Fall 2022 B461 Assignment 3
Triggers, Queries with Quantifiers

Keerthana, Srilekha, Muazzam Siddiqui

Released: 23rd September 2022
Due: 06th October 2022

1 Introduction

This assignment covers:

1. Triggers
2. Queries with Quantifiers

To turn in your assignment, you will need to upload to Canvas the following files:

- `assignment3.sql`
- `assignment3.txt`
- `assignment3.pdf`

The `assignment3.sql` contains the necessary SQL statements that solve the problems in this assignment. The `assignment3.sql` file must be such that the AI's can run it in their PostgreSQL environment.

The `assignment3.txt` file contains the results of running your queries.

The `assignment3.pdf` file to show your Venn Diagrams

Grading Rubric (100 pts total)

1. `10 pts` if the query returns expected results.
2. `0 - 9 pts` for incorrect results, the deduction of points will be gauged on how logically sound the query is.
3. Out of 10 questions, 3 will be randomly selected and full credit be will awarded for attempting them. **Note: the attempt should be relevant to the question and non-trivial.**

For the problems in this assignment we will use the following database schema:[1]

```
Westerosi(wid, wname, wlocation)
House(hname, kingdom)
Skill(skill)
OfHouse(wid, hname, wages)
HouseAllyRegion(hname, region)
WesterosiSkill(wid, skill)
Predecessor(succid, predid)
Knows(wid1, wid2)
```

In this database[2] we maintain a set of Westerosis[3] (`Westerosi`), a set of Houses (`House`), and a set of skills (`Skill`). The `wname` attribute in `Westerosi` is the name of the resident of Westeros.

The `wlocation` attribute in `Westerosi` specifies the area in which the person is currently stationed. The `hname` attribute in `House` is the name of a House in Westeros.

The `kingdom` attribute in `House` is the name of the location wherein the lord of the house resides. The `skill` attribute in `Skill` is the name of a skill possessed by Westerosis.

A Westerosi can be of at most one House. This information is maintained in the `OfHouse` relation. (We permit that a Westerosi does not belong to any House.) The `wages` attribute in `OfHouse` specifies the wages made by the Westerosi.

The `region` attribute in `HouseAllyRegion` indicates a region in which the house has allies. (Houses may have allies in multiple regions.)

A Westerosi can have multiple skills. This information is maintained in the `WesterosiSkill` relation. A skill can be the skill of multiple Westerosis. (A Westerosi may not have any skills, and a skill may have no Westerosis with that skill.)

A pair (s, p) in `Predecessor` indicates that a Westerosi (successor) s has a Westerosi p as one of his or her predecessors. We permit that a successor has multiple predecessors and that a predecessor may be succeeded by multiple successors. (It is possible that a Westerosi has no predecessor and that a Westerosi is not a predecessor.) We further require that a Westerosi and his or her predecessors must belong to the same House.

The relation `Knows` maintains a set of pairs (w_1, w_2) where w_1 and w_2 are wids of Westerosis. The pair (w_1, w_2) indicates that the person with wid w_1

[1] The primary key, which may consist of one or more attributes, of each of these relations is underlined.

[2] The values of the database are inspired by a popular series - Game of Thrones just to make the course a little fun. We in no way bear responsibility for any spoilers or faults in the storyline/theories based on these values. So kindly humor us and have just as fun with making the queries as we do in asking for them!

[3] Residents of Westeros

knows the person with wid w_2. We do not assume that the relation `Knows` is symmetric: it is possible that (w_1, w_2) is in the relation but that (w_2, w_1) is not.

The domain for the attributes `wid`, `wages`, `succid`, and `predid` is `integer`. The domain for all other attributes is `text`.

