# Developing a Conceptual Database for an MMORPG Item System using Object-Relational Concepts

Term Project for ACS 575 – Dr. Yoo

Purdue University, Fort Wayne

Alekzander Green

### Abstract

The Massively Multiplayer Online Role-Playing Game (MMORPG) video game genre is popular in the world of online gaming. It is a mix of two genres: Massively Multiplayer Online (MMO) games and Role-Playing Games (RPGs). MMOs are characterized by a large online presence, with many players playing the game together at one time. RPGs are characterized by fantasy environments that allow players to experience the game, usually by moving through the environment, defeating enemies, or collecting items for future use. The combination of these two genres makes for a data-intensive game system that needs to not only handle a large online presence, but also needs to handle all of the data for the players’ various inventories and such. To handle such data requirements, a database will be necessary. Due to the nature of the data used in the game, object-relational concepts seemed a natural fit. Therefore, a conceptual database that used object-relational concepts was developed using PostgreSQL, and an API built using FastAPI and served via Uvicorn was developed to support the database. This was an overall success, but future work will be necessary to tune the project and integrate it with a future MMORPG system.

### Project Description

### Problem Statement

MMORPGs need to store a lot of data. A large part of the genre is collecting and finding new items to use in the game. This project will attempt to build a database that is capable of storing and retrieving different kinds of items in an MMORPG and allowing the game to store a user’s inventory in a reliable manner. This will require the planning and implementation of a database as well as a backend system to support API calls to the database.

### Objectives

**Market Research** – Several RPG inventory systems were investigated to see how they classify items and organize inventories. The games selected will be from an array of different RPGs from various subgenres. This will ensure that we have a good idea of how we might implement an inventory system.

**Schema Design** – Conceptual, logical, and physical schemas were designed to plan and demonstrate how the database should function and how it has been implemented.

**Database Implementation** – A database capable of storing and retrieving data concerning items, inventories, and players has been implemented. This database system mostly focuses on how items and inventories are stored in a wide array of tables, but limited player information is also stored to connect players to their inventories. The database implementation was done using PostgreSQL and pgAdmin4.

**Backend Implementation** – A backend API is implemented to allow for streamlined and controlled database querying. Despite my prior experience with Django and the Django REST Framework, I decided to implement the API using FastAPI. This API allows me to retrieve information for any item in the database, add new items to the database, add items to player inventories, remove items from player inventories, retrieve player states, and store player states. Further actions may be needed, but these requirements will be necessary without a doubt. In addition to the core FastAPI library, the Uvicorn library was used to host the server and a PostgreSQL database adapter called psycopg3 was used to make database queries.

**Demo Application** – After implementing the database and backend API, a demo application was developed to showcase the database working. A simple desktop application built in C# with .NET Core was built for these purposes. This application is very primitive, and only serves to showcase limited inventory interactions.

**Testing and Refining** – Testing will be done continuously as prototyping is being carried out. After the prototypes are all put together, final testing and refining will take place. Many tests were carried out using Postman.

**Presentation** – Once the prototype is properly refined, a presentation will be developed to highlight the technology used and the project results.

**Unmet Goals** – The demo application is very primitive and does not properly make use of all of the possible interactions. Additionally, many of the database tables remain largely unused. Finally, use of stored procedures and UDFs were in process, but could not be refined in time to implement correctly, so they were withheld.

### Scope

Given that the system will be able to store custom data, there will not be much data available for the database. This system will focus on an MMORPG’s item and inventory system. Other systems within an MMORPG may be implemented but are considered stretch goals. The item system will consider different types of items and will implement extensive tables to accommodate them. This project will:

* Implement a database system.
* Implement a backend API.
* Develop a basic demo.
* Include tables for items of different types.
* Include tables containing player inventories and/or state.

This project will not:

* Consider other aspects of an MMORPG.
* Include a game that uses the database.
* Contain extensive data within tables.
* Have extensive functionality beyond inventory and item interactions.

