Assignment 1: SQL and Prolog

Part 1: SQL

In this assignment, you will work with a university database to retrieve and analyze data using SQL queries.

Submission Instructions:

Please submit the following items on Canvas:

1. A text file containing your SQL queries.
2. A screenshot showing the output of your queries when executed in MySQL. You may put all the files in a single doc file for submission.

Exercises

Exercise 0: Setting Up MySQL and Populating Data

Instructions: 1) Install MySQL if it is not already installed. Ensure the MySQL server is running. 2) Log in to MySQL: `mysql -u root -p` 3) Create and populate the database using the following commands:

```
CREATE DATABASE university_db;
USE university_db;

SOURCE <DDL.sql>;
# replace <DDL.sql> with the actual file path.
# If you are using MySQL Workbench, you can open the script file and execute it.

SOURCE <smallRelationsInsertFile.sql>
# replace .SQL file with the actual file path.
```

Both sql scripts can be found at Canvas.

Exercise 1: SQL Qeuries

1. (5 pts) Find the names of all the instructors from the Biology department. Hint: Use the `instructor` table.
2. (5 pts) Find the names of courses in the Computer science department which have 3 credits. Hint: Use the `course` table.
3. (5 pts) For the student with ID 76543, show the course id and the title of all courses registered for by the student. Hint: Use the `course` and `takes` tables.
4. (5 pts) Retrieve the departments that have more than two instructors (not including two). Hint: Use the `instructor` table.

Part 2: Prolog

In this assignment, you will model a family tree in Prolog and define relationships such as parent, sibling, cousin, and grandparent. You will use facts and rules to represent these relationships and allow Prolog to infer new ones through queries.

Submission Instructions:

Please submit the following items on Canvas:

1. The Prolog source file (family.pl) containing your defined facts and rules (including advanced predicates).
2. A text file containing the queries you wrote to test the relationships.
3. A screenshot showing the output of your queries when executed in Prolog.

Objectives:

- Define basic family relationships using facts.
- Write rules to infer more complex relationships.
- Use Prolog's query mechanism to test your rules and answer questions about family relationships.

Problem Description:

You are provided with a family tree. You are asked to represent the tree in Prolog using facts and define rules to deduce more complex relationships. The family members' ages are included, and you need to define rules that use these ages to infer who is the "older" sibling.

Family Tree:

```
                       Robert (68) -m- Linda (65)
                                    |
  +--------------------------+------+-------------------+
  |                          |                          |
Alice (42) -m- Bob (44)  Michael (41) -m- Sarah (42)  Mark (45) -m- Chloe (43)
            |                          |                         |
      +-----+-----+              +-----+------+          +-------+------+
      |           |              |            |          |              |
  Jake (15)   Emma (12)       Nancy (20)    Tom (10)  John (20)      Lisa (17)
                                  |
                              +---+---+
                              |       |
                          Peter (2)  Amy (1)
```

The family tree is structured as follows, where "-m-" denotes a marital relationship.

- Robert and Linda are married.

- They have three children: Alice, Michael, and Mark.
- Alice is married to Bob, and they have two children: Jake and Emma.
- Michael is married to Sarah, and they have two children: Nancy and Tom. Nancy has two children: Peter and Amy.
- Mark is married to Chloe, and they have two children: John and Lisa.
- Family members' ages are included.

IMPORTANT NOTICE: To avoid duplicates (or infinite loops) in the results, please minimize redundant facts. For example, if you define that Robert and Linda are married and that Linda is the mother of Alice, you do not need to explicitly state that Robert is the father of Alice, which can be infered by the rules. Similary, if you have defined the fact of `married(Hasband, Wife)`, you do not need to add `married(Wife, Husband)` separately.

This is importnat to keep this family tree as simple as possible.

Please try your best to eliminate duplicates; however, a limited number of duplicates are allowed without penalty.

Exercises

Exercise 1: Defining Facts

Create Prolog facts to represent the following:

1. (2 pts) mother(Mother, Child) - A fact that represents a mother-child relationship.
2. (2 pts) married(Hasband, Wife) - A fact to represent a marriage.
3. (2 pts) male(Person) - A fact to indicate that a person is male.
4. (2 pts) female(Person) - A fact to indicate that a person is female.
5. (2 pts) age(Person, Age) - A fact to represent the age of a person.

