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 12:30 and ends at 13:30.
   - Submit your answers by 13:30
   - 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 simple application scenario of an online shop: *Customers* are identified by a unique customer id and they have a name. Customers can make *Orders*, and we store the date of the order. Orders are identified by a unique order id, and we store the total price of the order. Each order includes 1 or more *Products*. Products are identified by a unique product id, and also store the name and the price of the product. An order can include the same product multiple times; we therefore record the quantity. Lastly, customers can rate products by giving them a score 1, 2, ..., or 5 (e.g., representing 1 star, 2 stars, etc).

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

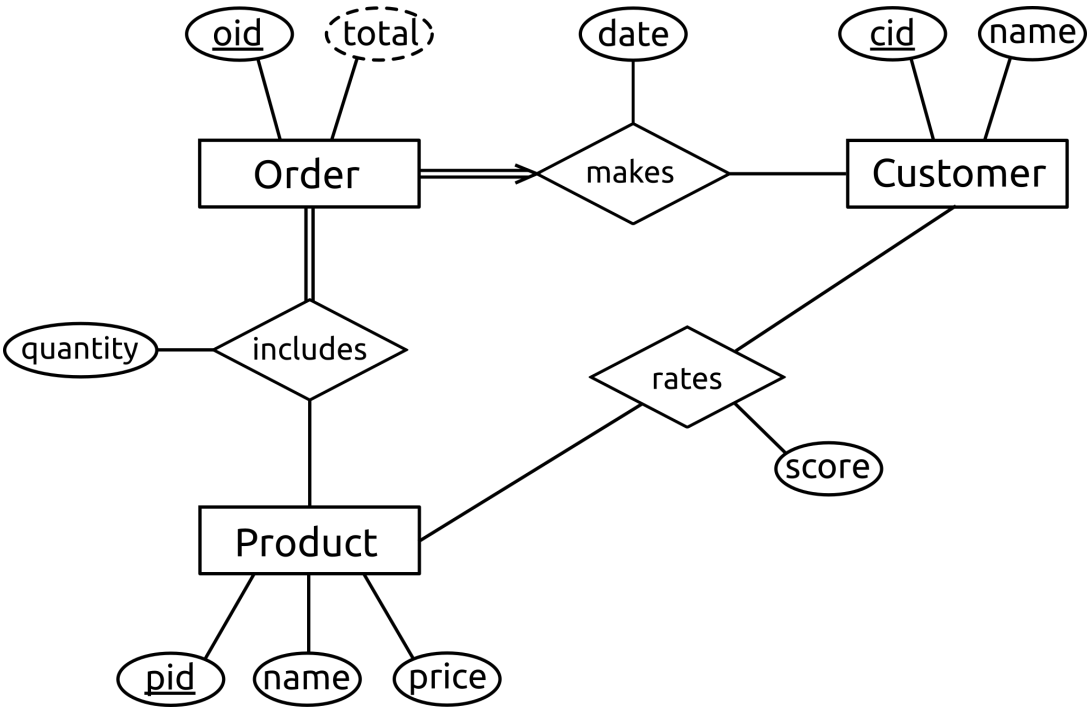


Figure 1: ER Diagram for Online Shop

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

- `Customer(cid, name)`
- `Order(oid, total, date, cid)`
- `Product(pid, name, price)`
- `Includes(oid, pid, quantity)`
- `Rates(cid, pid, score)`

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

- `Order.cid`→`Customer.cid`
- `Rates.cid`→`Customer.cid`
- `Rates.pid`→`Product.pid`
- `Includes.pid`→`Product.pid`
- `Includes.oid`→`Order.oid`

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

- The "total" price of an order must be the sum of the prices of all included items.
- An order is made by exactly one customer.
- Each order, product, and customer is uniquely identifiable.
- Customers can only rate items they have bought at least once.
- All products need to have different names.
- The "quantity" must be larger than 0.
- An order cannot be empty and must include at least 1 product.
- The value for "score" can only be 1, 2, ..., or 5.

Given these 8 constraints, answer the following 2 questions:

(a) Which of constraints above are captured by the ER diagram? Select all that apply.

(b) Which constraints can be captured when converting the ER Diagram into the database schema without using `TRIGGER`? Select all that apply.

1.2 Querying the database using the Relational Algebra (6 Marks). Based on the given database schema, the following questions are about querying this database using Relational Algebra expressions. Note that you will not be required to write your own expressions.

(a) Which of the following Relational Algebra expressions do **NOT** return the name of the customers that made at least one order worth more than $1,000? Select all that apply.

A $Q_{ans} = \pi_{name}(\sigma_{total>1000}(\rho_{cid2 \leftarrow cid}(Customer) \bowtie \pi_{total,cid}(Order)))$

B $Q_{ans} = \pi_{name}(\sigma_{cid=cid2 \ \wedge \ total>1000}(\rho_{cid2 \leftarrow cid}(Customer) \bowtie Order))$

C $Q_{ans} = \pi_{name}(Customer \bowtie \sigma_{total>1000}(Order))$

D $Q_{ans} = \pi_{name}(\sigma_{total>1000}(Customer \bowtie (\pi_{cid}(Customer) - \pi_{cid}(Order)))$

E None of the above.

(b) Which of the following Relational Algebra expressions do **NOT** return the name of all products that have never been purchased? Select all that apply.

