Name: Timothy Flucker

**Table of Contents**

[Project Direction Overview 2](#_Toc119333351)

[Use Cases and Fields 2](#_Toc119333352)

[Structural Database Rules 2](#_Toc119333353)

[Conceptual Entity-Relationship Diagram 2](#_Toc119333354)

[Full DBMS Physical ERD 2](#_Toc119333355)

[Stored Procedure Execution and Explanations 2](#_Toc119333356)

[Question Identification and Explanations 2](#_Toc119333357)

[Query Executions and Explanations 2](#_Toc119333358)

[Index Identification and Creations 2](#_Toc119333359)

[History Table Demonstration 3](#_Toc119333360)

[Data Visualizations 3](#_Toc119333361)

[Summary and Reflection 3](#_Toc119333362)

# Project Direction Overview

I would like to develop a database for a simple RPG (role playing game) like the 1989 video game DragonQuest. The database outlined below would contain all the necessary information about the Hero, Monsters, Skills, and Items that the player could interact with as they play the game. The data would be mostly Strings and Integer values which would be saved as VARCHAR, DECIMAL, and BIGINT values, however there will also be some DATE values saved to track when records are created and modified.

Some tables in this database will contain static data, such as Monster, Item, and Skill data which is not modified. These tables serve as reference values and some records may be copied and pasted into other tables or be used for stored procedures. Other tables, would be modified or have records added or deleted to keep an up-to-date account of player and item data.

The player can “save” the game which would take a snapshot of the current game data which could then be loaded and played again. This saved state would track the Hero’s abilities, inventory, and their current progress in the story. A quick summary of events would be provided to the player before they play so that they can determine what their next objective should be.

I am interested in this data and type of database because I really enjoy playing videogames and I really would like to learn how to create video games like DragonQuest. This game is a favorite of mine because it is a fun, low effort game that I really enjoy playing on my phone if I have a few minutes to spare. As I gained more experience in programming and software development, this game also became an interesting and fun mental exercise to try and deconstruct the game into code and programming logic. I am excited for this term project because I get to try and recreate this database on my own and get to try some of my own ideas for improvements to the game and new features.

**If you wanted to create my database on your machine, please run the scripts in the following order:**

1. create\_tables.sql
2. procedures.sql
3. data\_insertion.sql
4. (optional) questions.sql

# Use Cases and Fields

**Use Case 1: Player saves the game**

1. The player reaches a save point and chooses to save the game
2. The save point restores all HP and MP values to their maximum values
3. A snapshot of the players stats and inventory are saved

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it is needed |
| Save\_ID | Unique numerical identifier for each save | Unique information for the record |
| Save\_Date | The date and time this save occurred | Used to keep chronological history of saves |
| Player\_Level | Number indicating player’s level | Player’s level at the time of the save |
| Total\_Experience | Number indicating the sum of the player’s experience points | Used to verify the player’s current level and progress towards the next level |
| Player\_Info\_ID | ID value which relates to the PLAYER\_INFO table, which contains the player’s stats (HP, MP, MAX\_HP, MAX\_MP, ATK, DEF, SPD, etc.) at the time of the save | Allows for the re-creation of the player object with the same stats as the last save. |
| Player\_Gold | Number indicating the player’s total gold | Player’s total gold at the time of the save |

**Use Case 2: Player purchases an item in-game**

1. The player interacts with some merchant in-game to see what items are available at that location.
2. The player reviews the items and selects one that they want, specifying the quantity of the item they are about to purchase.
3. The player purchases the item and that item is now a part of their inventory.
4. This process can repeat until the player runs out of in-game currency or until they have everything they need.

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it is needed |
| Item\_ID | Unique numerical identifier for each item | Unique information for the record |
| Item\_Name | Name of the item | Differentiates the items available. |
| Item\_Effect | What effect the item has on the player or enemy | To give meaning to each item and allow the player to strategize |
| Buy\_Price | Numerical value for how much it costs to buy they item from a merchant | Determines if the item can be bought and its value |
| Sell\_Price | Numerical value for how much the player can sell the item for | Determines if the item can be sold and its value |
| Quantity | The quantity of the item in the player’s inventory | Determines how many instances of that item the player can use |
| Is\_Key\_Item | Boolean value to determine if item is key item or not | Used to determine if item is story-related and therefore cannot be sold |
| Is\_Usable\_In\_Battle | Boolean value to determine if item is usable in battle | Filters player’s inventory during battle to only show relevant items |
| Location\_ID | Name of the locations the item can be purchased from | Some items will be location-specific and other can be found at every location |

**Use Case 3: Player encounters a monster**

1. They player wanders through a location and triggers the application logic to initiate a battle
2. Based on the player’s location, a list of potential monsters is determined by the application
3. A random monster from this list is determined
4. The battle screen appears and shows the randomly selected monster
5. The player battles the monster and defeats it to earn the battle rewards (experience + gold)

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it is needed |
| Location\_ID | Unique numerical identifier for each location | Allows monsters to be location specific |
| Monster\_ID | Unique numerical identifier for each monster | Unique element for each monster |
| Monster\_Name | Name of the monster | Text is displayed to the player |
| Monster\_HP | Hit Points (Health) of the monster, numerical value | Used to determine how much damage is needed to defeat the monster |
| Monster\_MP | Magic Points of the monster, numerical value | Used to determine what spells the monster can cast |
| Monster\_ATK | Attack value (numerical) of the monster | Determines damage they can inflict |
| Monster\_DEF | Defense value (numerical) of the monster | Determines how they receive damage |
| Monster\_SPD | Speed value (numerical) of the monster | Determines whose turn is first (player or monster) |
| Experience | Experience Points (numerical) provided to the player once the monster is defeated | Provides progress to the player to help them level up |
| Gold | Gold (numerical) provided to the player once the monster is defeated | In-game currency used to purchase items and equipment |

**Use Case 4: Player levels up**

1. Player finishes a battle with a monster receives the battle rewards (experience + gold)
2. Application logic compares the players current level and total experience against the level chart.
3. If experience is greater than or equal to the total experience for the next level, then the player levels up and their stats improve and they may also learn some spell

