Databases

Björn Þór Jónsson

April 19, 2024

Instructions

You have 4 hours to answer 7 problems described in the following. The exam consists of 11 numbered pages. Unless instructed otherwise your answers must be provided in the Gradescope quiz *Final Exam 2024*, which is accessible from a Canvas assignment also named *Final Exam 2024*. Note that since the Gradescope quiz has 8 sub-questions for question 1, one question for each of questions 2–3 and 6–7, and sub-questions for questions 4–5, there are in total 20 questions in the quiz.

Database Description for Questions 1–3

In this exam you will work with a fictional (and poorly designed!) database of video games. To start working with the database, run the commands in `gag-april-2024.sql` found in the Canvas assignment using the PostgreSQL DBMS on your laptop. It is recommended to use `psql` for this purpose. The database has the following schema:

```
Country(CID, name, population)
Developer(DID, name, since, CID)
Game(GID, name, genre, DID)
Person (PID, name, birthyear)

Buys (GID, PID, year)
Plays (GID, PID)
```

Primary keys and foreign keys are defined and attributes are largely self-explanatory. You may study the DDL commands to understand the details of the tables (the CREATE TABLE statements are at the top of the script), consider the ER-diagram in Figure 1, or inspect the tables using SQL queries. Following are some additional notes that are important for your queries:

- The attribute `since` in the table `Developer` is the year the developer company was founded.
- The column `name` is a candidate key (`UNIQUE`) in the tables `Country`, `Developer` and `Game`, but not in the table `Person`.
- The database is randomly created and may contain strange data and errors. In particular, `Buys.year` and `Person.birthyear` are very inconsistent.

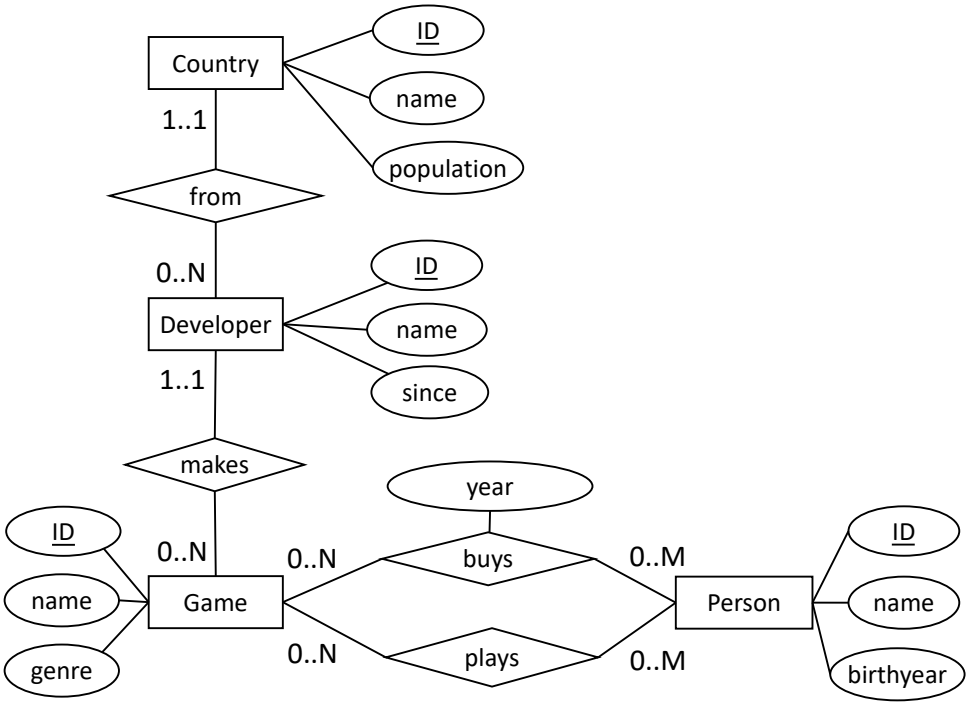

Figure 1: ER Diagram for the video games database.

Instructions for SQL Queries in Question 1

Queries must return correct results for any database instance. They should avoid system-specific features, including the LIMIT keyword. Queries should not return anything except the answer; a query that returns more information will not receive full points, even if the answer is part of the returned result. A sequence of several queries that answer the question will not receive full points, but subqueries and views can be used. Queries should be as simple as possible; queries that are unnecessarily complex may not get full marks, despite returning the correct answer. If you are unable to complete the query you can still submit your attempt, along with a brief description, and it may be given partial points.

1 SQL (40 points)

Answer each of the following questions using a single SQL query on the examination database. Submit each query in its respective box. If you use views, they must be defined with each query that uses the view. Queries must adhere to the detailed guidelines given on Page 3.

