University of Asia Pacific

Department of Computer Science and Engineering

Technical Report

Course Code: CSE 404
Course Title: Artificial Intelligence and Expert Systems Lab

Assignment 1: Implement a Basic Knowledgebase of Your Choice Using Prolog.

Submitted By:
Md Kayes Ahammed Biplob
Registration No: 22101151
Section: D

Submitted To:
Bidita Sarkar Diba
Lecturer
Dept. of Computer Science and Engineering

Problem Title:

Mythology Knowledgebase in Prolog

A Prolog-based knowledge representation system for storing and querying facts and relationships about gods, mortals, heroes, and creatures in Greek mythology.

Problem Description:

The **Mythology Knowledgebase** is designed to represent and explore relationships among mythological figures using Prolog.
It stores:

- **Facts**: gods, mortals, creatures, heroes, domains, and parent-child relationships.
- **Rules**: derived relationships such as siblings, ancestors, descendants, demigods, and hero-creature encounters.

The system can answer queries such as:

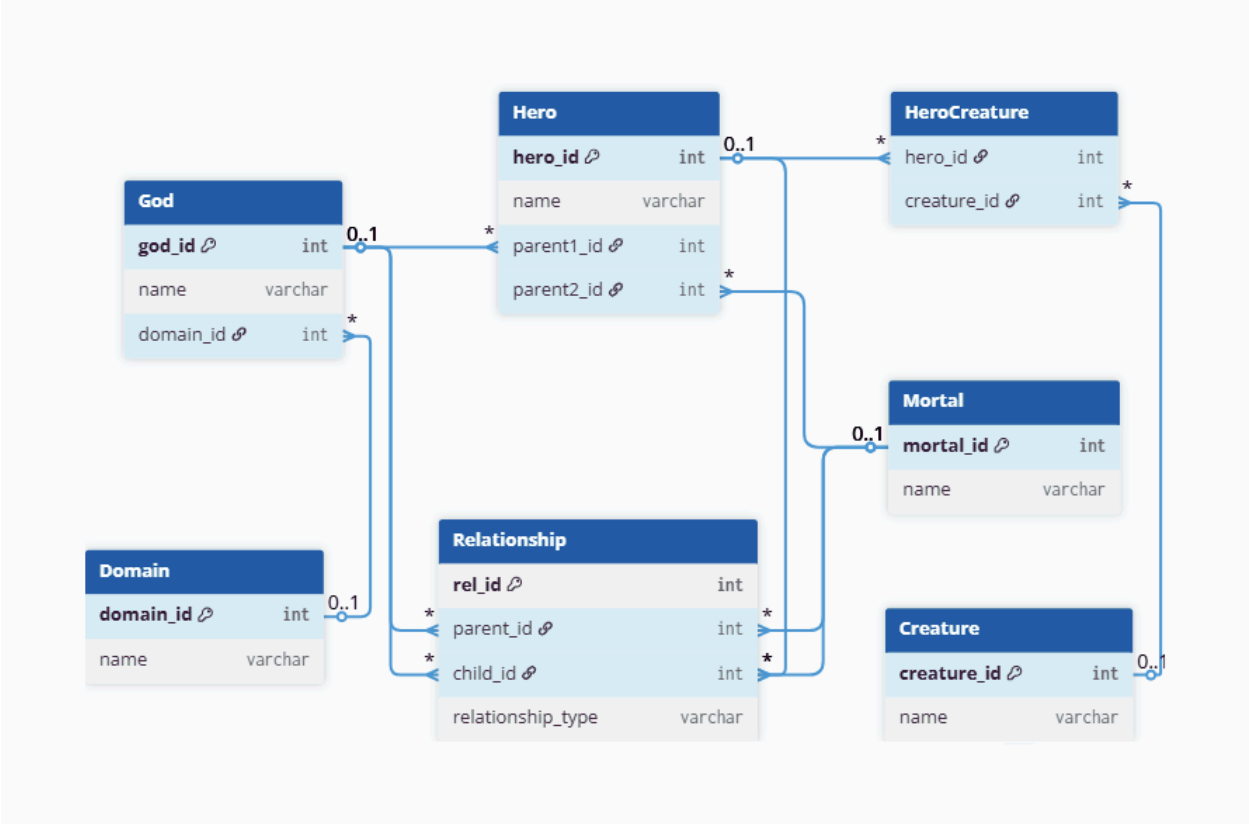
- Who are the children of Zeus?
- Is Heracles a demigod?
- Which heroes defeated which creatures?
- Who are the siblings of Apollo?
- What domain does Athena belong to?
- Who are the ancestors of Perseus?