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it is needed |
| Level | Numeric value for level | Primary key for the record |
| Exp\_Needed | Numeric value indicating how many experience points are required for that level | Used to determine what level the player currently is, and if they qualify to level based on their total experience |
| Spell\_Unlocked | Numeric value of the spell id | If the player’s level is greater than or equal to this level, they have access to this spell and spells of lower levels |
| STR\_Gain | Numeric value for how many points STR increases by during a level up | Printed to the player to show their characters improvement when leveling up |
| STR\_Base | Total base value for STR | The minimum value of STR before adding bonuses from equipment and items |
| DEF\_Gain | Numeric value for how many points DEF increases by during a level up | Printed to the player to show their characters improvement when leveling up |
| DEF\_Base | Total base value for DEF | The minimum value of DEF before adding bonuses from equipment and items |
| SPD\_Gain | Numeric value for how many points SPD increases by during a level up | Printed to the player to show their characters improvement when leveling up |
| SPD\_Base | Total base value for SPD | The minimum value of SPD before adding bonuses from equipment and items |
| HP\_Gain | Numeric value for how many points HP increases by during a level up | Printed to the player to show their characters improvement when leveling up |
| HP\_Base | Total base value for HP | The minimum value of HP before adding bonuses from equipment and items |
| MP\_Gain | Numeric value for how many points MP increases by during a level up | Printed to the player to show their characters improvement when leveling up |
| MP\_Base | Total base value for MP | The minimum value of MP before adding bonuses from equipment and items |

**Use Case 5: Player uses a spell in battle**

1. On the player’s turn in battle, the user decides to use a spell.
2. In the battle commands they select the “SPELL” option.
3. The player’s level is used to determine the different types of spells available
4. Each spell is shown with its associated MP cost
5. The player’s MP is shown to indicate which spells they can ‘afford’

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it is needed |
| Spell\_ID | Number value associated with the spell | Primary key of the record |
| Spell\_Name | Name of the spell | Displayed to the player if they choose to cast a spell |
| MP\_Cost | Numeric value of the MP cost of the spell | Compared against the player’s current MP to determine if the player can afford to cast that spell |
| Effect | Description of the spells effect | Provides the player with information about what each spell does to the player or enemy |
| Player\_MP | Numeric value of the current amount of MP that player character has | Used to determine what spells the player can afford to cast |

# Structural Database Rules

Use Case 1

* A player has multiple save files they can use to play the game; A save file can only associated to a single player.
* A save snapshot has a single HEROES\_JOURNEY record used to track major milestones; A HEROES\_JOURNEY record is related to a single save snapshot.

Use Case 2

* A merchant may exist at a location; a location may contain a merchant.
* An item (domain table) can be found in many player’s inventories; a player’s inventory may contain many items.
* An item (domain table) can be found in many merchant’s inventories; a merchant’s inventory may contain many items.
* An item can be equipment, consumable, or key\_item (subtypes); equipment, consumable, and key\_item are all items.

Use case 3

* A player must reside in a location; a location may contain the player.
* Each location contains many different monsters; a monster can be found in multiple locations. (MONSTER\_LOCATIONS)
* During battle, a monster may be able to cast multiple spells; Spells may be cast by different kinds of monsters in battle. (MONSTER\_SPELLS)
* A monster may be a minion or boss; all monsters are either a minion or a boss.

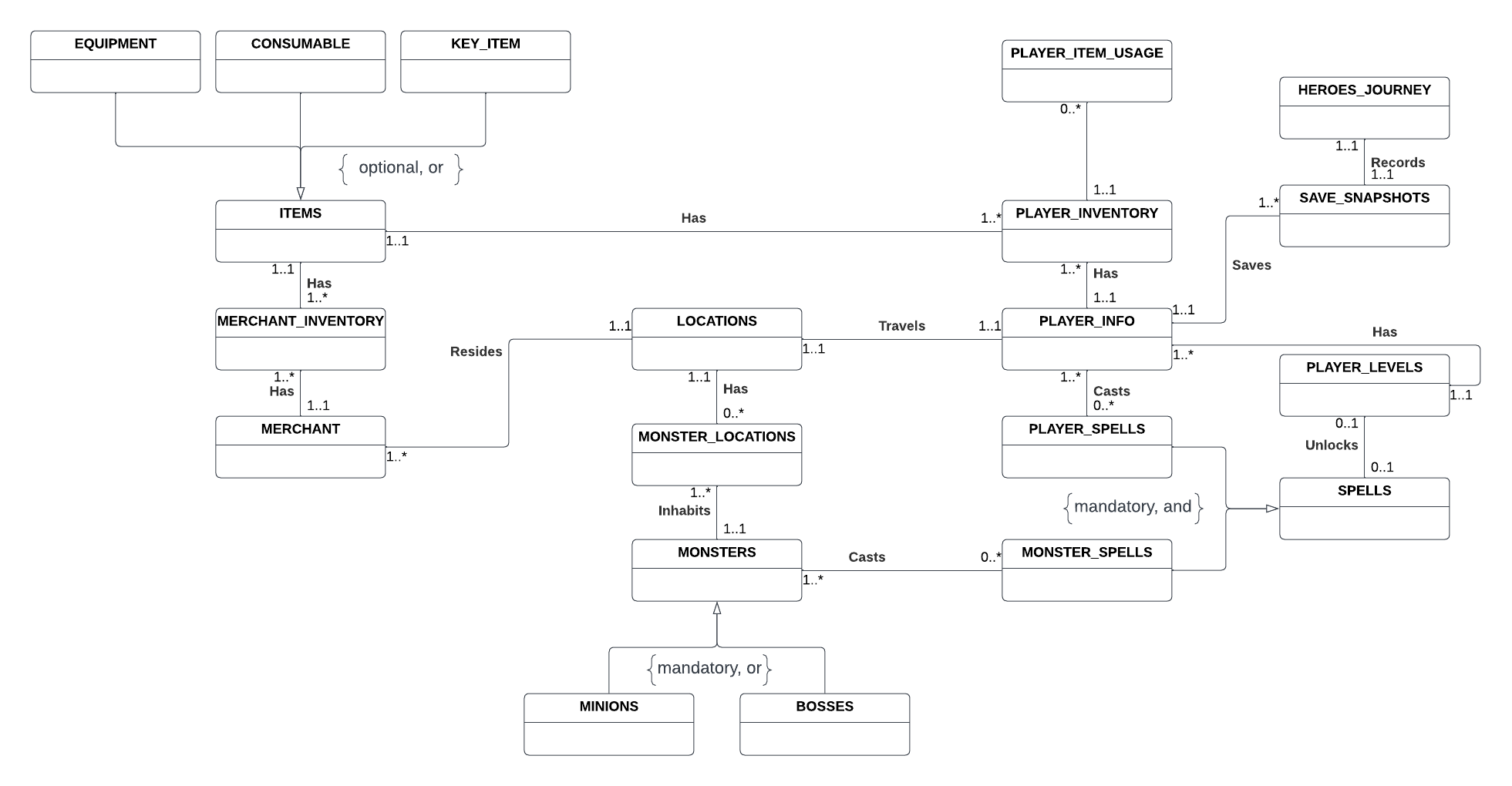
Use Case 4

* A player’s level is compared against a record in the player statistics table when they level up; a record in the player statistics table is used to update the current player info with new stat values when they level up
* One spell can be unlocked when a player levels up; when a player level up a single spell is unlocked.

