Databases Lab 04

This is the final lab of Databases. This lab focuses on improving your skills in dealing with SQL queries. There are tasks for both DDL and DML statements. It is recommended that you familiarize yourself with the assignments to allow a more effective participation in Laboratory 4. If you have questions or need any support, help each other, or use the forum in our moodle room.

Contents

1. Assignment 1: Chemistry database

Download the script "chemistry.sql" from Moodle and execute the script in PostgreSQL.

1. Write a transaction to insert two new elements in table ELEMENTS and a new connection in table COMPOUNDS into the database. If an error occurs during one of the insertions, all changes should be reversed. In addition, the transaction should be logged in table CHANGELOG.

Solution

- Solution via trigger:

```SQL
create or replace function changelog_trigger() returns trigger as $$
declare
  action text;
  table_name text;
  transaction_id bigint;
  timestamp timestamp;
  old_data jsonb;
  new_data jsonb;
begin
  action := lower(TG_OP::text);
  table_name := TG_TABLE_NAME::text;
  transaction_id := txid_current();
  timestamp := current_timestamp;

  if TG_OP = 'DELETE' then
    old_data := to_json(OLD.*);
  elseif TG_OP = 'INSERT' then
    new_data := to_json(NEW.*);
  elseif TG_OP = 'UPDATE' then
    old_data := to_json(OLD.*);
    new_data := to_json(NEW.*);
  end if;

  insert into lab4_1.changelog (action, table_name, timestamp, old_data, new_data)
  values (action, table_name, timestamp, old_data, new_data);

  return null;
end;
$$ language plpgsql;
```

- Solution without trigger would just log changes as an insert in the Transaction itself.
- Insert into the db. Rollback is automatic.

> ### 🧩 Solution
>
> ```sql
> 1  BEGIN;
> 2  INSERT INTO lab4_1.elements( elementid, symbol, name, atomicnumber,
>    atomicweight)
> 3  VALUES( 11, 'He', 'Helium', 2, 4.009);
> 4  INSERT INTO lab4_1.elements( elementid, symbol, name, atomicnumber,
>    atomicweight)
> 5  VALUES( 12, 'Li', 'Lithium', 4, 8.109);
> 6  INSERT INTO lab4_1.compounds( compoundid, name, formula)
> 7  VALUES( 100, 'Propane', 'C2H6');
> 8  COMMIT;
> ```

2. Write a transaction to add
   - a new lab (ChemLab6 located at Building C, Room 101) into table LABORATORIES,
   - a new researcher (Sophia Neumann) assigned to this lab into table RESEARCHERS, and
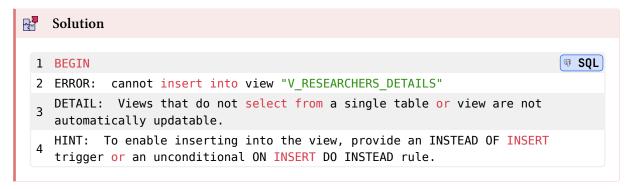   - two new reactions discovered by this new researcher into table REACTIONS.

If an insertion fails, all changes should be reversed. Additionally, log each change with a detailed description in table CHANGELOG.

Solution

```sql
1   BEGIN;
2   INSERT INTO lab4_1.laboratories(
3     labid,
4     labname,
5     location
6   )
7   VALUES
8   (
9   100,
10  'ChemLab6',
11    'Building C, Room 101'
12    );
13  INSERT INTO lab4_1.researchers (researcherid, firstname, lastname, labid)
14  VALUES (100, 'Sophia', 'Neumann', 100);
15
16  INSERT INTO lab4_1.reactions (reactionid, name, description)
17  VALUES (100, 'Redox', 'Reducing one side and increasing the other.');
18
19  INSERT INTO lab4_1.reactions (reactionid, name, description)
20  VALUES (101, 'Reverse Redox', 'Reverses all changes done by a redox
    reaction.');
21  COMMIT;
```

3. Create the view `V_RESEARCHERS_DETAILS` to display information about all researchers in detail, including their full name, the name of their lab and the location of the lab. The view should combine data from the tables RESEARCHERS and LABS.

Solution

```sql
1   CREATE OR REPLACE VIEW "V_RESEARCHERS_DETAILS" AS
2   SELECT
3     r.firstname,
4     r.lastname,
5     l.labname,
6     l.location
7   FROM
8     lab4_1.researchers r
9   LEFT JOIN
10    lab4_1.laboratories l ON r.labid = l.labid;
```

4. Try to insert, delete, and update tuples in the view `V_RESEARCHERS_DETAILS`. Which operations (`INSERT`, `DELETE`, and `UPDATE`) can be executed and which not? Explain your answer.