A $Q_{ans} = \pi_{name}(Product \bowtie (\pi_{pid}(Product) - \pi_{pid}(Includes)))$

B
$$Q_1 = \pi_{pid,name}(Product) \ltimes_{pid=pid2} (\rho_{pid2 \leftarrow pid}(Includes))$$
$$Q_{ans} = \pi_{name}(\sigma_{pid2 \text{ IS NULL} \ \wedge \ pid \text{ IS NULL}}(Q_1))$$

C $Q_{ans} = \pi_{name}(\pi_{pid,name}(Product) - \pi_{pid,oid}(Includes))$

D
$$Q_1 = Product \ltimes_{pid=pid2} (\rho_{pid2 \leftarrow pid}(Includes))$$
$$Q_{ans} = \pi_{name}(\sigma_{pid2 \text{ IS NULL} \ \vee \ pid \text{ IS NULL}}(Q_1))$$

E None of the above.

(c) Which of the following Relational Algebra expressions do **NOT** return the name of all products that the customer with $cid = 123$ has rated 3 or better? Select all that apply.

A $Q_{ans} = \pi_{name}(\sigma_{cid=123}(\pi_{cid}(Customer)) \bowtie (\sigma_{score \geq 3.0}(Rates)) \bowtie Products)$

B
$$Q_1 = \text{Customer} \times (\rho_{pid2 \leftarrow pid, cid2 \leftarrow cid}(\text{Rates})) \times (\rho_{pname \leftarrow name}(\text{Product}))$$
$$Q_{ans} = \pi_{pname}(\sigma_{cid=cid2 \ \wedge \ pid=pid2 \ \wedge \ score \geq 3.0 \ \wedge \ cid=123}(Q_1))$$

C
$$Q_1 = \sigma_{cid=123}(Customer \bowtie \pi_{pid,name}(Product))$$
$$Q_{ans} = \pi_{name}(\sigma_{score \geq 3.0}(Rates) \bowtie \pi_{cid,pid}(Q_1))$$

D
$$Q_1 = \pi_{cid}(\sigma_{cid=123}(Customer)) \times \pi_{pid,name}(Product)$$
$$Q_{ans} = \pi_{name}(\sigma_{score \geq 3.0}(Rates) \bowtie Q_1)$$

E None of the above.

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 (name)"`, and if your answer to this question is the query `"SELECT name FROM countries;"`, then you must enter your answer in the answer box as follows:

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

- 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 the name of all African countries with a population of more than 100,000,000 people!

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

2.2 Query 2 (2 Marks): For each continent, find the name of the continent with the corresponding number of countries in that continent which do **NOT** have any airport!

```
CREATE OR REPLACE VIEW q2 (continent, country_count) AS
-- your query goes here
;
```

2.3 Query 3 (2 Marks): Find the name of the top 10 countries with the most land borders and the number of land borders they have!

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

2.4 Query 4 (2 Marks): Find all pairs of name of countries that share a common land border where you can cross from Europe (`country_name1`) into Asia (`country_name2`)! For example Armenia (Europe) shares a land border with Iran (Asia), so ('Armenia', 'Iran') should be in the result.

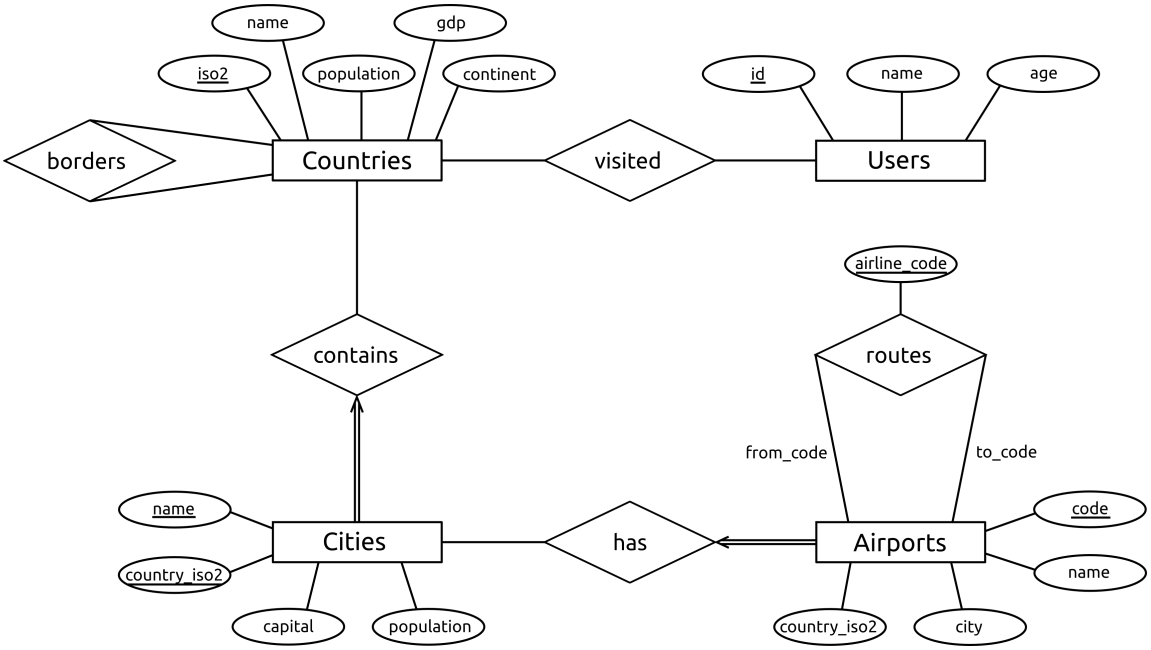
```
CREATE OR REPLACE VIEW q4 (country_name1, country_name2) AS
-- your query goes here
;
```

2.5 Query 5 (3 Marks): Find the names of all Asian countries to which Singapore Airlines (`airline_code='SQ'`) does not fly! For example, Singapore Airlines does not fly to Bhutan. Include the countries that do not have any airport. For example, Palestine does not have an airport, so Palestine should also be in the result.

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

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;
```