Use Case 5

* A player can have many spells associated to themselves; a spell can be associated to multiple players. (PLAYER\_SPELLS)

# Conceptual Entity-Relationship Diagram



# Full DBMS Physical ERD

The central entity in this DBMS is PLAYER\_INFO and is used to relate to many other entities. A PLAYER\_INFO entity can have many SAVE\_SNAPSHOT entities and each SAVE\_SNAPSHOT entity has a single HEROES\_JOURNEY entity to track major milestones the player achieves in the game such as obtaining a key item or defeating a boss monster. The PLAYER\_INFO is also associated to a single LOCATION through its foreign key reference, and this location will update as the player travels the game world. A PLAYER\_INFO record is also linked to a few join tables between other entities to help represent some many-to-many relationships.

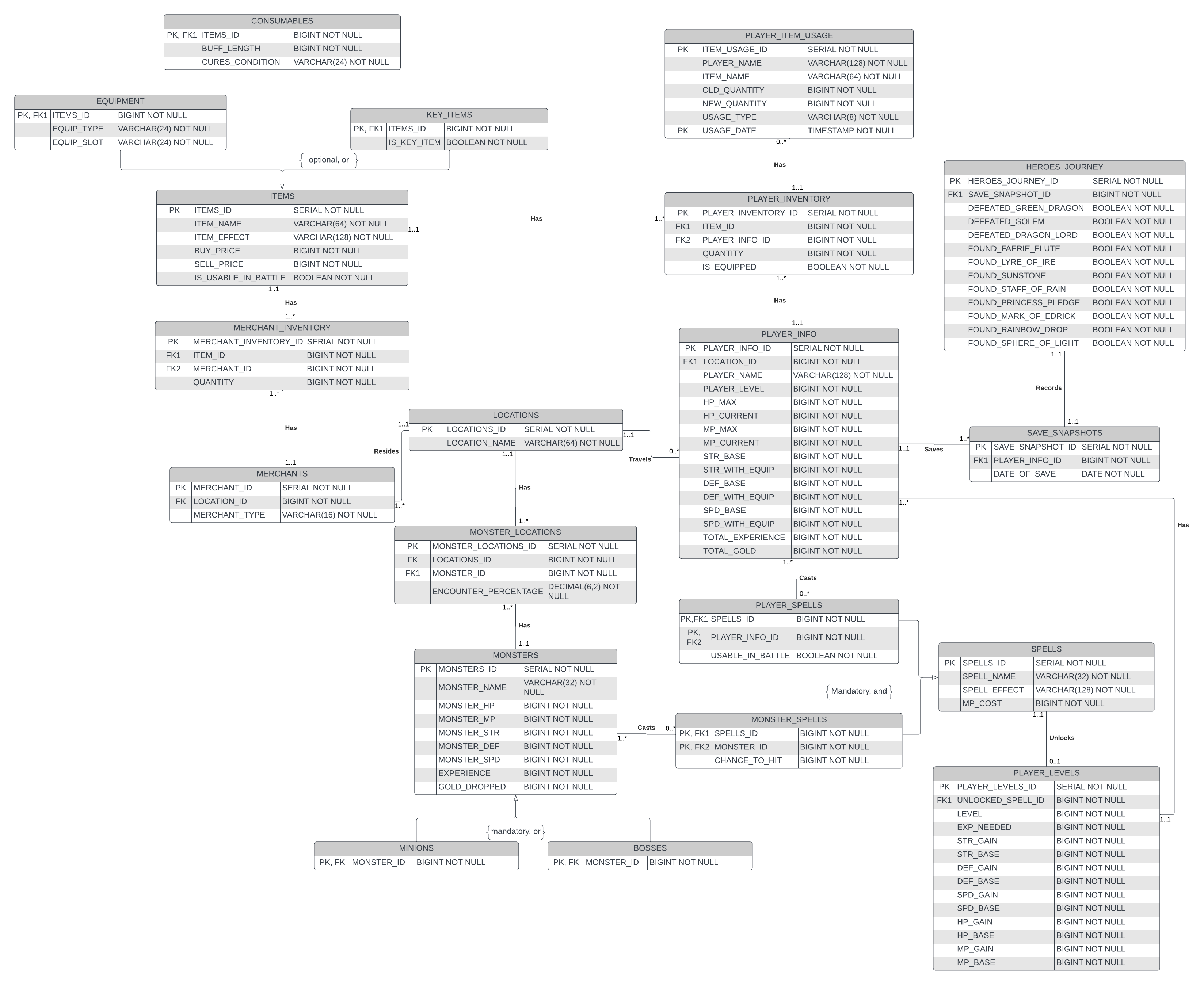
PLAYER\_INFO is related the SPELLS entity through the PLAYER\_SPELLS entity. This entity contains records with a PLAYER\_INFO\_ID and a SPELL\_ID to signify that a PLAYER\_INFO entity may have access to multiple SPELLS as they progress through the game. The PLAYER\_SPELLS entity is a subtype of the SPELLS supertype which contains the common information about the spell like its name, cost, and effect. The other subtype MONSTER\_SPELLS has a similar function to PLAYER\_SPELLS however instead of referencing a PLAYER\_INFO entity, it references a MONSTERS entity. The MONSTERS entity contains information about the various monsters the player will battle through the course of the game and it has two subtypes: MINIONS and BOSSES. MINIONS will be the normal monsters the player encounters in the wild or in a dungeon, however the BOSSES subtype relates to MONSTERS entities that guard certain key items, or are relevant to the story in some capacity. Since the player travels to different LOCATIONS to explore or progress the story, they will encounter MONSTERS in these LOCATIONS which is represented in the MONSTER\_LOCATIONS entity. This entity exists because multiple monsters can exist in a single LOCATION and certain MONSTERS can be found in multiple LOCATIONS.