(a) The database has 136 developers from Japan. How many games have been made by developers from Japan?

(b) Person with ID = 60 has bought games from 4 different developers. How many different developers has the person with ID = 222 bought games from?

(c) 275 different developers are tied in the position of having developed the fewest games, that is only one each. What is the ID of the developer which has produced the most games?

Note: This query returns an identifier, not a count of result rows. In this database instance, there is a single developer that has made the most games, but your query must return all developers in case of ties.

(d) How many games have been bought fewer than 3 times?

(e) The game 'The Legend of Zelda' has been sold only once, according to the database. What is the name of the game which has been sold most often?

Note: This query returns a single short string. As before, your query should work equally well in case of ties.

(f) How many people are either born before 1980 or have bought a game in 2001 that they have played?

(g) There are 61 people who have bought some game from all countries that have a population of more than 300 000 000. How many people have bought some game from all countries that have a population smaller than 500 000?

Note: This is a division query; points will only be awarded if division is attempted.

(h) According to the database, how many different people have bought some game from South Korea before the developer was founded and before the person was born, but do not play that game?

2 SQL Programming (5 points)

Consider the SQL trigger code in Figure 2.

```
CREATE OR REPLACE FUNCTION NoNegativePopulation()
RETURNS TRIGGER
AS $$
BEGIN
    IF TG_OP = 'INSERT'
    THEN
        RAISE EXCEPTION 'Operation not allowed!'
        USING ERRCODE = '45000';
    END IF;
    IF NEW.population < 0
    THEN
        NEW.population = NULL;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER NoNegativePopulation
AFTER INSERT OR UPDATE ON Country
FOR EACH ROW EXECUTE PROCEDURE NoNegativePopulation();

INSERT INTO Country (CID, name, population)
VALUES (44, 'Morocco', 34924821);

UPDATE Country
SET population = -20
WHERE name = 'Japan';
```

Figure 2: Trigger `NoNegativePopulation` for the `Country` relation.

Select the true statements, based on the given database instance:

(a) The INSERT statement will fail.

(b) The UPDATE statement will change the population of Japan to -20.

(c) The trigger is correctly written as an AFTER trigger.

(d) NEW is not available in AFTER trigger.

3 Relational Algebra and Calculus (5 points)

Write the following SQL queries in both relational algebra and tuple relational calculus. Recall that both relational algebra and tuple relational calculus remove duplicates by default.

(a)
```
SELECT G.name
FROM Game G
WHERE G.genre = 'MMO'
  AND G.DID = 12;
```

(b)
```
SELECT DISTINCT C.name
FROM Country C
    JOIN Developer D on D.CID = C.CID
    JOIN Game G on G.DID = D.DID;
```

4 ER Diagrams and Normalization (30 points)

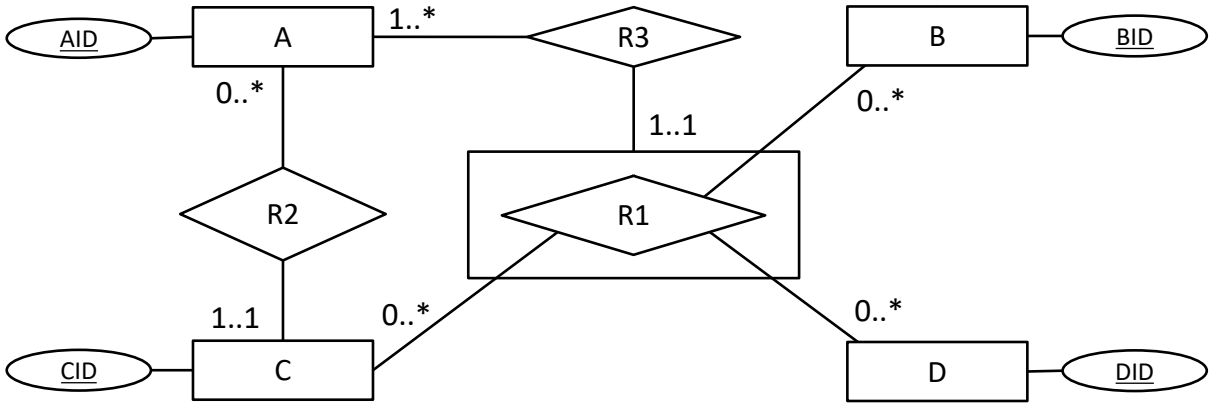

Figure 3: ER Diagram for Questions 4(a) and 4(b).

a) Consider the ER diagram in Figure 3. Select the statements that will be true for every instance of the database. You should base your answers **only** on the ER diagram:

(a) Each A is connected to exactly one C through relationship R2.

