DATABASE EXAM

By

Kristofer Pedersen
cph-kp278

Mads Teddy Francois Knudsen
cph-mk682

Michael Frederiksen
cph-mf315

Søren Merved
cph-sm428

Github link:
https://github.com/Mfknudsen/Database-Exam

PowerPoint:
https://docs.google.com/presentation/d/1_UwxjmLRqS2JXsLkI4M9Io2KoOZ5e0vRIyEwgzeh8QA/edit?usp=sharing

Select a business or organization:
We have chosen to build our exam databases around a hobby, which two of our four members are actively enjoying, called Warhammer 40k. From this hobby we have taken two aspects to focus around.
The first aspect is the story which is often called lore. The lore comes from multiple sources, be it books, games or animations.The lore is expansive and is updated quite often from for example new releases of books.
The second aspect is the popular tabletop board game of the same name. This board game contains different armies with different squads, vehicles, monsters and leaders. All of which have different rules which make it difficult to remember it all when playing.

Define application domain:

- Use case 1

User is currently playing a game against another opponent and forgets the rule of the unit (a squad) and wants to use our application to get the rules.

- Use case 2

User is currently playing a game against another opponent and wants to know all the rules regarding one of the opponents units.

- Use case 3

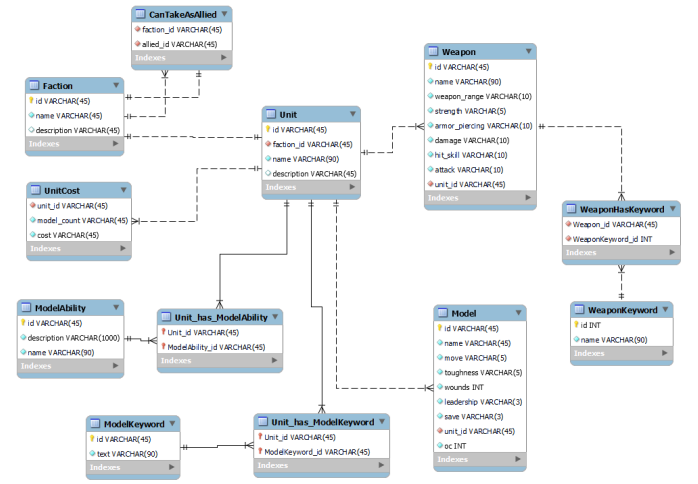
User is currently playing a game against another opponent and wants to know more about the lives of a unit within the game's universe.

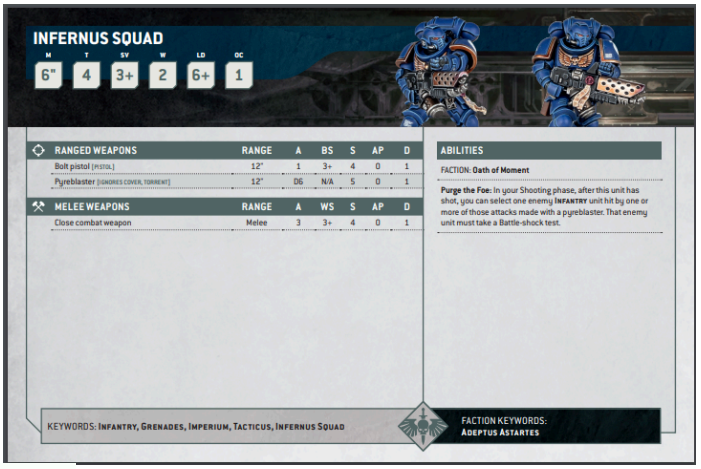
Select and motivate the selection:
Our project idea originates from what one of our group members uses in their workplace. A chatbot based on the internal documents from the workplace with multiple databases feeding into a single vector database.
For our project we have decided to use one sql and two document databases as sources for a single vector database.
For the data used in the sql database, we used the data found in this repository https://github.com/BSData/wh40k-10e . We parsed and cleaned data from multiple .cat files containing the rules for each faction and their respective units.
For the first document database we used web scraping on a fandom wiki website to get the lore of the fictional universe.
And for the second document database we keep the chat historic based on a username that is updated when a user uses the application.

Design, create and populate the databases:
For our sql database we used mysql and their workbench application to create an EER diagram from which we could forward engineer to generate a query script and would drop it if it already existed and then create a new version of the database.

For both of the document databases we didn’t need to create a structure like with sql. For the lore we had it generate an id and for the data we had title, url and content as strings, we had chunk id as an int and embedding as an array of floats.

The following image represents a dataslate for a unit. It gives the information, profiles and specs for a unit. Several units make up an army or a faction. Each unit is unique and has rules associated with it.
Each unit can consist of several models with profiles, in this example there is only 1 model hence 1 profile.
Weapons can either be melee or ranged but for simplicity we have merged it to one table, Ballistic skill (BS) and Weapon Skill (WS) has been replaced with hit_skill in the table weapons.
Weapons can have several keywords, and are unit exclusive to a single weapon, therefore the many to many relationship.
Weapon keywords can be seen in the profile for under weapons for example Pryeblaster has the keywords “torrent” and “ignores cover”.
Model keywords are similar ideas as weapon keywords. They are at the bottom of the unit card.






```
_id: ObjectId('665c34411f12872976efed9a')
user_id : "warhammer"
messages : Array (7)
  0: Object
     role : "system"
     content : "You are a helpful chatbot that ans
  1: Object
     role : "user"
     content : "Hello"
  2: Object
     role : "assistant"
     content : "Hi! How can I assist you today?"
  3: Object
  4: Object
  5: Object
  6: Object
```


To store the conversations between users and the chatbot, we decided to create a mongoDB. Each conversation has an id, which is autogenerated, the id of the user that wrote the message, as well as an array containing all the messages of the chat, both the ones from the user and the chatbot.

Deploy the databases:
For the two document databases and our vector database, which uses Mongodb and pinecone respectively, they both are only using a free version of their services. Sql is only deployed on localhost.

With our lore document database in MongoDB we have it set up as a clustered database with one primary and two secondaries.

If we look at our lore database using CAP theorem.
CAP:

- Consistency:

With our cluster nodes all being within the same region we can expect that a change in active nodes and with which is currently the primary will have little to no major effect for the speed.

- Availability:

With a clustered setup we have a better chance of the database being available for usage by a user.

- Partition tolerance:

We currently only have two backup nodes other than the primary nodes which will work with a maximum of two breaks.

Create one or more simple client application:
We have decided to make a simple application interface where the user first inputs a username, so that the chat historics can be stored properly, and then an input for the users query and an output field where the chatbot's answer can be displayed. Both fields are text fields.

Replication:
We set up replication locally on our MySQL database using GTID based replication. GTID is a unique identifier created and associated with each transaction committed on the source server, this ensures each transaction is uniquely identified across servers. This also means that each transaction committed on the source can’t be applied more than once on the replicas, as the transactions are skipped if the replica encounters a transaction GTID, which has already been applied.
How to set up GTID based replication locally, can vary depending on versions and OS, but from our use it can be broadly summarized in 4 steps:

- Enabling GTID, using row based logging for best performance with GTID, on the primary server and the replication server, specifying unique server ids.
- Creating a replication user on the primary server, with necessary privileges.
- Creating a backup of the primary database, and restoring it on the replica for initial data.
- Setting the replication server up for replication, by setting the replication source to the primary server, by specifying the primary server's details (name, port, etc.)