Some LOCATIONS, like towns or castles, contains no MONSTERS but instead contain MERCHANTS which will buy and sell items with the player. Merchants that exist in these LOCATIONS have certain ITEMS which they trade and this information is represented in the MERCHANT\_INVENTORY entity. The records of this entity have a reference to the MERCHANT entity as well as the ITEMS entity, in addition to other information. The ITEMS entity contains the more global item information including the item’s name, description, effect, buy/sell prices, and other information. ITEMS is also a supertype for three different subtypes: EQUIPMENT, CONSUMABLES, and KEY\_ITEMS. The first two subtypes, EQUIPMENT and CONSUMABLES can be found with merchants, while the other type KEY\_ITEMS are found in specific locations or are awarded after defeating a monster boss. Once players purchase, or find items a record in the PLAYER\_INVENTORY is created. This entity is like the MERCHANT\_INVENTORY as it contains references to the ITEMS entity and the PLAYER\_INFO entity (instead of MERCHANT). The player can then use some of these items in battle or even equip certain items like weapons and armor.

The PLAYER\_LEVELS entity is referred to after a player successfully completes a battle. The player’s total experience is compared to the various levels to determine if the player has leveled up. If so, the attributes in the PLAYER\_INFO will increase a corresponding amount and the player may also unlock a SPELL. If a SPELL is unlocked at, they newly achieved level, a record in the PLAYER\_SPELLS entity will be created and the player will have access to that newly acquired spell.

PLAYER\_ITEM\_USAGE is the new history table created for Iteration 6. It tracks changes of the quantity field in a PLAYER\_INVENTORY record, and could be useful for some type of “resupply” feature where an item shop can offer to purchase items used in the field. New records are inserted into the PLAYER\_ITEM\_USAGE using a function which is called by a trigger. This trigger calls the function after the update to a PLAYER\_INVENTORY record. The history table has a USAGE\_TYPE field that helps differentiate how an item’s quantity changed. The values CONSUMED and BOUGHT that an item was consumed by the player or bought in a store respectively, however the third value EMPTIED indicates that the quantity changed from 1 to 0 and as an additional action the trigger function will delete the associated PLAYER\_INVENTORY record.

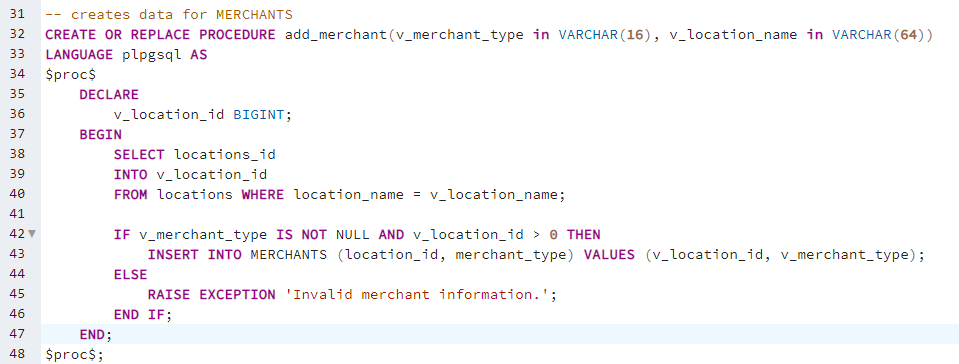
I believe that my DBMS design properly captures all the necessary data need to play this game and that the entities are structured and related to each other in a way that eliminates data redundancy. Each entity has a clear and obvious purpose, join tables have been created to ensure that the many-to-many relationships are represented properly and in an organized way. I also think that my use of supertypes and subtypes allows for a clear diversification of data in a logical and structured pattern.



# Stored Procedure Execution and Explanations

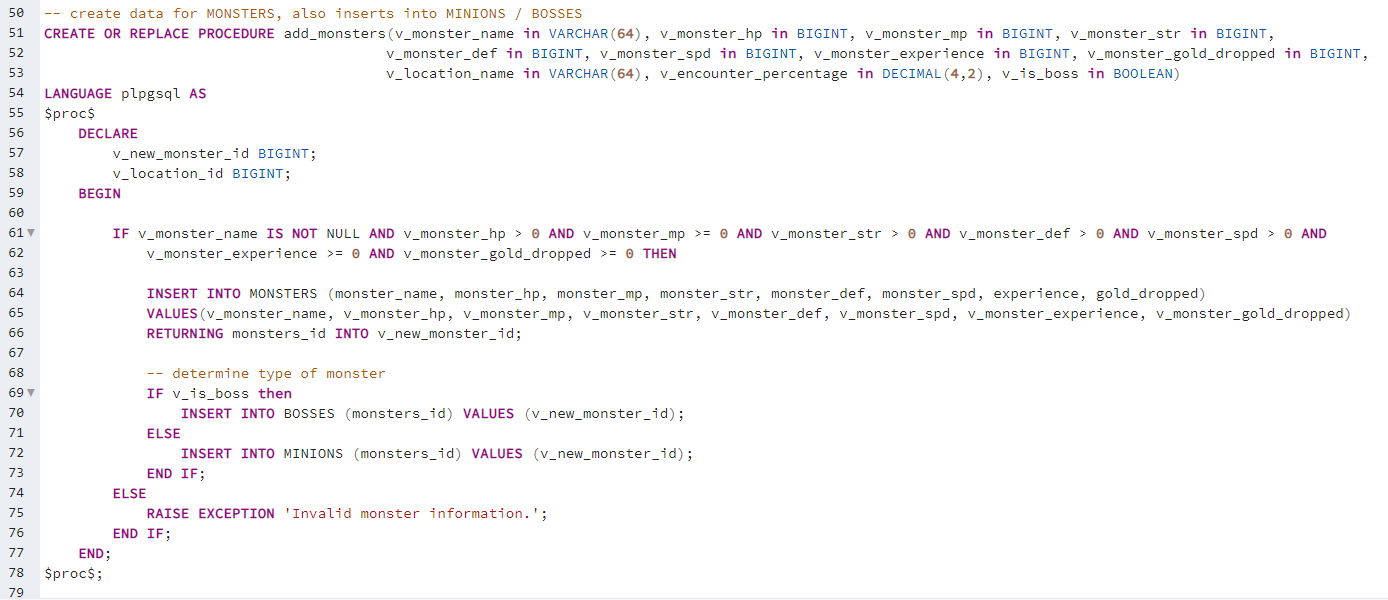
I have provided two screenshots each of a different stored procedure to show how I created and designed them. All my other stored procedures can be found in the **procedures.sql** file that I have provided along with this document.

For my stored procedures, all my procedure parameters have a “v\_” appended before them to help me distinguish them from similarly titled field names in my queries. I also apply this standard to any variables I declare in my procedures for the same reason.



The stored procedure shown above is used to insert data into the MERCHANTS table. It takes two VARCHAR arguments, one for the MERCHANT\_TYPE and another for the LOCATION\_NAME. The second parameter is used to determine its relevant LOCATION\_ID value to satisfy the foreign key constraint on the MERCHANTS table with the LOCATIONS table.