> **Solution**
>
> ```SQL
> 1  BEGIN
> 2  ERROR:  cannot insert into view "V_RESEARCHERS_DETAILS"
> 3  DETAIL:  Views that do not select from a single table or view are not
>    automatically updatable.
> 4  HINT:  To enable inserting into the view, provide an INSTEAD OF INSERT
>    trigger or an unconditional ON INSERT DO INSTEAD rule.
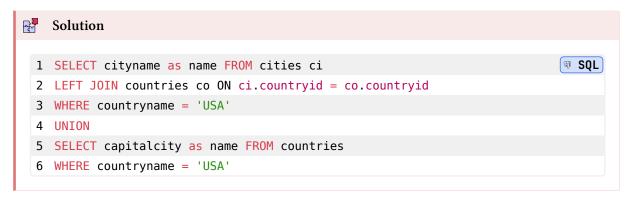> ```

2. Assignment 2: Geography database

Download the script "geography.sql" from Moodle and execute the script in postgreSQL. Write SQL queries for the following questions.

1. What is the capital of Germany?

> **Solution**
>
> ```SQL
> 1  SELECT CAPITALCITY FROM COUNTRIES WHERE countryname = 'Germany';
> ```

2. Write an SQL query to list all cities in the USA.

> **Solution**
>
> ```SQL
> 1  SELECT cityname as name FROM cities ci
> 2  LEFT JOIN countries co ON ci.countryid = co.countryid
> 3  WHERE countryname = 'USA'
> 4  UNION
> 5  SELECT capitalcity as name FROM countries
> 6  WHERE countryname = 'USA'
> ```

3. Write an SQL query to find the capitals and populations of all countries with names beginning with the letter "C".

> **Solution**
>
> ```SQL
> 1  SELECT capitalcity, population FROM countries
> 2  WHERE countryname like 'C%';
> ```

4. List all rivers that are longer than 4000 km.

> **Solution**
>
> ```sql
> 1  SELECT rivername FROM rivers
> 2  WHERE length > 4000;
> ```

5. Identify the highest mountains in descending order of height.

> **Solution**
>
> ```sql
> 1  SELECT mountainname FROM mountains
> 2  order by height Desc;
> ```

6. List all cities with a population over 5 million in descending order of population.

> **Solution**
>
> ```sql
> 1  SELECT cityname FROM cities
> 2  WHERE population > 5000000
> 3  order by population desc;
> ```

7. Add a new language called "Swahili" and a new country "Kenya" with CountryID 19. Then, link the language and the country in table COUNTRYLANGUAGES. Write a query displaying all information about the country "Kenya" and the language "Swahili" that checks the completeness of the data.

> **Solution**
>
> ```sql
>  1  INSERT INTO languages (languageid, languagename)
>  2  VALUES (19, 'Swahili');
>  3
>  4  INSERT INTO countries
>  5  (countryid, countryname, population, capitalcity)
>  6  VALUES (19, 'Kenya', 10000000, 'Nairobi');
>  7
>  8  INSERT INTO countrylanguages (countryid, languageid)
>  9  VALUES (19, 19);
> 10
> 11  SELECT * FROM countries co
> 12  LEFT JOIN countrylanguages cl on cl.countryid = co.countryid
> 13  LEFT JOIN languages l on cl.languageid = l.languageid
> 14  WHERE countryname = 'Kenya';
> ```

8. Create the view `V_LARGEST_CITY_AND_HIGHEST_MOUNTAIN` that shows for each

country the name of the country, the name of the most populated city, the population of this city, the name of the highest mountain and the height of this mountain.

> **Solution**
>
> ```SQL
> CREATE OR REPLACE VIEW "V_LARGEST_CITY_AND_HIGHEST_MOUNTAIN" AS
> SELECT
>     co.countryname,
>     ci.cityname,
>     MAX(ci.population) max_population,
>     m.mountainname,
>     MAX(m.height) max_height
> FROM countries co
> LEFT JOIN cities ci ON ci.countryid = co.countryid
> LEFT JOIN mountains m ON m.countryid = co.countryid
> GROUP BY countryname, cityname, mountainname;
> ```

9. Try to insert, delete, and update tuples in the view

`V_LARGEST_CITY_AND_HIGHEST_MOUNTAIN`. Which operations (`INSERT`, `DELETE`, and `UPDATE`) can be executed and which not? Explain your answer.