We assume the following foreign key constraints:

- `wid` is a foreign key in `OfHouse` referencing the primary key `wid` in `Westerosi`;
- `hname` is a foreign key in `OfHouse` referencing the primary key `hname` in `House`;
- `hname` is a foreign key in `HouseAllyRegion` referencing the primary key `hname` in `House`;
- `wid` is a foreign key in `WesterosiSkill` referencing the primary key `wid` in `Westerosi`;
- `skill` is a foreign key in `WesterosiSkill` referencing the primary key `skill` in `Skill`;
- `succid` is a foreign key in `Predecessor` referencing the primary key `wid` in `Westerosi`; and
- `predid` is a foreign key in `Predecessor` referencing the primary key `wid` in `Westerosi`;
- `wid1` is a foreign key in `Knows` referencing the primary key `wid` in `Westerosi`; and
- `wid2` is a foreign key in `Knows` referencing the primary key `wid` in `Westerosi`

The file `assignment3.sql` contains the data supplied for this assignment.

2 Triggers

To begin the problems in this section, you should first remove the en- tire database, including the relation schema. You should create the relations without specifying the primary and foreign key constraints. You should also not yet populate the relations with data. Solve the following problems:

1. Develop appropriate insert and delete triggers that implement the primary key and foreign key constraints that are specified for the `Westerosi, House,` and `ofHouse` relations.
   Your triggers should report appropriate error conditions. For this problem, implement the triggers such that foreign key constraints are maintained using the cascading delete semantics.
   For a reference on cascading deletes associated with foreign keys maintenance consult the PostgreSQL manual page
   `https://www.postgresql.orgdocs9.2ddl-constraints.html`

   Test your triggers using appropriate inserts and deletes.

2. Consider two relations R(A:integer,B:integer) and S(B:integer) and a view with the following definition:

   ```
   select distinct r.A
   from R r, S s
   where r.A > 10 and r.B = s.B;
   ```

   Suppose we want to maintain this view as a materialized view called V(A:integer) upon the insertion of tuples in R and in S. (You do not have to consider deletions in this question.) Define SQL insert triggers and their associated trigger functions on the relations R and S that implement this materialized view. Write your trigger functions in the language 'plpgsql.' Make sure that your trigger functions act in an incremental way and that no duplicates appear in the materialized view.

3. Consider the following on the WesterosiSkill relation. "Each skill of a Westerosi who belongs to house `Lannister` must also be a skill of a Westerosi who belongs to the house `NightsWatch`."
   Write a trigger that maintains this constraint when inserting pairs $(wid, skill)$ into the `WesterosiSkill` table. If constraint is violated raise exception with appropriate error messages.

4. Consider the view WesterosiHasSkills(wid ,skills) which associates with each Westerosi, identified by a wid, his or her set of skills.

```
Create view WesterosiHasSkills as
        select distinct W.wid from
        Westerosi W, WesterosiSkill WS
        where W.wid = WS.wid
        order by 1;
```

   Write a trigger that will delete all the skill records from $WesterosiSkill$ relation when a Westerosi entry(wid) is deleted from the above $WesterosiHasSkills$ view. Show appropriate delete statements.

3 Queries with quantifiers

Using the method of Venn diagrams with conditions (Show these venn diagrams with conditions in pdf file) and write SQL queries for the following queries with quantifiers.

To get full credit in these problems, you must write appropriate views, parameterized views and functions for the sets A and B that occur in the Venn diagram with conditions. Show these venn diagrams with conditions in pdf file for these queries. (Query + Venn Diagram: 10 points for each problem).

Hint: You can create views, functions and then use them in your query to find the answer.

Make the following queries **without using the COUNT function**:

5. Find the hname of each house who only has Westerosis currently stationed in 'Winterfell' or 'CastleBlack' location.

6. Find the wid of each Westerosi who knows all Westerosis who belongs to 'Lannister' house and makes wages that is equal to 55000.

7. Find the pairs `(s1, s2)` of different Successors(Westerosis) such that some Predecessors of Successor1 are Predecessors of Successor2.

   Make the following queries **with using the COUNT function** and other aggregation functions if required:

8. Find the hname of each house that not only has Westerosis with 'Politics' in their skills set.

9. Find the pairs `(wid1, wid2)` of different Westerosis such that Westerosi with wid1 and the Westerosi with wid2 knows same number of Westerosis.

10. Find the hname of each house that has strictly 2 Westerosis with minimum wages at that house and knows at least 3 Westerosis.