This stored procedure contains a validation for the v\_merchant\_type parameter and the procedure variable v\_location\_id to ensure that values have been provided and that the location exists in the database. If these validations pass, then data is inserted into the database, otherwise an error message is returned.



The stored procedure shown above is a litte more complicated than the previous example due to its many parameters as well as its multiple inserts. This procedure inserts data into the MONSTERS table and it also inserts data into the MINIONS or BOSSES subtype tables depending on the parameter values provided. Similar to the first example, the procedure parameters are checked to ensure that no NULL values or negative number values are provided for the relevant column types. Then one of the parameters is used to determine if the MONSTER record that will be inserted belongs to the MINIONS or MONSTERS table. Once determined the record is inserted into that table.

All of my other tables in my design use procedures to insert data and they can be found in my **procedures.sql** file that is provided along with this document.

# Question Identification and Explanations

**Question 1 – What items are available for sale by merchants at a particular location?**

This query provides a high-level summary of the different merchants in a particular location and what each of them is selling. This query is useful to the organization since this information can be manipulated by the application to create the various merchant instances and populate their inventories using only one query as opposed to potentially making a query for each merchant. This query utilizes the LOCATIONS, MERCHANTS, MERCHANT\_INVENTORY, and ITEMS tables.

**Question 2 – What items do the player have currently equipped?**

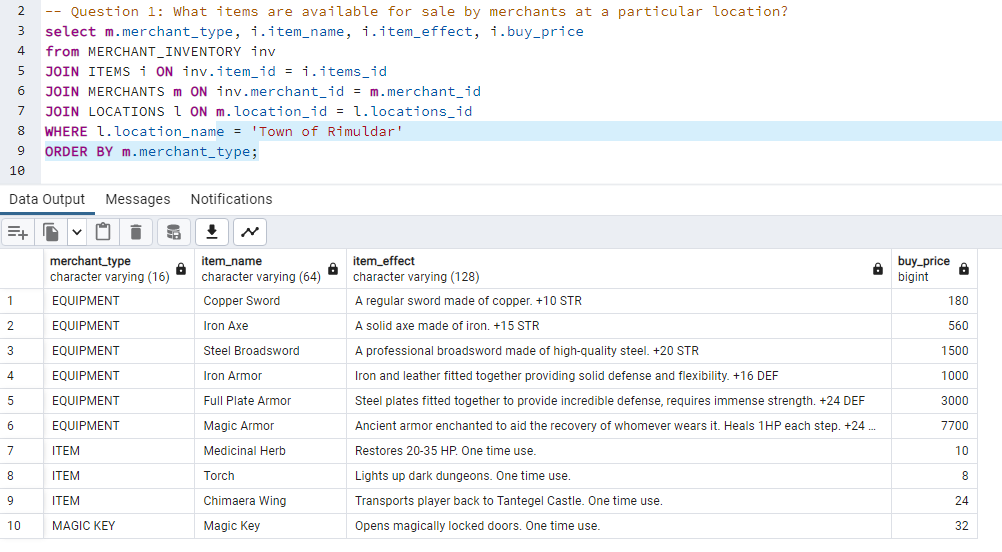
This query is useful because the player needs to have the ability to what equipment in their inventory is currently equipped. This is necessary to know since it influences the players final stats in battle (STR, DEF, SPD) and some equipment also has special effects, which are only applicable if equipped. This query will utilize the PLAYER\_INFO, PLAYER\_INVENTORY, ITEMS, and EQUIPMENT tables.

**Questions 3 – What MINIONS and BOSSES are in a particular location and any related sub-locations it may have?**

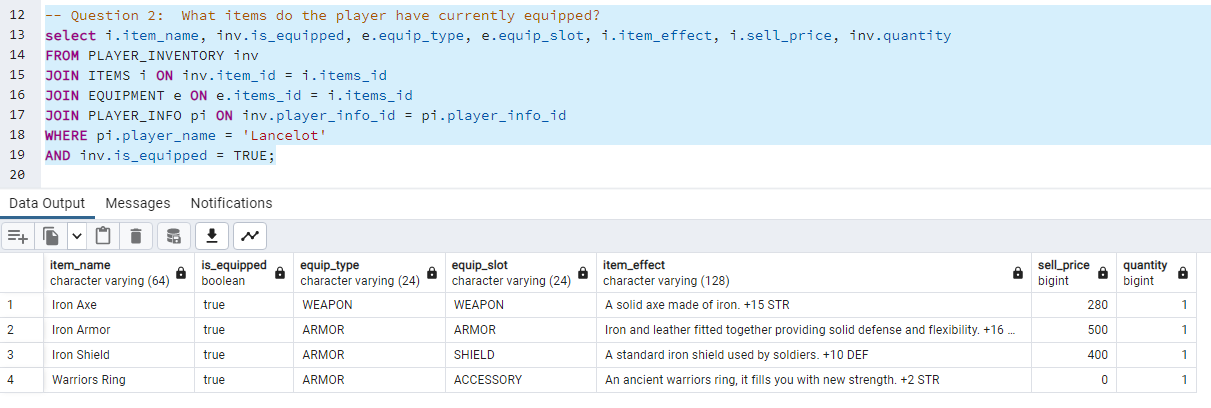
This query provides a list of minions and bosses in a general area in order to initialize a location or dungeon when the player enters it. By providing a smaller subset of data, any processes or methods that need this data can avoid multiple complicated queries. This query utilizes the LOCATIONS, MONSTERS, its sub-type tables (MINIONS and BOSSES), and MONSTER\_LOCATIONS.