Recursion is applied for **ancestor** and **descendant** rules, enabling multi-generational queries.

Tools and Languages Used:

1. **Programming Language:** Prolog (SWI-Prolog)
2. **Development Tool:** SWI-Prolog Editor
3. **Diagram Tools:** draw.io / dbdiagram.io for ER Diagram, DALL·E for visualization
4. **OS:** Windows 10

Diagram / Figure:



ER Diagram of Mythology Knowledgebase

The diagram models the **Mythology Knowledgebase** with six main entities:

1. **God**
    - **Attributes**:
        - `god_id` (**Primary Key**) – unique identifier for each god.
        - `name` – name of the god (e.g., Zeus, Poseidon).
        - `domain_id` (**Foreign Key**) – links the god to a **Domain** (e.g., sky, sea, underworld).
    - **Relationships**:
        - Connected to **Domain** (many gods can belong to one domain).
        - Linked to **Relationship** as a possible parent of other gods, heroes, or mortals.
2. **Mortal**
    - **Attributes**:
        - `mortal_id` (**Primary Key**) – unique identifier for each mortal.
        - `name` – name of the mortal (e.g., Alcmene, Danae).
    - **Relationships**:
        - Can be a parent to a hero through **Relationship** or directly via `Hero.parent2_id`.
3. **Hero**
    - **Attributes**:
        - `hero_id` (**Primary Key**) – unique identifier for each hero.

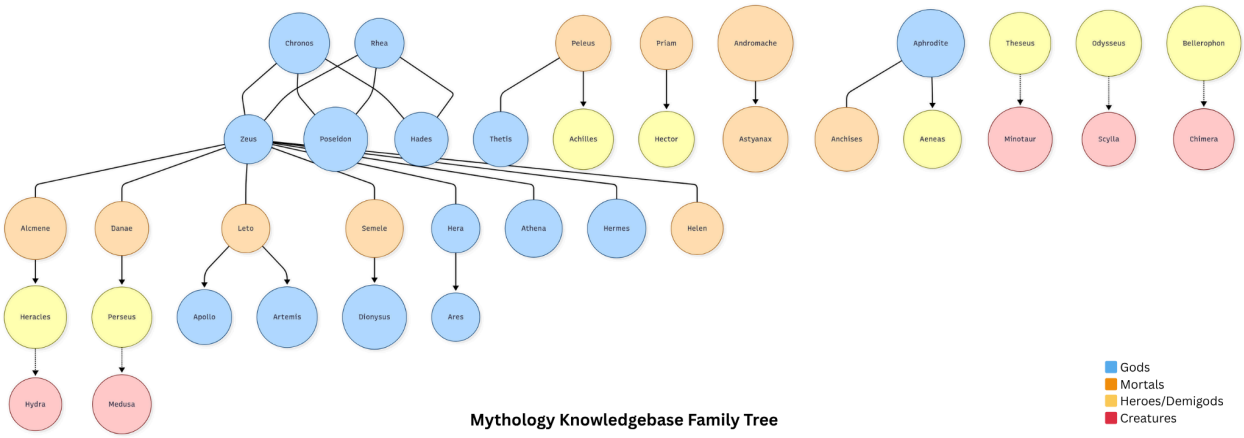
        - `name` – name of the hero (e.g., Heracles, Perseus).
        - `parent1_id` (**Foreign Key**) – references a **God**.
        - `parent2_id` (**Foreign Key**) – references a **Mortal**.
    - **Relationships**:
        - Child of a god and a mortal (demigod lineage).
        - Linked to **HeroCreature** to show which creatures the hero encounters or defeats.
4. **Creature**
    - **Attributes**:
        - `creature_id` (**Primary Key**) – unique identifier for each creature.
        - `name` – name of the creature (e.g., Minotaur, Hydra).
    - **Relationships**:
        - Connected to heroes through **HeroCreature** (many-to-many relationship).
5. **Domain**
    - **Attributes**:
        - `domain_id` (**Primary Key**) – unique identifier for the domain.
        - `name` – name of the domain (e.g., Sky, Sea, Underworld).
    - **Relationships**:
        - Referenced by **God** to show their sphere of influence.