> **Solution**
>
> - All three operations will not work, because we are A using Aggregate Functions and B are using multiple base tables. The only thing that would work are `INSTEAD OF INSERT ON` Trigger, that could handle insertion to base tables.

3. Assignment 3: SQL-statements for the COMPANY example from Elmasri also used in the lecture

Let's revisit the company example.

1. Create a SQL statement which creates the view `V_PROJECT1`. View `V_PROJECT1` has the project name, controlling department name, number of employees, and total hours worked per week on the project for each project.

Solution

```sql
1   CREATE OR REPLACE VIEW "V_PROJECT1" AS
2   SELECT
3     p.projectname,
4     d.deptname,
5     COUNT(e.empname) as empnum,
6     SUM(wo.hours) as hoursnum
7   FROM projects p
8   LEFT JOIN works_on wo ON wo.projectid = p.projectid
9   LEFT JOIN emp e ON e.empid = wo.empid
10  LEFT JOIN dept d ON d.deptid = e.deptid
11  GROUP BY projectname, deptname;
```

2. Create a SQL statement which creates the view V_PROJECT2. View V_PROJECT2 that has the project name, controlling department name, number of employees, and total hours worked per week on the project for each project with more than one employee working on it.

Solution

```sql
1   CREATE OR REPLACE VIEW "V_PROJECT1" AS
2   SELECT
3     p.projectname,
4     d.deptname,
5     COUNT(e.empname) as empnum,
6     SUM(wo.hours) as hoursnum
7   FROM projects p
8   LEFT JOIN works_on wo ON wo.projectid = p.projectid
9   LEFT JOIN emp e ON e.empid = wo.empid
10  LEFT JOIN dept d ON d.deptid = e.deptid
11  HAVING COUNT(empname) > 1
12  GROUP BY projectname, deptname;
```

3. Assume the view V_SENIORS with following code is given.

```sql
1 CREATE VIEW V_SENIORS AS
2 SELECT *
3 FROM employee
4 WHERE salary > 45000
5 WITH CHECK OPTION;
```

- Explain the purpose of views in general and the function of the view V_SENIORS.
- Is it possible to modify the tuples in view V_SENIORS?
  ‣ First think about it in theory.

‣ Second try it out in a database:
- Update the salary of one employee in view `V_SENIORS` and in base

table `EMPLOYEE`.
- Insert some new employees in view `V_SENIORS` and in base table

`EMPLOYEE`.
- Delete some tuples in view `V_SENIORS` and in base table `EMPLOYEE`.

> **Solution**
>
> - The View shows all employees that have an annual salary over 45000
> - It is possible, since the view does not include group by or multiple base tables
>   - ‣ However, the view does not allow updates to rows, where `salary < 45000`

4. Assignment 4: Transactions

For this assignment, you need two distinct sessions (connections) to your database. In PostgreSQL's pgAdmin you can just open two instances of either the CLI `psql` or the query tool, which will act as independent session. The two database sessions are denoted as `session1` and `session2`, resp. For every task include your statements and the corresponding output in `session1` and `session2` in a table. Make sure everything is in chronological order. Example:

Timestamp	Session 1	Session 2

0	`1  INSERT INTO TAB1...;` 🔲 **SQL**
	
	    • Result screenshot
	
	`1  SELECT * FROM TAB1;` 🔲 **SQL**
	
	    • Result screenshot
		-

1		`1  SELECT * FROM TAB1;` 🔲 **SQL**

Table 1: Sessions

1. In `session1` run a command creating a table named `TAB1` with two attributes:
   - `id` with data type `integer`, `primary key`
   - `n` with data type `integer`

   When is table `TAB1` visible in `session2`?

> **Solution**
>
> - The table is visible immediately after completing the `CREATE TABLE`-statement.

2. In `session1` insert the following tuples into `TAB1` within one transaction:

```
1 (1,1), (2,2), (3,3)
```

What content of TAB1 is displayed in session2 before and after you commit your changes in session1?

> **⊞ Solution**
>
> - Before the commit, the changes are only visible to session1. After the commit, the changes are visible to both sessions.

3. In session1 update the value of n to 33 for the tuple with id 3 (without committing). Afterwards rollback that transaction.

- What value of n (id=3) is displayed in session1 before and after the rollback?
- What value of n (id=3) is displayed in session2 before and after the rollback?

> **⊞ Solution**
>
> - Before the rollback, session1 shows n = 33, while session2 shows n = 3.
> - After the rollback, session1 shows n = 3, while session2 shows n = 3.