Basics of database systems

Project Age of Wonders – Database design

By

Tuomas Lättilä (000437136)

Konsta Jalkanen (000489689)

Lappeenranta-Lahti University of Technology LUT
Software Engineering

Basics of database systems
Spring 2024

TABLE OF CONTENTS

1 DEFINITION

In the project ‘Age of Wonders - database’, the idea is to develop a comprehensive database for role-play games that have different types of organisms in them. The database will work as a bestiary of the fantasy species of the planet Wonha. It will include the species across all of the organisms, their stats, and living styles, and every organism will be assigned with its living areas. This database makes it easier to manage and organize the information of the Age of Wonders role-play game and it makes easier to display the desired data during play.

The database will include five tables for different organisms. Aqua for animals living on water, Bestia for animals living on land, Fungus for mushrooms and such, Herba for plants, and Morbus for diseases, small micro-organisms, and parasites. There is also organismType unknown for unknown cases. Most of the naming comes from Latin. The database will also include living areas, living styles, and souls that are linked to those organisms.

These database queries will be implemented:

1. Find specific(s) Organisms or morbus by name, id, or keywords in their description.
2. List all expect foreign keys within any table.
3. List all organisms, their Id, description, their stats baseline, and natural skills.
4. List all organisms by the same living style.
5. List all organisms by living area.
6. List all organisms by can be infected with a specific morbus chosen by the user or by specific symptom.
7. List all organisms and which morbus can infect them.

2 MODELING

2.1 Conceptual model

Figure 1 shows the ER model of the designed database. There are five entities (Living areas, Living Styles, Soul, Organism Type, and Morbus) and one weak entity (Organism). An organism has attributes ‘orgID’, ‘Name’, ‘description’, and ‘type’. Morbus has attributes ‘orgID’, ‘Name’, ‘description’, and ‘Symptoms’. In both entities, the ‘orgID’ and ’Name’

are their key attributes. Soul entity contain a baseline for the stats, natural skills, and Skill limits which each are a multivalued attribute. Living areas and living styles have name, id, and description attributes. The relationships are N:M relationship in between Organism and Living area, and Morbus and organism. N:1 relationship between Organism and Living style, and Organism and type. A 1:1 relationship is between Organism and the Soul.