(b) Each B is connected to exactly one C through relationship R1.

(c) Each A is connected to exactly one D through relationships R3 and R1.

(d) Each D is connected to at least one A through relationships R1 and R3.

(e) Some A may be connected to itself through some relationships.

(f) The resulting database has 7 tables.

b) Write SQL DDL commands to create a database based on the ER diagram in Figure 3. The DDL script *must run* in PostgreSQL as a whole. The relations must include all primary key and foreign key constraints. Constraints that cannot be enforced with standard primary key and foreign key constraints should be omitted. The type of the ID attributes should be INT.

c) Write an ER diagram for a database of medieval times based on the following requirements. The diagram should clearly show the entities, relationships and participation constrains described below. Use the notation presented in the textbook and lectures. Assume that each entity has an ID, unless explicitly stated otherwise.

- Each Ruler is either a Despot or a Tyrant.
- Each Ruler has some Underlings. Each Underling belongs to a single Ruler.
- Tyrants may kill Underlings. Each Underling can obviously only be killed by one Tyrant.
- All Despots hate at least one Tyrant.
- Tyrants may have their personal Demons. Demons to not have an ID, but are only identified by the name given to them by their Tyrant.

d) Consider a table $R(L, M, N, O, P)$ with the following dependencies:

$$\begin{aligned} LM &\rightarrow NOP \\ N &\rightarrow L \\ O &\rightarrow P \end{aligned}$$

Select the true statements:

(a) NM is a (candidate) key of R.

(b) $L \rightarrow LM$ is a trivial functional dependency.

(c) Normalizing to 3NF or BCNF results in exactly two relations.

(d) The relation can be normalized to BCNF without losing dependencies.

e) Consider a table $R(L, M, N, O, P)$ with the following dependencies:

$$\begin{aligned} LM &\rightarrow NOP \\ L &\rightarrow O \\ N &\rightarrow P \end{aligned}$$

Normalize R to the highest possible normal form (3NF or BCNF), based on functional dependencies, while allowing all functional dependencies (excluding trivial, unavoidable, and redundant dependencies) to be checked within a single relation. For each resulting relation, write its columns and clearly indicate whether it is in BCNF.

5 Index Selection (10 points)

Consider the following large relation with 100 billion rows of social media comments:

Comments(postID, userID, text, URL, order)

For each of the queries below, select the index that a good query optimiser is most likely to use for the Comments table to process the query. Assume that all indexes are *unclustered* B+-trees. Also, assume that there are 1 billion distinct postID values, 20 million distinct userID values, 20 distinct order values, and that all text and URL attributes are long strings. For each query below, select the best index for the query, or select "no index" if a full table scan would yield better performance than any of the possible indexes.

(a) Comments(postID)

(b) Comments(userID)

(c) Comments(order)

(d) Comments(postID, text)

(e) No index

Query 1

```
select C.text
from Comments C
where C.postID = 2;
```

Query 2

```
select *
from Comments
where userID = 2543;
```

Query 3

```
select *
from Comments
where order = 1;
```

6 Python Database Programming (5 points)

Consider the Python code in Figure 4. Assume that the connection string has the correct database name, user name and password.

```python
import psycopg as pg
from psycopg.rows import dict_row

def game_avg_age_of_players(conn: pg.Connection, game: str):
    query = """
            SELECT p.birthyear
              FROM Game g
              JOIN Plays pl ON pl.gid = g.gid
              JOIN Person p ON pl.pid = p.pid
             WHERE g.name = %s
            """
    try:
        birthyears = conn.execute(query, [game])
        ages = [2024 - y['birthyear'] for y in birthyears.fetchmany(5)]
        return sum(ages) / len(ages)
    except Exception as error:
        print(error)

conn_string = "host=localhost dbname=[yourdb] user=[youruser] password=[yourpassword]"
with pg.connect(conn_string, autocommit=True, row_factory=dict_row) as conn:
    print(game_avg_age_of_players(conn, 'Super Mario'))
```

Figure 4: Python code for finding the average of players of a particular game.

Select the true statements, based on the given database instance:

(a) The function is safe against SQL Injections.

(b) The connection is safely closed.

(c) The calculated average will be correctly computed for any game.

(d) The average calculation can only be done in Python, not in standard SQL.

7 All The Rest (5 points)

Select the true statements:

(a) NoSQL systems are a very varied group of systems with different characteristics.

(b) Velocity is not an important V in Big Data.

(c) SSDs are better at random writes than HDDs.

(d) SQL injections are a critical security issue.