) AS
SELECT 'dummy' AS code, 'answer' AS name
;
```

- 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!)

7. (2 points) We say that a course is an introductory course if it (1) begins with the word `'Intro'` in its name and (2) must have a digit `1` in the third place of the course code.

For instance, `'CS101'` satisfies condition (2) because the third place in the course code is `1`. Its name `'Intro to Programming'` also satisfies condition (1) because it begins with `'Intro'` in its name and it is a word since it is followed by a space.

Find all introductory course code and name. There should be 5 rows in the output from the sample data including MT101 which is Intro to Marketing.

```
CREATE OR REPLACE VIEW q7 (code, name) AS
-- your query goes here
;
```

Solution:

```
CREATE OR REPLACE VIEW q7 (code, name) AS
  SELECT DISTINCT code, name
  FROM   Courses
  WHERE  code LIKE '__1%'      -- or '__1__'
    AND  name LIKE 'Intro %'   -- notice the space
;
```

8. (2 points) Find all departments that offers at least 4 *distinct* courses in semester 2. Include the department address and the number of courses offered in the result.

There should be 5 rows in the output with two departments offering only 4 courses including Marketing and Philosophy.

```
CREATE OR REPLACE VIEW q8 (name, address, course_count) AS
-- your query goes here
;
```

Solution:

```
CREATE OR REPLACE VIEW q8 (name, address, course_count) AS
  SELECT DISTINCT D.name, D.address, COUNT (DISTINCT C.code)
  FROM   Departments D, Courses C, Offers O
  WHERE  D.name = O.name AND O.code = C.code
    AND  O.sem = 2
  GROUP BY D.name
  HAVING   COUNT (DISTINCT C.code) >= 4
;
```

9. (2 points) Find the codes and names of the top 3 faculties with the most number of departments. If there are two faculties with the same number of departments, the department with the smaller code lexicographically (*i.e., the usual lexicographical ordering*) should have higher ranking.

For instance, if both `SOC` and `SCI` somehow have the same number of departments (*not in our sample data*), then we should list `SCI` before `SOC` because of the lexicographical ordering. This may affect the output if both tied for the third position. In such cases, only `SCI` is in the result.

Note that two faculties are differentiated by their faculty code. So two faculties may have the same name but different code. In particular, for top 3 faculties, we really mean 3 different faculties with different codes. So you may have duplicate name.

Order the result in descending order of the ranking above. There should only be *at most* 3 rows in the output and the ordering matters. Note that the result will not contain any ambiguity as we also order based on the faculty code.

In the sample data, the highest count in the result is SCI with 5 departments and the lowest count in the result is SOC with only 2 departments.

We also exclude faculties *without* any departments. So, if there are only 2 faculties in the database or there are only 2 faculties with departments, there should only be 2 rows in the output.

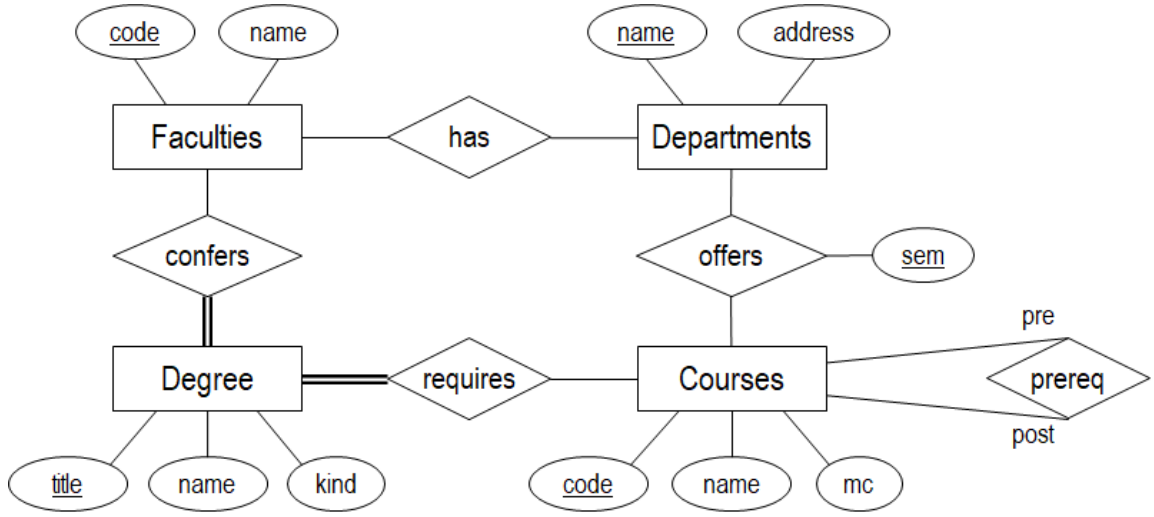
```
CREATE OR REPLACE VIEW q9 (code, name, dept_count) AS
-- your query goes here
;
```

Solution:

```
CREATE OR REPLACE VIEW q9 (code, name, dept_count) AS
  SELECT DISTINCT F.code, F.name, COUNT (DISTINCT H.name) AS
   dept_count
  FROM   Faculties F, Has H
  WHERE  F.code = H.code
  GROUP BY F.code
  ORDER BY dept_count DESC, F.code ASC
  LIMIT    3
;
```

A Appendix

A.1 ER Diagram of Database for Section 2



A.2 CREATE TABLE statements of Database for Section 2

```
CREATE TABLE Faculties (
  code  VARCHAR(4) PRIMARY KEY,
  name  VARCHAR(50) NOT NULL
);

CREATE TABLE Departments (
  name    VARCHAR(50) PRIMARY KEY,
  address VARCHAR(50)
);

CREATE TABLE Has (
  code  VARCHAR(4) REFERENCES Faculties,
  name  VARCHAR(50) REFERENCES Departments,
  PRIMARY KEY (code, name)
);

CREATE TABLE Degree (
  title VARCHAR(15) PRIMARY KEY,
  name  VARCHAR(50) NOT NULL,
  kind  VARCHAR(10) NOT NULL
);

CREATE TABLE Courses (
  code  VARCHAR(5) PRIMARY KEY,
  name  VARCHAR(50) NOT NULL,
  mc    INT NOT NULL CHECK (mc > 0)
);
```

```
CREATE TABLE Offers (
  code  VARCHAR(5) REFERENCES Courses,
  name  VARCHAR(50) REFERENCES Departments,
  sem   INT NOT NULL,
  PRIMARY KEY (code, name, sem)
);

CREATE TABLE Prereq (
  pre   VARCHAR(5) REFERENCES Courses,
  post  VARCHAR(5) REFERENCES Courses,
  PRIMARY KEY (pre, post)
);

CREATE TABLE Requires (
  title VARCHAR(15) REFERENCES Degree,
  code  VARCHAR(5) REFERENCES Courses,
  PRIMARY KEY (title, code)
);

CREATE TABLE Confers (
  title VARCHAR(15) REFERENCES Degree,
  code  VARCHAR(4) REFERENCES Faculties,
  PRIMARY KEY (title, code)
);
```