**Group Members:** Nathan Gagné, Yixi Wu, Chang Geng

**Due date:** December 23rd 2021

**COMP3380 Project Report – DnD Database**

**Summary of the data:**

We chose the topic of the Dungeons and Dragons creature, weapon, and spell data for our project because we knew that this type of data could lead to interesting queries and would be fun to work with as opposed to a more grounded dataset. Another reason we felt comfortable choosing it is because Nathan has extensive experience playing the game. We had someone who could provide insight on the details of the raw data and provide input on implementation decisions in order to remain faithful to what the data represents.

The original data we used is composed of three excel files that we converted into comma separated value (CSV) format:

* Monster.csv (192KB, 802 records)
* Spell.csv (74KB, 361 records)
* Weapon.csv (4KB, 50 records)

These three datasets were then broken down into smaller tables to account for multivalued attributes and a more manageable relational model.

**The data model:**

Despite having no tables that had to be merged during normalization on their primary keys, our final model had many tables because the Creature entity (renamed from Monster.csv) has many multivalued attributes that needed to be preserved.

Also, it was clear that Spell and Weapon could not be contained in the same table, but they both had damage types and would be used in relationships with the Creature table. In order to solve this problem and provide an intermediary between Creature and the Spell and Weapon tables, we created a DamageType entity. The WeakTo, ImmuneTo, Resists relationships would be between Creature and Damage Type, which would then be related to Weapon and Spell separately. This way is more logical and reduces coupling between our main tables if we were ever to update any of them.

The participation and cardinality ratios we detailed in our Part 1.2 submission were simple to determine and reflect the final database.

\*Note: During the implementation process, the WeaponProfs MVA table was removed from our relational model because it proved to be completely redundant with the data stored in CanUseWeapon.

**Summary of the database:**

1. Creature (60KB, cardinality: 798, arity: 14)
2. Speeds (23KB, cardinality: 1158, arity: 2)
3. CreatureSTs (16KB, cardinality: 754, arity: 2)
4. Skills (27KB, cardinality: 1121, arity: 2)
5. Senses (21KB, cardinality: 697, arity: 2)
6. Languages (26KB, cardinality: 1219, arity: 2)
7. AdditionalFeatures (51KB, cardinality: 1793, arity: 2)
8. DamageType (1KB, cardinality: 13, arity: 1)
9. Spell (43KB, cardinality: 361, arity: 14)
10. Weapon (3KB, cardinality: 51, arity: 13)
11. PlayerClass (1KB, cardinality: 12, arity: 6)
12. ClassSTs (1KB, cardinality: 24, arity: 2)
13. WeakTo (1KB, cardinality: 32, arity: 2)
14. ImmuneTo (8KB, cardinality: 381, arity: 2)
15. Resists (12KB, cardinality: 575, arity: 2)
16. SpellDmgType (4KB, cardinality: 141, arity: 2)
17. WeapDmgType (2KB, cardinality: 51, arity: 2)
18. CanUseWeapon (6KB, cardinality: 333, arity: 2)
19. CanUseSpell (20KB, cardinality: 953, arity: 2)

Total size of CSVs: 326KB Total # of records: 10,467

**SQL Queries:**

1. Which combinations of spells and attacks could a Warlock use such that an Ice Mephit (weak to Bludgeoning and Fire) would be weak to both? Show Weapon and Spell combinations.
2. Which spells deal damage but have no somatic component (don't need hand gestures to cast)? Order by spell damage in descending order.
3. What damage types are the most common in spells of lvl 5 and up? Show the damage types and the number of spells with that damage type as NumSpells5Above.
4. What creatures have a challenge rating (CR) smaller than 5, under 100 HP, and resist slashing, piercing, and bludgeoning? Show creature names, their challenge rating, and all of their stats.
5. How many creatures are immune to each damage type? Show each damage type and the number of creatures immune to that damage type as NumImmune.
6. How many fire and ice spells can each class use? Show class names, the number of fire spells as FireSpells, the number of cold spells as ColdSpells, and the sum of those as Total.
7. What are the 5 classes that can use the most weapons and spells? Show class name, number of weapons as NumWeaps, number of spells as NumSpells.
8. What are the 10 creature that have the highest sum of their six stats + Hit Points (HP) + Armor Class (AC)? Show the Creature's information, and the sum of those stats as SumStats.
9. What creatures resist the greatest number of weapons and spells? Show the creature's number of weapon and spell resistances.
10. What is the distribution of HP for all Creature?
11. What creature types generally walk the slowest? (Show the avg speed of those creature types)
12. What is the maximum amount of damage possible from a spell against a given creature at a given level. What is the damage type of that Spell? Choose the spell level and the creature to look up here: (user input)
13. What group of creatures with the same alignment can communicate the best with each other? (What language is known the most by creatures in the same alignment? What creatures knows that language? Creature with Language label "Any" or "All" can potentially communicate with everyone)
14. What are the most common Additional Features among all creatures?

**Summary of the interface:**

The database implementation we designed had database management and backend operations written in python, with an HTML frontend. It revolves around these three files contained in our project submission:

1. myDatabase.py
2. backend.py
3. frontend.html

**myDatabase.py:** (import sqlite3, traceback, os, and urllib.parse libraries)

In myDatabase.py, we create all of our SQL tables using the CREATE TABLE notation we learned in this course. We also insert all the data contained in our CSV files into the appropriate tables. In this file, we also have the hardcoded prewritten queries that the user can run in our interface as well as the GET \* FROM (TABLE) statements that are executed to get the entire tables to appear. It also contains the code that allows to limit the number of rows in the output and order by any single column.

**backend.py:** (import socket, json, sys, traceback, re, and myDatabase)

backend.py is our intermediary between frontend.html and myDatabase.py. We can receive HTTP requests for various interactivity in our frontend and call functions in our myDatabase.py to execute the SQL we want.

**frontend.html:**

frontend.html contains the bulk of the work that went into our interface. We use it to create all of the tables that contain our data, the buttons, and all the formatting that appears in our user interface. Whenever a button is pressed in our frontend, it sends an HTTP request to our backend, which then calls functions in our myDatabase.py to execute the appropriate SQL statements.

**Instructions:**

1. Unzip the contents of 3380ProjectGroup14.zip into a folder of your choosing.
2. Run backend.py using any python environment you have at your disposal.
3. Open up a browser and go to the address localhost:9999
4. Interact with an intuitive database interface!

Along with the self-explanatory check-boxes and drop-down menus, you can also click on a column header to toggle a sort by descending/ascending on that column.

The “+” boxes for some of the columns on the main tables are used to consolidate the multivalued attributes of a main entity into one simple location.