# Query Executions and Explanations

**Question 1 Execution**

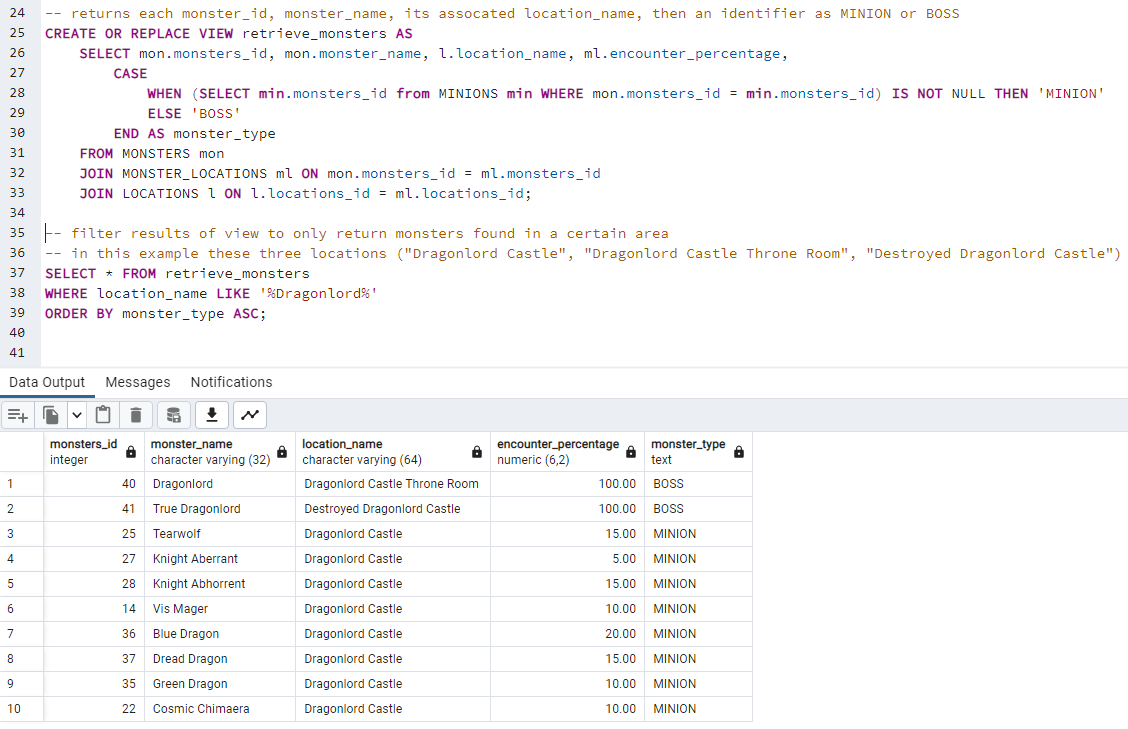
This query joins four tables as requested and displays all items for sale by all merchants in a specific location. The location is determined by provided a value for LOCATIONS.location\_name which in this case is the value “Town of Rimuldar”. The result of this query is the items sold by the Equipment shop, the Item shop, and the Magic Key shop.

The query returns the merchant type, which indicates which shop sells the item, the item\_name, the item’s effect, and its buy price. Changing this value will yield different results and can be displayed by changing the value to either ‘Town of Tantegel’ or ‘Town of Kol’.

**Question 2 Execution**

This query joins the supertype ITEMS with one of its subtypes EQUIPMENT in addition to the PLAYER\_INVENTORY and PLAYER\_INFO tables to return all of the items currently equipped by the player. This list of items is pulled from the PLAYER\_INVENTORY which contains all of the ITEMS that the player currently has. The player is determined by providing a value for the PLAYER\_INFO.player\_name value which in this case is ‘Lancelot’. Changing this value will yield different results and can be displayed by changing the value to either ‘Gawain’ or ‘Newbie’.

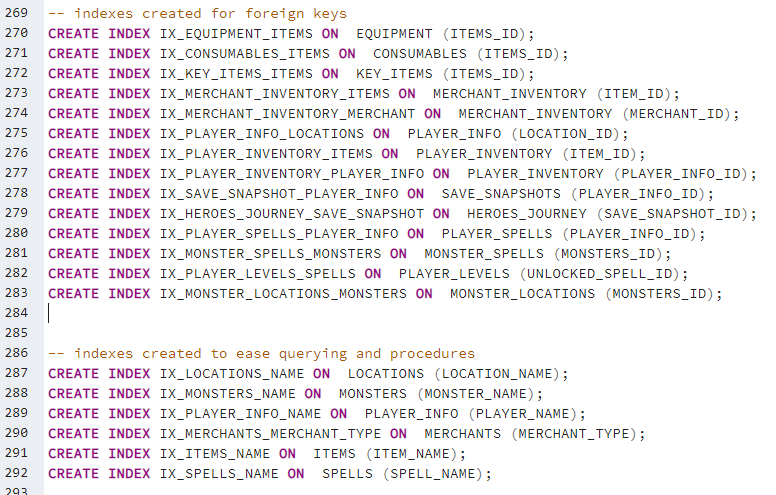
**Question 3 Execution**



This view returns a list of all monsters with their ID and name values, as well as their locations with encounter percentages, and an indicator if they are a MINION or a BOSS. Calling this view and filtering the results like the screenshot, allows a user to return a list of specific monsters related to a location, or multiple locations with a similar name. In the screenshot the latter case is demonstrated with the locations named LIKE “Dragonlord” which yields three different locations: Dragonlord Castle, Dragonlord Castle Throne Room, Destroyed Dragonlord Castle. For each location, the encounter percentages should all sum to 100%.