6. **Relationship**
    - **Attributes**:
        - `rel_id` (**Primary Key**) – unique identifier for the relationship.
        - `parent_id` (**Foreign Key**) – can reference a **God** or **Mortal**.
        - `child_id` (**Foreign Key**) – can reference a **God**, **Mortal**, or **Hero**.
        - `relationship_type` – description of the relation (e.g., father, mother, mentor).
    - **Relationships**:
        - Acts as a generalized link between any parent–child pair in the mythology.
7. **HeroCreature**
    - **Attributes**:
        - `hero_id` (**Foreign Key**) – references **Hero**.
        - `creature_id` (**Foreign Key**) – references **Creature**.
    - **Relationships**:
        - Resolves the many-to-many relationship between heroes and creatures.

Connection:

- **God** ↔ **Domain**: One-to-many (each god belongs to one domain; each domain may have many gods).
- **Hero**: Always has one god parent and one mortal parent.

- **Creature**: Connected to many heroes via the **HeroCreature** bridge table.
- **Relationship**: Flexible table to store parent–child or other relationships between any characters.
- **Mortal**: Linked to heroes as parents and may also appear in **Relationship** table.



Mythology Knowledgebase Family Tree

Sample Input / Output:

Sample Facts:

```
god(zeus).
mortal(alcmene).
creature(medusa).
hero(heracles).
parent(zeus, heracles).
parent(alcmene, heracles).
domain(zeus, sky).
```

Sample Rules:

```
% Recursive ancestor rule
ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
```

```
% Demigod rule
demigod(X) :-
    parent(P1, X), parent(P2, X),
    god(P1), mortal(P2);
    god(P2), mortal(P1).
```

Sample Queries and Output:

```
%|----------------------------------------|
%| R #1: SIBLING RULE                     |
%|----------------------------------------|
sibling(X, Y) :-
    parent(P, X),
    parent(P, Y),
    X \= Y.
```

?- sibling(zeus, poseidon).

true.

```
%|----------------------------------------|
%| R #2: DEMIGOD RULE                     |
%|----------------------------------------|
demigod(X) :-
    parent(P1, X),
```

```
    parent(P2, X),
    god(P1), mortal(P2);
    god(P2), mortal(P1).
```

?- demigod(aeneas).

true.

```
%|---------------------------------------|
%| R #3: ANCESTOR RULE (RECURSIVE)       |
%|---------------------------------------|
ancestor(X, Y) :-
    parent(X, Y).
ancestor(X, Y) :-
    parent(X, Z),
    ancestor(Z, Y).
```

?- ancestor(chronos, heracles).

true.

Conclusion and Challenges:

Conclusion:
The Mythology Knowledgebase demonstrates how Prolog can effectively represent and query complex relationships in a mythological context.
Using facts and recursive rules, it allows inference of multi-level relationships like ancestry and complex classifications such as “demigod.”

Challenges:

- Designing a clear ER model that supports gods, mortals, and heroes with shared relationships.
- Handling multiple parent types (god and mortal) in Prolog rules.
- Structuring recursive rules to avoid infinite loops.
- Managing many-to-many relationships (heroes vs. creatures).