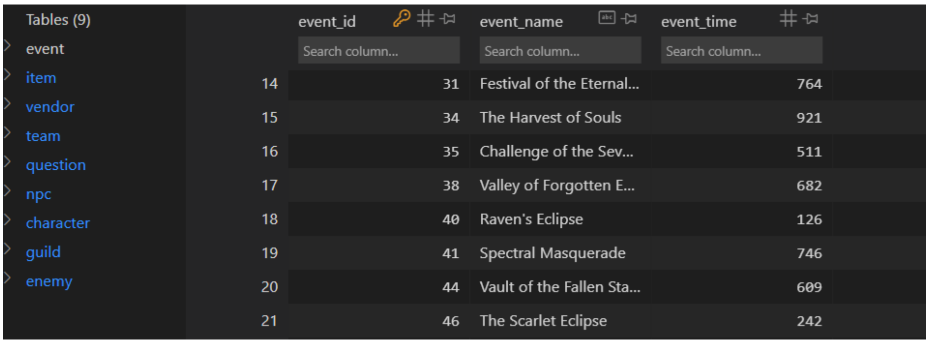
REPORT WEEK 2

For the Week 2 challenge, we started by implementing a SQLite test database with a straightforward schema to practise loading data. Following that, we made necessary adjustments to the existing MySQL database to streamline the data reading process. We also created a 'create_database.py' script for easy database resetting in case of errors.



Tables (9)
event
item
vendor
team
question
npc
character
guild
enemy

	event_id	event_name	event_time
	Search column...	Search column...	Search column...
14	31	Festival of the Eternal...	764
15	34	The Harvest of Souls	921
16	35	Challenge of the Sev...	511
17	38	Valley of Forgotten E...	682
18	40	Raven's Eclipse	126
19	41	Spectral Masquerade	746
20	44	Vault of the Fallen Sta...	609
21	46	The Scarlet Eclipse	242

Our initial approach for data processing, as outlined in 'data_reader.py,' involved attempting to directly parse the generated data and load it into the database. However, this method proved to be inefficient and error-prone. We experimented with using 'INSERT IGNORE' to handle duplicate values, but inconsistencies persisted.

In 'data_reader2.py,' we significantly improved our data processing workflow. This class was dedicated solely to processing the text file data without loading it into the database. We created two folders—'entities_csv' and 'enemies_csv'—where we stored the initially messy data in concise CSV file formats, making it easier to work with and manipulate.

Once we had the data loaded into the appropriate CSV files, we turned to 'data_process.py.' Here, we utilised the Pandas library to load the CSV files into dataframes, enabling us to address several challenges we encountered:

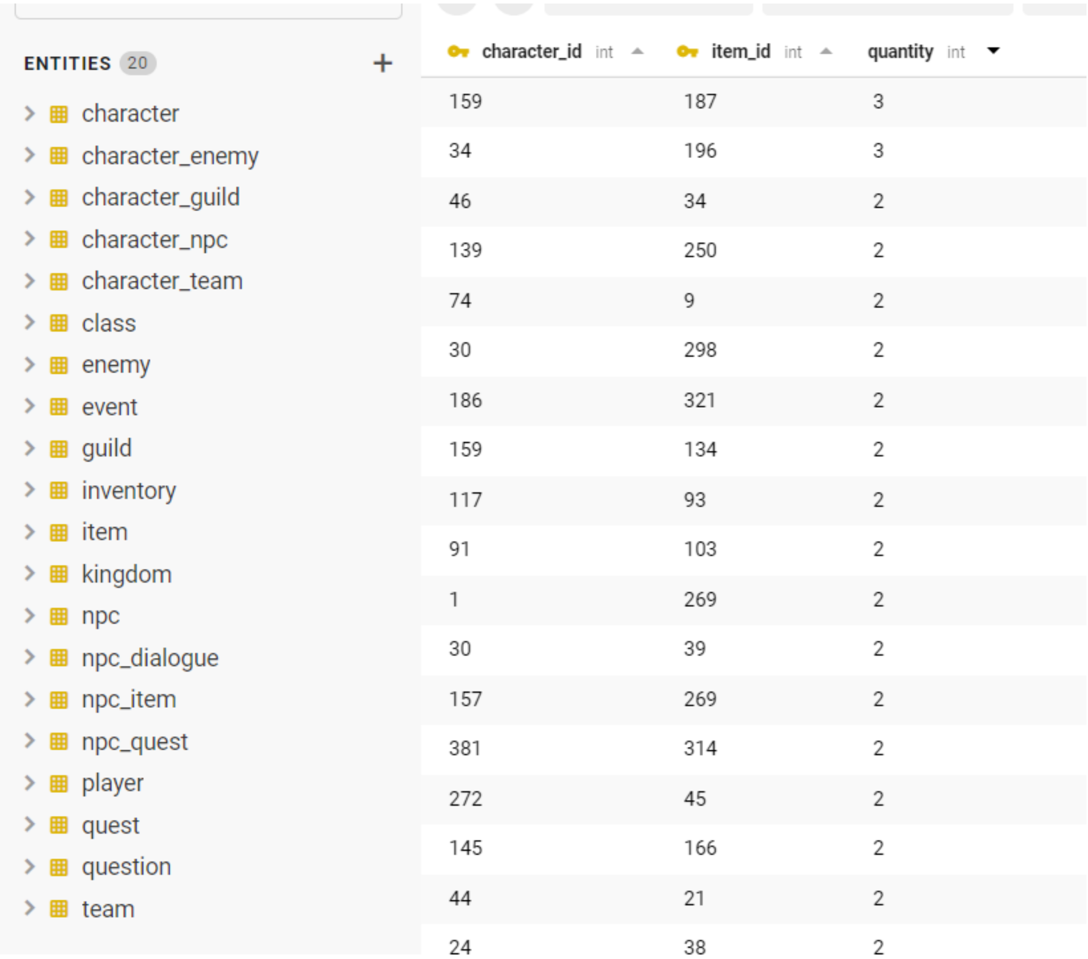
1) Ensuring no duplicates of primary key values.
2) Ensuring there were no null fields in our tables.
3) Ensuring that foreign key (FK) constraints were respected.
4) One of the most significant challenges we faced revolved around handling inventory items for players. This involved events with additional entities, and we needed to account for every time a player bought or sold an item to an NPC. We not only had to add or subtract the item but also validate its existence and ensure the item count didn't go negative. To address this, we used group-by statements and counting functions within the dataframes to maintain data integrity.

Furthermore, to align with our database structure, we had to modify the 'entities.json' file. This involved adding more entities, defining custom value sources, and adjusting certain parameters within the 'generate_data.py' script.

To bring all these intricate functionalities together, we created 'runner.py,' which performs the following tasks:

a. Sets up a fresh and ready-to-use database.
b. Executes two batch scripts to ensure the cleanliness of the CSV file folders.
c. Runs 'generate_data.py' three times, generating and reading the data into the CSV files.
d. Processes all the generated data and inserts it into the database while upholding database integrity and constraints.

In the end, we achieved a fully populated and consistent database containing 20 entities.



ENTITIES 20

- character
- character_enemy
- character_guild
- character_npc
- character_team
- class
- enemy
- event
- guild
- inventory
- item
- kingdom
- npc
- npc_dialogue
- npc_item
- npc_quest
- player
- quest
- question
- team

character_id int	item_id int	quantity int
159	187	3
34	196	3
46	34	2
139	250	2
74	9	2
30	298	2
186	321	2
159	134	2
117	93	2
91	103	2
1	269	2
30	39	2
157	269	2
381	314	2
272	45	2
145	166	2
44	21	2
24	38	2

-Niki Lalev