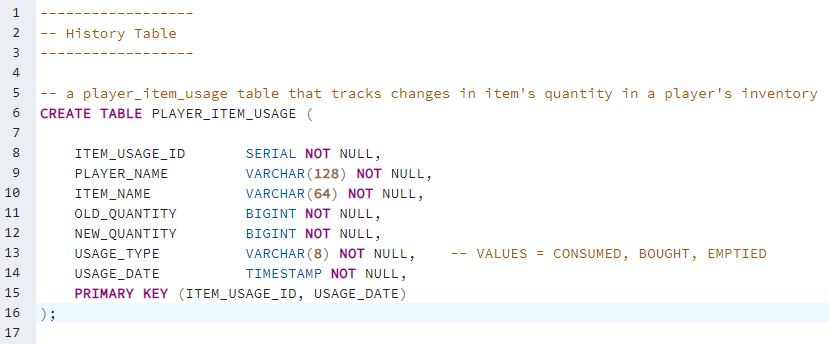
# Index Identification and Creations

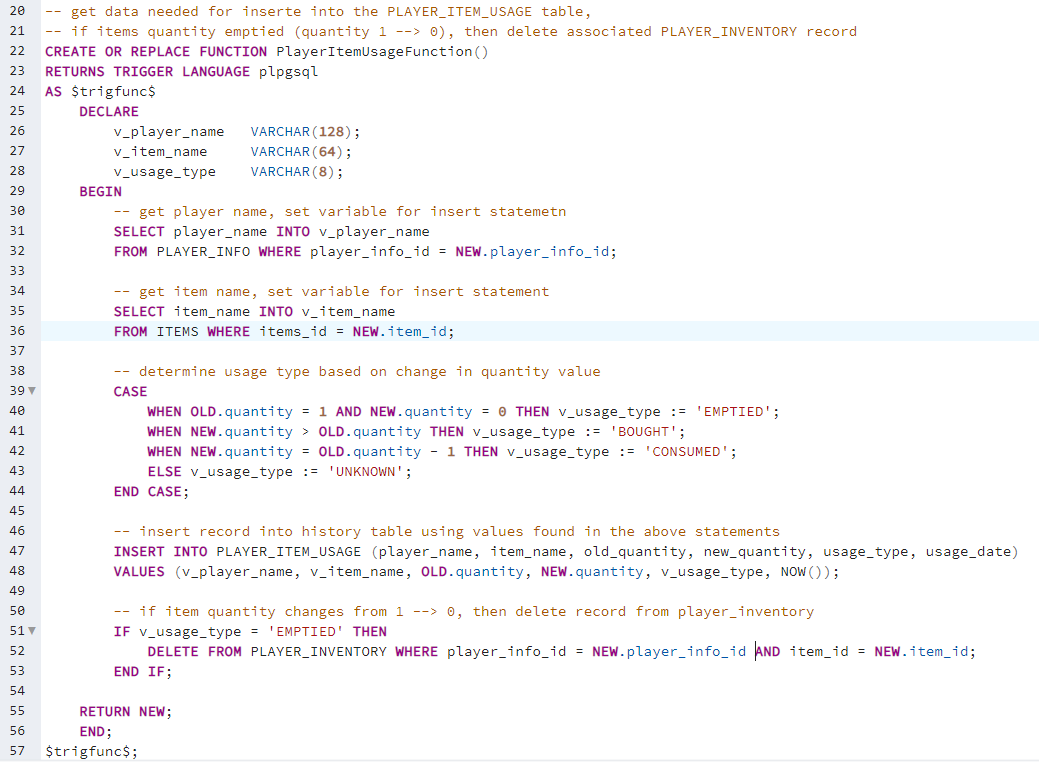
In the screenshot below, I show the various indexes that were created to aid the performance of my procedures and queries. The first section of indexes was created to be applied to the various foreign keys for each of my tables. This is to ensure that complicated queries with JOIN operations on my tables can be performative. The next section of indexes is for fields commonly used in my stored procedures and in the queries for my questions. These fields are used in WHERE clauses in my procedures and question queries to filter results and the indexes allow these queries and procedures to return results with optimal performance times.

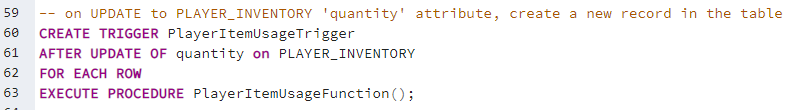


# History Table Demonstration

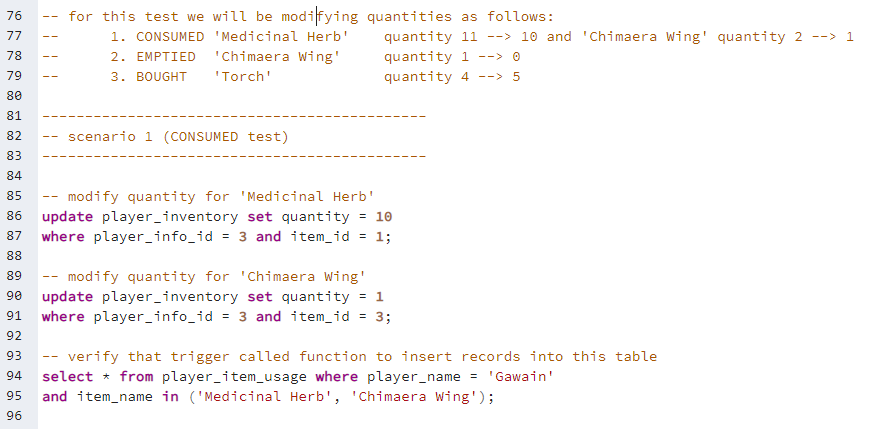
As stated previously in the Full DBMS Physical ERD section, the PLAYER\_ITEM\_USAGE is the new history table created for Iteration 6. It tracks changes of the quantity field in a PLAYER\_INVENTORY record. New records are inserted into the PLAYER\_ITEM\_USAGE using a function which is called by a trigger. This trigger calls the function after the update to a PLAYER\_INVENTORY record. The history table has a USAGE\_TYPE field that helps differentiate how an item’s quantity changed. The values CONSUMED and BOUGHT that an item was consumed by the player or bought in a store respectively, however the third value EMPTIED indicates that the quantity changed from 1 to 0 and as an additional action the trigger function will delete the associated PLAYER\_INVENTORY record.



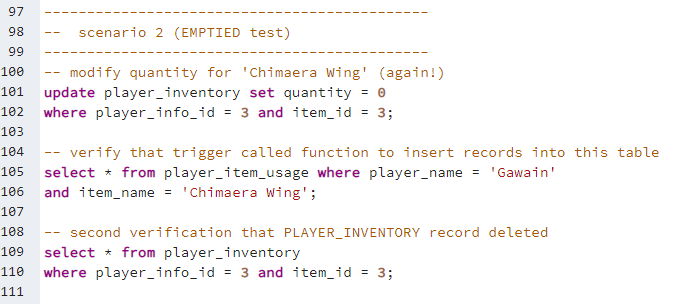


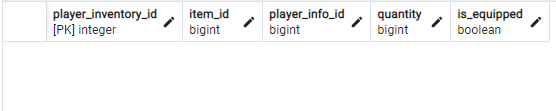


After creating the history table, the trigger, and the trigger function I created three test scenarios. These scenarios verify that the trigger function captures the three USAGE\_TYPE values (CONSUMED, BOUGHT, and EMPTIED). Additionally, the EMPTIED scenario also tests that the original record is deleted from the PLAYER\_INVENTORY tables. The specifics for each test can be seen at the top of the screenshot below.

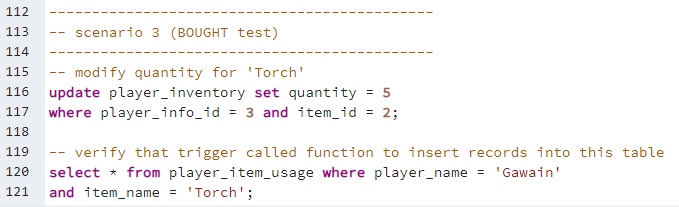


We can see that for the results of this test that two new records are inserted into the PLAYER\_ITEM\_USAGE table: one for the Medicinal Herb and another for the Chimaera Wing. Both records have the appropriate USAGE\_TYPE value, and the quantity values (new and old) are tracked.