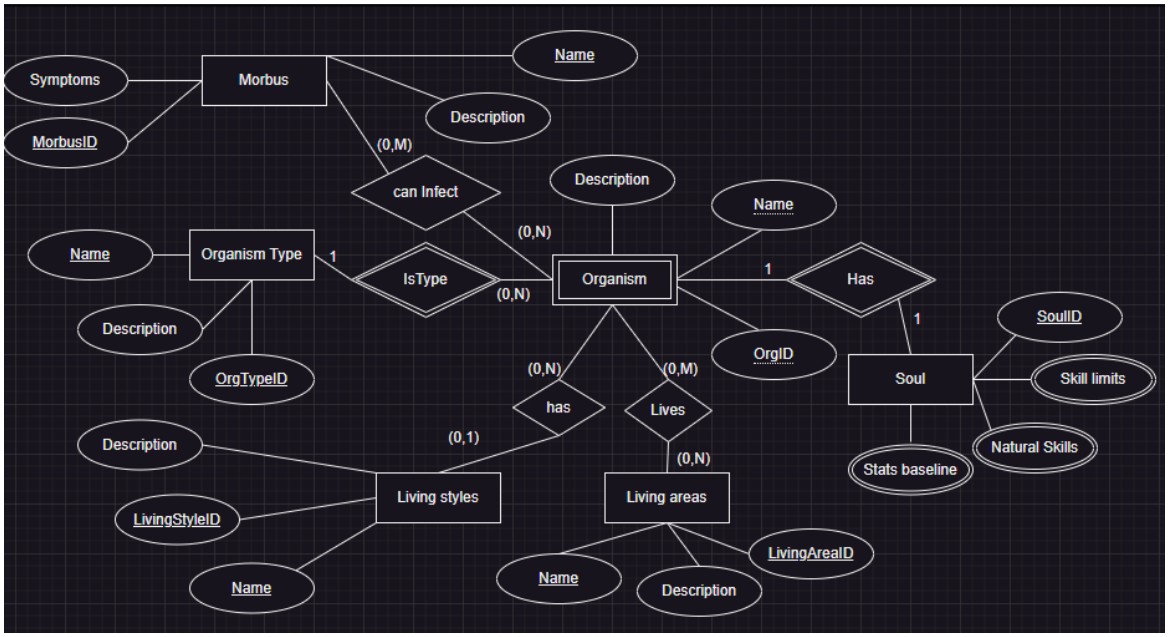


Figure 1: ER model

2.2 Logical model

Figure 2 shows the logical model that has been created based on the ER model. The N: M relationships are made by adding a middle table that holds the connection data. A 1:1 relationship is made with both ways going foreign keys to show which entities are connected. Other relationships (1:N) are made with one-way foreign keys where the key is saved in the N end of the relationship. Otherwise, it fully follows the ER model.

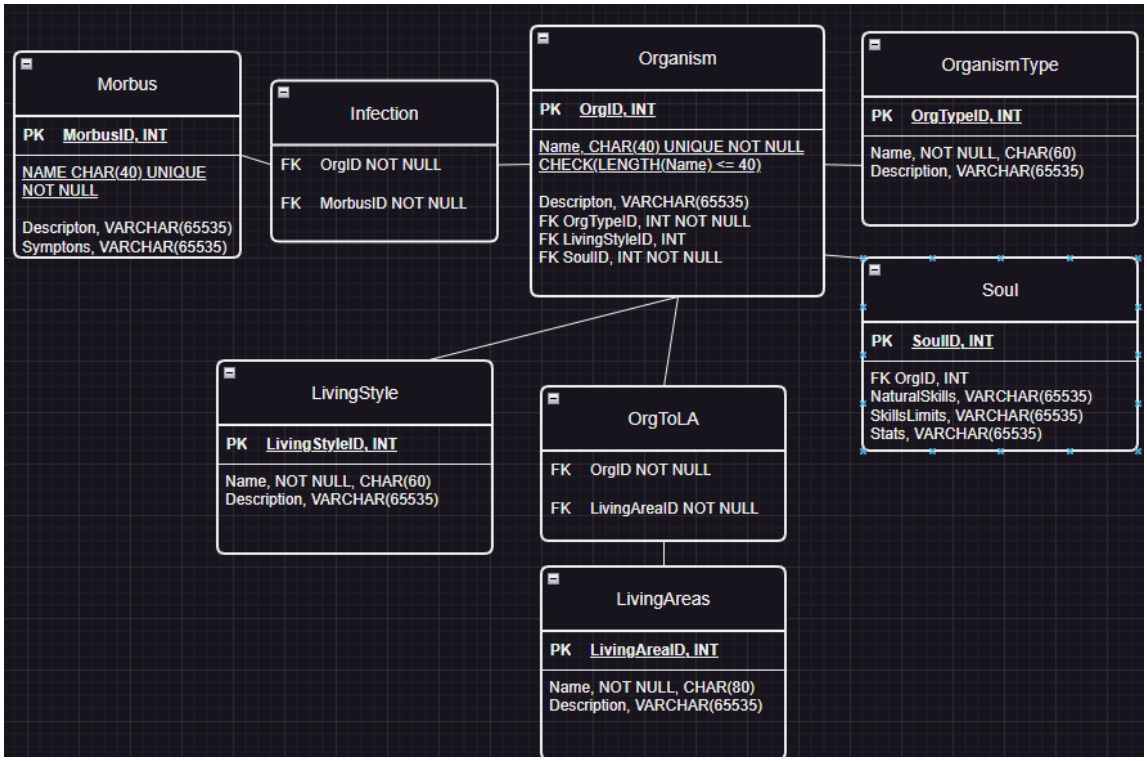


Figure 2: Logical model from the conceptual model

3 DATABASE IMPLEMENTATION

3.1 Constraints and relations of the database

During implementation, the following constraints are created for the relations:

- **Organism**:
  - **OrgID**: INT, NOT NULL, PRIMARY KEY
  - **Name**: CHAR(40), NOT NULL, UNIQUE, CHECK (LENGTH(Name) <= 40)
  - **Description**: VARCHAR(65535), DEFAULT 'No description given!'
  - **OrgTypeID**: INT, NOT NULL, FOREIGN KEY reference to OrganismType, ON DELETE CASCADE
  - **LivingStyleID**: INT, FOREIGN KEY reference to LivingStyle, ON DELETE SET NULL
  - **SoulID**: INT, NOT NULL, FOREIGN KEY reference to Soul, ON DELETE CASCADE
- **Morbus**:
  - **MorbusID**: INT, NOT NULL, PRIMARY KEY
  - **Name**: CHAR(40), NOT NULL, UNIQUE, CHECK (LENGTH(Name) <= 40)
  - **Description**: VARCHAR(65535)
  - **Symptoms**: VARCHAR(65535)
- **OrganismType**:
  - **OrgTypeID**: INT, NOT NULL, PRIMARY KEY
  - **Name**: CHAR(60), NOT NULL
  - **Description**: VARCHAR(65535)
- **LivingStyle**:
  - **LivingStyleID**: INT, NOT NULL, PRIMARY KEY
  - **Name**: CHAR(60), NOT NULL
  - **Description**: VARCHAR(65535)
- **LivingAreas**:
  - **LivingAreaID**: INT, NOT NULL, PRIMARY KEY
  - **Name**: CHAR(60), NOT NULL
  - **Description**: VARCHAR(65535)
- **Soul**:

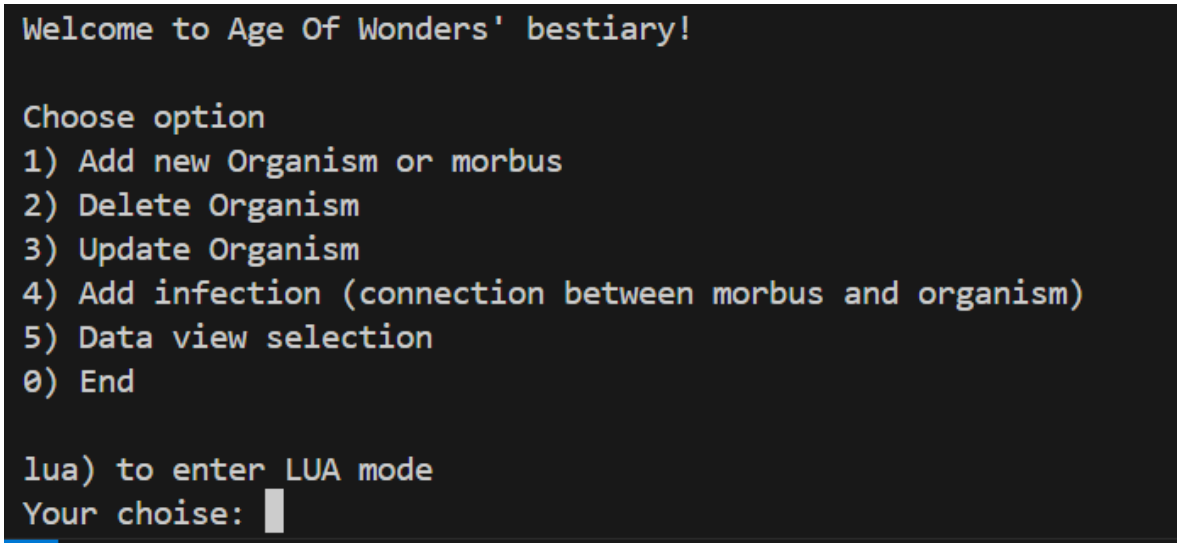
  - **SoulID**: INT, NOT NULL, PRIMARY KEY
  - **NaturalSkills**: VARCHAR(65535)
  - **SkillsLimit**: VARCHAR(65535)
  - **Stats**: VARCHAR(65535)
  - **OrgID**: INT, FOREIGN KEY reference to Organism, ON DELETE CASCADE
- **OrgToLA**:
  - **OrgID**: INT, NOT NULL, FOREIGN KEY reference to Organism, ON DELETE CASCADE
  - **LivingAreaID**: INT, NOT NULL, FOREIGN KEY reference to LivingAreas, ON DELETE CASCADE
- **Infection**:
  - **OrgID**: INT, NOT NULL, FOREIGN KEY reference to Organism, ON DELETE CASCADE
  - **MorbusID**: INT, NOT NULL, FOREIGN KEY reference to Morbus, ON DELETE CASCADE

In addition to the integrity constraints listed above, the database will also implement two indices; One based on the organisms' name, and another based on the morbuses' name. These indices allow quick search for organisms or morbuses.