Exercise 2: Defining Rules

Write rules to define complex relationships. Use recursion and Prolog's declarative approach to represent these relationships.

1. (2 pts) Father Relationship:

   Define a rule `father(Father, Child)` to infer the father of a child.

2. (2 pts) Parent Relationship:

   Define a rule `parent(Parent, Child)` to infer the parent of a child.

3. (2 pts) Sibling Relationship:

   Define a rule `sibling(Person1, Person2)` to determine if two people are siblings. Siblings share at least one parent.

4. (2 pts) Grandparent Relationship:

Define a rule `grandparent(Grandparent, Grandchild)` that identifies grandparents.

5. (2 pts) Cousin Relationship:

   Define a rule `cousin(Person1, Person2)` for cousin relationships. Cousins have parents who are siblings.

6. (2 pts) Uncle/Aunt Relationship:

   Define rules `uncle(Uncle, Child)` and `aunt(Aunt, Child)` to infer whether someone is an uncle or aunt. Uncles and aunts are the siblings of a parent.

7. (2 pts) Ancestor Relationship:

   Define a *recursive* rule `ancestor(Ancestor, Descendant)` to determine if a person is an ancestor of another.

8. (2 pts) Older Sibling Relationship:

   Define a rule `older_sibling(Older, Younger)` to determine if one sibling is older than another.

9. (4 pts) Bigger Brother/Bigger Sister:

   Define rules `bigger_brother(Brother, Sibling)` and `bigger_sister(Sister, Sibling)` that combine gender and age to infer whether a brother or sister is older than a sibling.

Exercise 3: Queries

Once you've defined the facts and rules, write Prolog queries to answer the following questions about the family tree:

1. (1 pt) Who are the siblings of Alice?
2. (1 pt) Who are the cousins of Jake?
3. (1 pt) Who are the grandparents of Peter?
4. (1 pt) Who are the uncles of Emma?
5. (1 pt) Is Robert an ancestor of Amy?
6. (1 pt) Who is the uncle of Peter?
7. (1 pt) List all descendants of Robert.
8. (1 pt) Who is the bigger brother of Emma?
9. (1 pt) Who is the older sibling of Tom?
10. (1 pt) Who are the children of Linda?

Exercise 4: Advanced predicates

Implement the following predicates and test them with appropriate queries.

1. (5 pts) Brother-in-law/Sister-in-law Relationship:

Please implement predicates `brother_in_law(BrotherInLaw, Person)` and `sister_in_law(SisterInLaw, Person)` to represent relationships where someone is a sibling of a spouse or married to a sibling.

2. (5 pts) “nth” Ancestor of a Person:

   Please implement a predciat to find an ancestor at a specific generation level, where n is the number of generations between the person and their ancestor (e.g., the nth ancestor of someone could be their parent, grandparent, great-grandparent, etc.).

   Hint: You may write a recursion program.

   ```
   nth_ancestor(N, Ancestor, Person).
   ```

   Example Query:

   ```
   ?- nth_ancestor(2, Ancestor, tom).
   ```

   Expected Output: Based on the family tree, Tom’s 2nd ancestor (grandparent) is:

   ```
   Ancestor = robert ;
   Ancestor = linda ;
   ```

3. (5 pts) Common Ancestor:

   Define a rule common_ancestor(Person1, Person2, Ancestor) that identifies the common ancestor between two people.

   Example Query:

   ```
   ?- common_ancestor(peter, emma, Ancestor).
   ```

   Expected Output:

   ```
   Ancestor = robert ;
   Ancestor = linda.
   ```

4. (5 pts) Common Ancestor List:

   Create a predicate common_ancestors(Person1, Person2, AncestorList) that returns a list of all common ancestors between two people.

   Hint: You may want to use a predicate function `findall`.

   Example Query:

   ```
   ?- common_ancestor(peter, emma, Ancestors).
   ```

   Expected Output:

   ```
   Ancestors = [robert, linda].
   ```