Solutions for Exercise no. 3

28.03.2017

2 SQL – DDL

Consider the following three relation instances: **table1**, **table2**, and **table3**. The attributes that make up primary keys are underlined. In **table2**, attribute A is a foreign key to attribute A in **table1**; and attributes {C,D} in **table2** form a foreign key to attributes {C,D} in **table3**.

table1

A	B
'a'	'b'

table2

A	C	D
'a'	'x'	1
'a'	'x'	2

table3

C	D	E
'x'	1	'w'
'x'	2	'z'

1. Write **CREATE TABLE** statements to create the three tables. Infer suitable data types from the example instances. Include definitions of primary keys and foreign keys.

```
CREATE TABLE table1 (
  A VARCHAR(10) PRIMARY KEY,
  B VARCHAR(10)
);

CREATE TABLE table3 (
  C VARCHAR(10),
  D INTEGER,
  E VARCHAR(10),
  PRIMARY KEY (C,D)
);

CREATE TABLE table2 (
  A VARCHAR(10),
  C VARCHAR(10),
  D INTEGER,
  PRIMARY KEY (A,C,D),
  FOREIGN KEY (A) REFERENCES table1(A),
  FOREIGN KEY (C,D) REFERENCES table3(C,D)
);
```

   Notice, the string attributes could have also other data types, e.g. **VARCHAR**(1), **CHAR**(1), and in Postgres also **VARCHAR** or **TEXT** work.

2. Does the order of the **CREATE TABLE** statements matter in your solution? Explain in *one* sentence!

   Yes, the order does matter. Table **table2** can only be created after **table1** and **table3** were created, because the definition of **table2** references the two other tables in its foreign key constraints.

3. Write **INSERT** statements to fill the tables with the example data.

```
INSERT INTO table1(A,B) VALUES ('a', 'b');
INSERT INTO table3(C,D,E) VALUES ('x',1,'w'), ('x',2,'z');
INSERT INTO table2(A,C,D) VALUES ('a','x',1), ('a','x',2);
```

4. Does the order of the **INSERT** statements matter in your solution? Explain in *one* sentence!

   Yes, the order does matter. The rows in **table2** can only be inserted after the other two tables were filled and the foreign key constraints can be satisfied.

5. Given the three tables filled with the data from the example and given the mentioned key constraints,

   (a) is it allowed to insert tuple ('a', 'x', 2) in **table2**? Explain in *one* sentence!

   No, this tuple already exists and the primary key prevents us from inserting a duplicate.

   (b) is it allowed to insert tuple ('b', 'x', 2) in **table2**? Explain in *one* sentence!

   No, the foreign key constraint on attribute A would be violated.

3 SQL – DML

For the following queries: (a) formulate and write down the query in SQL, (b) execute them on your Mondial database, and (c) write down the number of rows returned. Make sure your expressions are correct for all Mondial database instances.

1. The codes of all countries for which (a) the GDP is to at least 90% composed of the Service and Industry sectors together, or (b) the inflation is lower than 2%. In Mondial, percentages are stored as NUMERICs in the range from 0 to 100.

   ```
   SELECT Country
   FROM economy
   WHERE (Service + Industry) >= 90 OR Inflation < 2;
   ```

   The query returns 75 rows when executed on the Mondial database.

2. The names and codes of all countries that gained independence in 20th century (i.e. when the year of independence is in between 1901 and 2000, bounds included)

   ```
   SELECT Name, Code
   FROM country JOIN politics ON Code = Country
   WHERE EXTRACT(YEAR FROM Independence) BETWEEN 1901 AND 2000;
   ```

   The query returns 143 rows when executed on the Mondial database.

3. The names of all cities with non-unique names. Return each city name only once. No aggregation allowed.

```
SELECT DISTINCT c1.Name
FROM city c1 JOIN city c2 ON
     c1.Name = c2.Name AND (c1.Province <> c2.Province
     OR c1.Country <> c2.Country);
```

The query returns 57 rows when executed on the Mondial database.

4. The names of all provinces with at least one million inhabitants through which exactly five rivers flow.

```
SELECT g.Province
FROM geoRiver g JOIN province p ON
       g.Province = p.Name
       AND g.Country = p.Country
WHERE p.Population >= 1000000
GROUP BY g.Province, g.Country
HAVING COUNT(g.River) = 5;
```

The query returns 3 rows when executed on the Mondial database.

4 SQL – Understanding Queries

For each of the following queries explain *briefly* but *precisely* in natural language what they return *independently* of the actual data stored in the Mondial database. Include the result's size and order in your description. In a second step, *briefly* explain the difference between the queries.

1. The **LIMIT** 1 clause in the following query limits the number of result rows to at most 1. See also PostgreSQL's documentation[1].

```
SELECT Name, Area
FROM desert
ORDER BY Area DESC, Name ASC
LIMIT 1;
```

The query returns the name and area of the desert that has the largest area among all deserts and has the lexicographically smallest name.

2.
```
SELECT Name, Area
FROM desert
WHERE Area = (
  SELECT MAX(Area)
  FROM desert
)
ORDER BY Name ASC;
```

The query returns the names and areas of the deserts that have the largest area among all deserts, sorted lexicographically by name in ascending order.

[1] https://www.postgresql.org/docs/current/static/queries-limit.html

3.
```
SELECT d1.Name, d1.Area
FROM desert d1 JOIN (
  SELECT Area FROM desert
  EXCEPT
  SELECT d3.Area
  FROM desert d3 JOIN desert d4 ON d3.Area < d4.Area
) d2 ON d1.Area = d2.Area
ORDER BY d1.Name ASC;
```

This query is equivalent to the query in (2).

Difference

Queries (2) and (3) are equivalent. Their result differs from the result of query (1) if the database contains more than one largest desert. In this case, queries (2) and (3) return the names and areas of all largest deserts, ordered (lexicographically) by name, while query (1) returns only the name and area of the largest desert with the (lexicographically) smallest name.