Fall 2022 B461 Assignment 2
Views, Functions, Expressions and Set Predicates

Srinivas Kini, Aravind Reddy Sheru, Muazzam Siddiqui

Released: Sep 9, 2022
Due: September 22, 2022

1 Introduction

The goals for this assignment are to

1. Formulate queries using set predicates.
2. Define functions that can aid in computing results in larger queries.
3. Build familiarity with SQL Views.

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

- `assignment2.sql`
- `assignment2.txt`

The `assignment2.sql` contains the necessary SQL statements that solve the problems in this assignment. The `assignment2.sql` file must be such that the AI's can run it in their PostgreSQL environment.

The `assignment2.txt` file contains the results of running your queries.

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`

We define 2 more relations:

- `Pizza(diameter int)`
- `LinkedList(node int, nextNode int)`

That are unrelated to the schema mentioned above, but are used in separate questions.

The file `a2data.sql` contains the data supplied for this assignment.

2 Expressions & Functions

Tip: To make things simpler, make use of functions for intermediate calculations and break the problem down into smaller pieces.

1. Consider the relation `Equation(a int, b int, c int)`, write an sql query to find the roots r1,r2 of the the quadratic equation: $ax^2 + bx + c$
2. Using the relation `Pizza`, write a SQL query that calculates the ratio of the areas of the 2 largest Pizzas (largest / second largest) rounded to 2 decimal places. ($\pi = 3.14$)
3. Create a function `skillsInRange(n1 int, n2 int)` returns the count of Westerosis that have at least `n1` skills and at most `n2` skills. Test your queries with inputs:
   - `skillsInRange(0, 1)`
   - `skillsInRange(4, 5)`
4. Create a function/parameterized view `familyGuy(housename)` that takes an `hname` as input, and returns the `wids` of the Westerosi with the most amount of immediate successors. Test your query with inputs:
   - `familyGuy('Stark')`
   - `familyGuy('Baratheon')`

3 SQL with Set Predicates

5. Find the wid of the Westerosi who don't have 'Archery' or 'Swordsmanship' as their skill.

   (a) Formulate this query in Pure SQL by only using the `EXISTS` or `NOT EXISTS` set predicates. You can not use the set operations `INTERSECT`, `UNION`, and `EXCEPT`.

   (b) Formulate this query in Pure SQL by only using the `IN`, `NOT IN`, `SOME`, or `ALL` set membership predicates. You can not use the set operations `INTERSECT`, `UNION`, and `EXCEPT`.

6. Find all pairs `(h1,h2)` of `hnames` of different houses such that h1 and h2 do no have any westerosis belonging to the same `wlocation`.

   (a) Formulate this query in Pure SQL by only using the `EXISTS` or `NOT EXISTS` set predicates. You can not use the set operations `INTERSECT`, `UNION`, and `EXCEPT`.

   (b) Formulate this query in Pure SQL by only using the `IN`, `NOT IN`, `SOME`, or `ALL` set membership predicates. You can not use the set operations `INTERSECT`, `UNION`, and `EXCEPT`.

   (c) Formulate this query in Pure SQL without using set predicates. You can use the set operations INTERSECT, UNION, and EXCEPT.

7. Find the `(wid,hname)` of all the westerosis who belong to a `hname` and know at least 2 people belonging to the same house.

   (a) Formulate this query in Pure SQL by only using the `EXISTS` or `NOT EXISTS` set predicates. You can not use the set operations `INTERSECT`, `UNION`, and `EXCEPT`.

   (b) Formulate this query in Pure SQL by only using the `IN`, `NOT IN`, `SOME`, or `ALL` set membership predicates. You can not use the set operations `INTERSECT`, `UNION`, and `EXCEPT`.

   (c) Formulate this query in Pure SQL without using set predicates. You can use the set operations INTERSECT, UNION, and EXCEPT.

4 Views

8. Use a VIEW to return the pairs `(wid, wname)` of all Westerosis that earn a wage strictly greater than their immediate predecessors.

9. Define a materialized view `HouseLeader` that, for each `hname`, returns the `wid` of Westerosis known by atleast one Westerosis from the same `region`.

10. Let `LinkedList(node integer, nextNode integer)` be a binary relation, where a pair (n, m) in `LinkedList` indicates that node n is succeeded by node m. The `SequentialOrder(node integer)` view is inductively defined using the following two rules:

    - If n is `NULL`, m is a node in `SequentialOrder`, and represents the **head** of the `LinkedList` relation. (**Base rule**)
    - If s is a node in `SequentialOrder` and (n, m) is a pair in `LinkedList` such that $s = n$, it implies that m succeeds s in the order. If m is `NULL`, n is the last node in `LinkedList`. (**Inductive Rule**)

    Write a recursive view `SequentialOrder(node integer)` that starts at the **head** and visits each node in `LinkedList` in sequential order. You may assume each node in `LinkedList` is unique. Test your view with the data in the `a2data.sql file`.