For scenario 2 (screenshots above), we see that a second record for the ‘Chimaera Wing’ item has been added to the ITEM\_USAGE\_TABLE showing that the quantity has been reduced from 1 to 0. The first record is from the previous test scenario and shows us that the history table is providing historical data useful to the user. The second screenshot shows us that the original PLAYER\_INVENTORY record has now been deleted since the item’s quantity was reduced to 0.



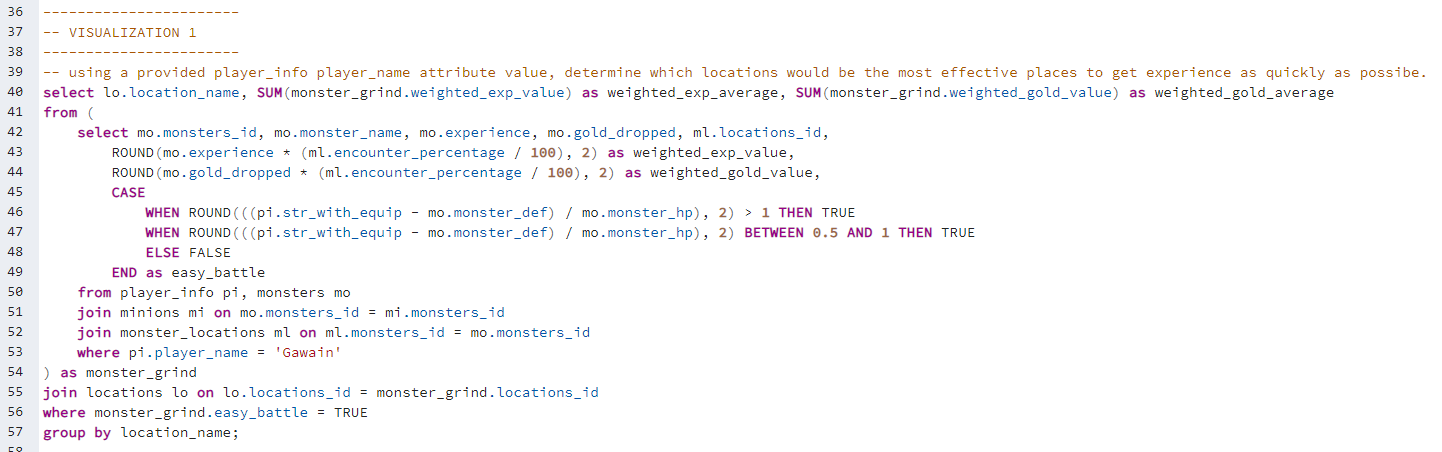


The final test scenario, a new record is inserted into the PLAYER\_ITEM\_USAGE table with a positive increment to the item’s quantity. In the game this would occur when the player buys an item or one is found in the wild. Similar to the first test scenario, no additional logic occurs in this scenario since the items still have a non-zero quantity.

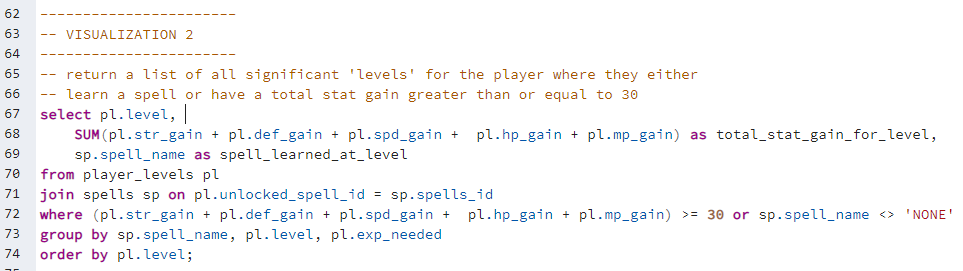
This history table is useful for tracking changes to the player’s inventory and to provide an automated way to clean up the PLAYER\_INVENTORY table since it change rapidly during gameplayer, and since it has the potential to become the largest table in this DBMS. Implementing ways to limit or remove data will help improve performance for any queries that rely on this table.

# Data Visualizations

The first data visualization I created was a way for a player to determine which areas would be the best to rapidly gain experience or gold. In gaming terms, this process is called “grinding” and refers to repetitively battling or performing some action to improve your character to level up in the most efficient way possible. The query I developed joins the MONSTERS (specifically MINIONS), PLAYER\_INFO, LOCATIONS, and MONSTER\_LOCATIONS together to return all location names in addition to a weighted average of experience and gold for each location.

The weighted averages were determined by first determining if the player battling that monster was easy and could be finished with one or two actions. This data point was obtained by subtracting the monster defense value from the player’s strength with equip value and then dividing that result by the monster’s health. Values of this equation that were greater than 1 and between 0.5 and 1 meant that the player could finish the battle quickly. Monsters that passed this threshold had their ‘experience’ multiplied by the ‘encounter percentage’ value aggregated with the SUM() function. By changing the ‘player\_name’ value in the inner query’s WHERE clause, different results can be obtained since it is dependent on the player’s ‘strength with equip’ value. The screenshots below are the results of three different player name values: Newbie (Level 1 player), Lancelot (level 10 player), and Gawain (level 25 player).

The second data visualization is meant to provide the player with a list of significant levels. A level is deemed significant if a spell is learned or if the total sum of stats gained at the level surpasses a threshold of 30. This data can be useful if a player is struggling when they enter a new area or if they have trouble defeating a boss. The information provided in this query should motivate them to find a suitable location to level up quickly in order to gain the stats or spells and attempt the new challenge.



# Summary and Reflection

My database is for a videogame like the 1989 videogame DragonQuest which will contain information about the HERO, MONSTERS, ITEMS, SPELLS. It will also record snapshots of the current state of the game when the player chooses to “save” the game and stop playing. The tables mentioned above contain static reference data that will be used by an application, but more dynamic tables will be created and used when the player “saves” the game.

This iteration was a lot of fun trying to figure out the various queries behind my visualizations and to try and present something meaningful and interesting to the user that might not be obvious to them. It was a great exercise in trying to apply all the lessons learned in this class with the database that I created. I am very glad that I took this class since it really solidified a lot of previous knowledge and experience that I had, but it gave me a look at the complete process of creating a database which I did not always experience in my professional work. I am very excited to take these skills with me to the real world.

**If you wanted to create my database on your machine, please run the scripts in the following order:**

1. create\_tables.sql
2. procedures.sql
3. data\_insertion.sql
4. (optional) questions.sql

**\*\* If dropping and recreating multiple times, please remember to drop the view created in the questions.sql file \*\***