### Conceptual Schema

A diagram of a company

Description automatically generated

### Logical Schema

ENUMS

account\_status\_enum ('ONLINE', 'OFFLINE', 'BANNED', 'INACTIVE')

server\_status\_enum ('UP', 'DOWN')

item\_rarity\_enum ('Junk', 'Common', 'Uncommon', 'Rare', 'Mythic')

weapon\_hands\_enum ('Main-Hand', 'Off-Hand', 'Two-Hand', 'One-Hand')

weapon\_type\_enum ('Sword', 'Axe', 'Spear', 'Hammer', 'Shield')

armor\_slot\_enum ('Head', 'Body', 'Waist', 'Hands', 'Legs', 'Feet')

armor\_type\_enum ('Light', 'Medium', 'Heavy')

SEQUENCES

item\_id\_seq (BIGINT)

TYPES

item\_type (id, name, rarity [item\_rarity\_enum], description, cost, is\_stackable, icon)

item\_stack\_type (item [item\_type], stackSize)

equipment\_set\_type (mainHand [weapon], offhand [weapon], head [armor], body [armor], waist [armor], hands [armor], legs [armor], feet [armor])

TABLES

item [item\_type] (**id** …)

equipment [item] (…, durability, remDurability, repairCost)

weapon [equipment] (…, hands [weapon\_hands\_enum], type [weapon\_type\_enum], damage)

armor [equipment] (…, slot [armor\_slot\_enum], type [armor\_type\_enum], defense)

quest\_item [item] (…)

junk\_item [item] (…)

usable\_item [item] (…, uses, remUses, cooldown, remCooldown)

ingredient [item] (…)

account (**email**, password, status [account\_status\_enum])

server (**id**, name, status [server\_status\_enum])

character (***server\_id [server(id)]*, name**, *account\_email [account(email)]*, inventory[], bank[], equipped[equipment\_set\_type], money)

### Physical Schema

A screenshot of a computer program

Description automatically generatedA screenshot of a computer code

Description automatically generated

A diagram of a computer

Description automatically generated with medium confidence

### System Architecture

The system uses a three-tiered architecture with a database server, application server, and client application.

**Database Server** – The database server was built using PostgreSQL as the DBMS of choice. Development occurred using pgAdmin4. The database server runs as a service on my local machine.

**Application Server** – The application server was built using the Python FastAPI library. The FastAPI app is served via Uvicorn. The FastAPI app connects to the local database server using the psycopg3 database adapter. All requests are made via HTTP with JSON data passed as inputs and outputs.

**Client** – The client application is a simple Windows Forms App built using C# and .NET Core. This is a GUI application that makes requests using the built-in HTTPClient and async structures.

### DBMS Technology and Development Methods

The system was developed using various technologies and development methods.

**PosgreSQL** – PostgreSQL (Postgres) is an Object-Relational (OR) DBMS that was used to implement a database server. Several types and tables were developed in accordance with Postgres’s standards of implementation. This DBMS was chosen as it is open-source and widely popular in addition to providing many OR features that were chosen for this project.

**pgAdmin4** – This is the application that comes with the installation of Postgres. This application was used to write queries and develop the database.

**FastAPI** – This is a Python API library. This library was used to develop a simple backend API using various functions with decorators to enable efficient routing. This framework was chosen to develop the backend because it was a new technology that I had been wanting to explore.

**Uvicorn** – This is a Python server library. This library was used to host the FastAPI application. It was chosen for its simplicity in addition to FastAPI being built on top of Starlette, an extension of Uvicorn.

**Psycopg3** – This is a Python-PostgreSQL database adapter library. This library was used to query the database server for the purpose of responding to the client. This was chosen over other technologies like SQLAlchemy because it is a popular database adapter rather than an ORM. The reason for this being a factor is because I was unsure of how difficult it would be to use an ORM for some of the Postgres-specific OR concepts that were implemented.