3.2 Python interface

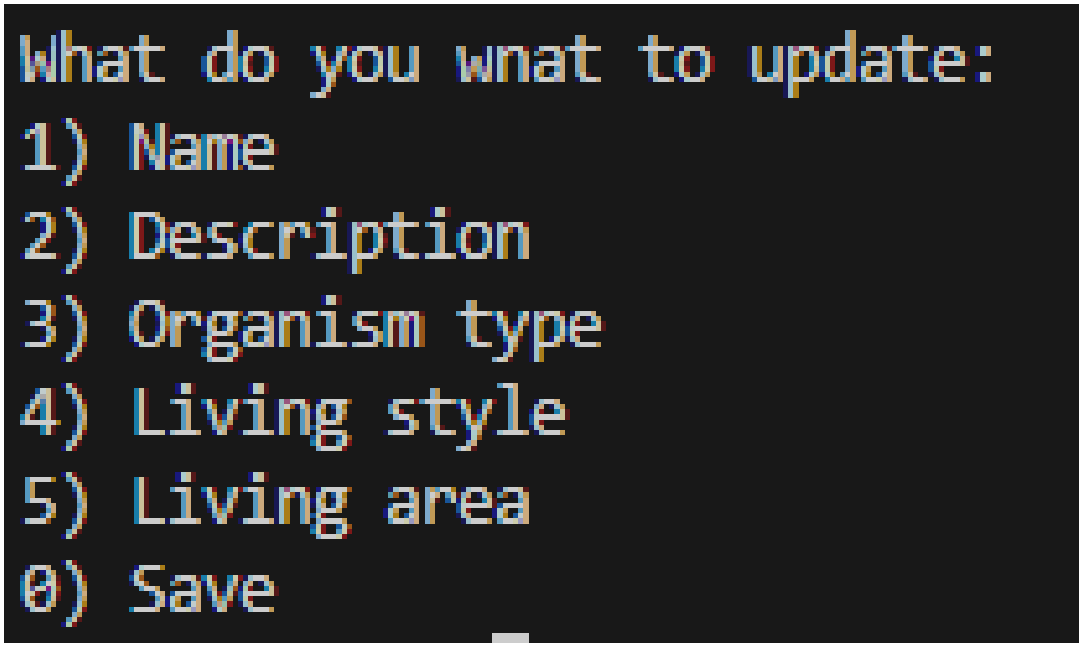
In addition to the database, we created a Python interface for the database. This interface is used via the command line and has some basic data modification functions and data query print functions. It runs in a loop and can be exited when the user wants. The following pictures and tables describe the functionalities in more detail.

Main option selection:



Option	Description
1) Add new Organism or morbus	Here you can insert a new organism or morbus to the database. The program asks for all the necessary values for the new Organism. Also, a new Soul will be created at the same time. A new soul will be linked to the organism.
2) Delete Organism	Here you can delete some specific Organism by their ID. Program prints all the Organism, and asks for ID.
3) Update Organism	Here you can update some specific Organism. It runs in a loop, and you can update how many values you want. Stops by saving the updated values.
4) Add infection	The program adds a connection between morbus and the organism
5) Data view selection	Another selection menu, where the user may choose what to view.
lua) LUA mode	The user may put any SQL command as input and the program reads output and commits changes if necessary.
0) End	Exits the program.

Update options:



Name	Description
1) Name	Users may update the name of any organism
2) Description	Users may update the description of the chosen organism.
3) Organism type	Users may update the organism type of the chosen organism.
4) Living style	Users may update the living style description of the chosen organism.
5) Living area	Users may update the living area of the chosen organism.
0) save	Changes will be saved. The commit command is run.

View options:

```
Choose a dataview that you want to see:
1) Print living areas and living style of a specific organism
2) Print Organisms
3) Print Souls
4) Find by id or keyword
5) Print Infections
0) Exit
Your choise:
```

Name	description
1) Print living areas and living style of a specific organism	Prints living areas and living styles of a specific organism.
2) Print organisms	Prints organisms.
3) Print souls	Prints souls.
4) Find by id or keyword	Find the organism or morbus by id, name, or keyword in the description or symptom.
5) Print infections	Prints all infections.
0) exit	exits

4 DISCUSSION

Noticeable things:

In the return box is a file called querries.sql. It contains all queries and commands used in the Python front end. The file itself is NOT runnable due to multiple queries asking for user input. You can run them if you change “+[something]+” to a variable.

The Python programs use libraries: “os”, “time”, and “sqlite3”

Link to Github: https://github.com/jepuli124/AOWdatabase