IE 6700 CRN 19106 Data Management for Analytics Project Report

Group Number: 9

Student Names: Niresh Subramanian and Raaga Sindhu Mangalagiri

Executive Summary

The goal of this study was to set up the conceptual model for Pokémon UNITE, set up a relational database, and analyze the data to help the developers & publishers of the game make effective business decisions. The ideas presented in the study would help the business drive conversations that support retention and user experience strategies.

Relevant entities and relationship types that help address the business problem at hand were chosen and a simplified conceptual model was designed and represented using EER and UML diagrams. The conceptual model was then mapped to a relational model and all the essential relations were normalized. Sample data from the game's official website was scrapped for the analysis. The relational model was implemented using MySQL and MongoDB. SQL and NoSQL queries that help address the problem were drafted and run. Additionally, a connection between the database and the Jupyter environment was set up to leverage python and create visualizations that would help the business.

Introduction

Pokémon UNITE is a free-to-play, multiplayer online battle arena video game developed by TiMi Studio Group and published by The Pokémon Company for Android and iOS and by Nintendo for the Nintendo Switch. As of April 2022, the game has been downloaded over 70 million times across all platforms.

The matches consist of two teams, each with 5 players. At the end of the battle, the team with the most points wins. When players keep battling, they earn in-game rewards that will unlock more Pokémon, as well as outfits for their Trainer (character) and outfits for their Pokémon. They'll also be able to unlock rewards with Aeos gems, which can be purchased with real-world money.

Despite the tremendous success of the game in terms of downloads, the number of active players in the game is slowly declining. Additionally, the game has behaved poorly in terms of total gross revenue. According to several sources on the internet, it can be seen that the Revenue per Download (RPD) remains flat. The disparity between downloads and revenue, the fact that revenue is not seeing constant growth, and that seem so closely linked to downloads is quite concerning. It hints that the game is having trouble transforming the huge flow of installs into loyal players.

Problem Statement

The Pokémon UNITE team invests thousands of dollars in building and maintaining a free-to-play game containing millions of users. Poor customer experience leads to low retention rates and lower monetization opportunities, negatively impacting the business' return on investment (ROI) and profit margins. A successful solution would be to continuously improve the game mechanics in subsequent patch updates by analyzing game data and player behavior.

Conceptual Data Modeling

As a first step, the conceptual data model reflecting the game's mechanics and model is set up using EER as provided below. A detailed explanation of the model is as follows:

- The DOB and email address are required from the user when he/she decides to sign up as a player. Each email is considered a different user and a player can only be created by signing up with exactly one email address.
- When a user signs up as a player, the user is expected to create a user name for the player, and a player ID is assigned upon successful creation
- Post signing up, every time the user logs into the game, the login and log-out times are recorded and a session ID is generated for the player. Multiple players can have the same session IDs as it is recorded in a sequential manner for each player. Additionally, the player's season rank is also kept track of
- The game contains several Pokémon identified by the Pokémon's unique ID. Additionally, each Pokémon has an ability that gives it buffs when a condition is met. They have values for stats like HP, attack, defense, special attack, special defense, and speed. Depending on the stats, the Pokémon are categorized into classes/roles (Attacker, Defender, Support, Speedster, All-rounder). Further, they have a list of moves they learn as they level up in the game
- Pokémon can be given out as a reward during events. Each event is identified by its unique ID, has a name, and runs for a limited time interval (Marked by an event start and end date)
- A user can use any Pokémon (In a team of 5, no two players can pick the same Pokémon) and enter a match. There are three types of matches in the game - quick, standard, and ranked. A quick match happens for a duration of 5 minutes and is suitable for users seeking quick and fun games. The user can select from various maps. Standard and ranked matches happen for a duration of 10 minutes. For ranked matches, the opponents and allies are pooled in together if they belong to the same rank points range and when a player is promoted to a higher rank, several rewards are given out. For every match that happens, a unique match ID is assigned, to which 5v5 battles are held.
- When entering a battle, a Pokémon can be given a battle item that has a specific gimmick and a cooldown. In addition to this, a Pokémon can be equipped with a held item that enhances its stats and provides buffs in the game. Apart from battle and held items, other items are also available and help the player in their journey. Some of the items are given away as rewards from events and they are all sold at the store
- All the items and Pokémon are sold at the Store which has the name as its unique identifier. All Pokémon licenses (required to take a Pokémon into battle) can be purchased in the Licence store and the items are sold at the Accessory store. Additionally, an outfitting store is also available where the user can purchase outfits to customize their character or for their Pokémon
- A build refers to Pokémon along with its held and battle items. The player can customize a Pokémon's build and participate in matches. For a match to occur, at least one player should be

available (As bot games can be encountered in the initial stages to teach the player the basics) and a total of 10 players (5 on each side) choosing their own set of builds are added to the match. The results of the battle are recorded by assigning a battle ID and the match result (Win, loss, and surrender) is recorded corresponding to each match ID, player ID, and build ID

The EER diagram representing the conceptual model is provided on the next page for your reference. The following are some of the constraints that cannot be represented in the EER diagram:

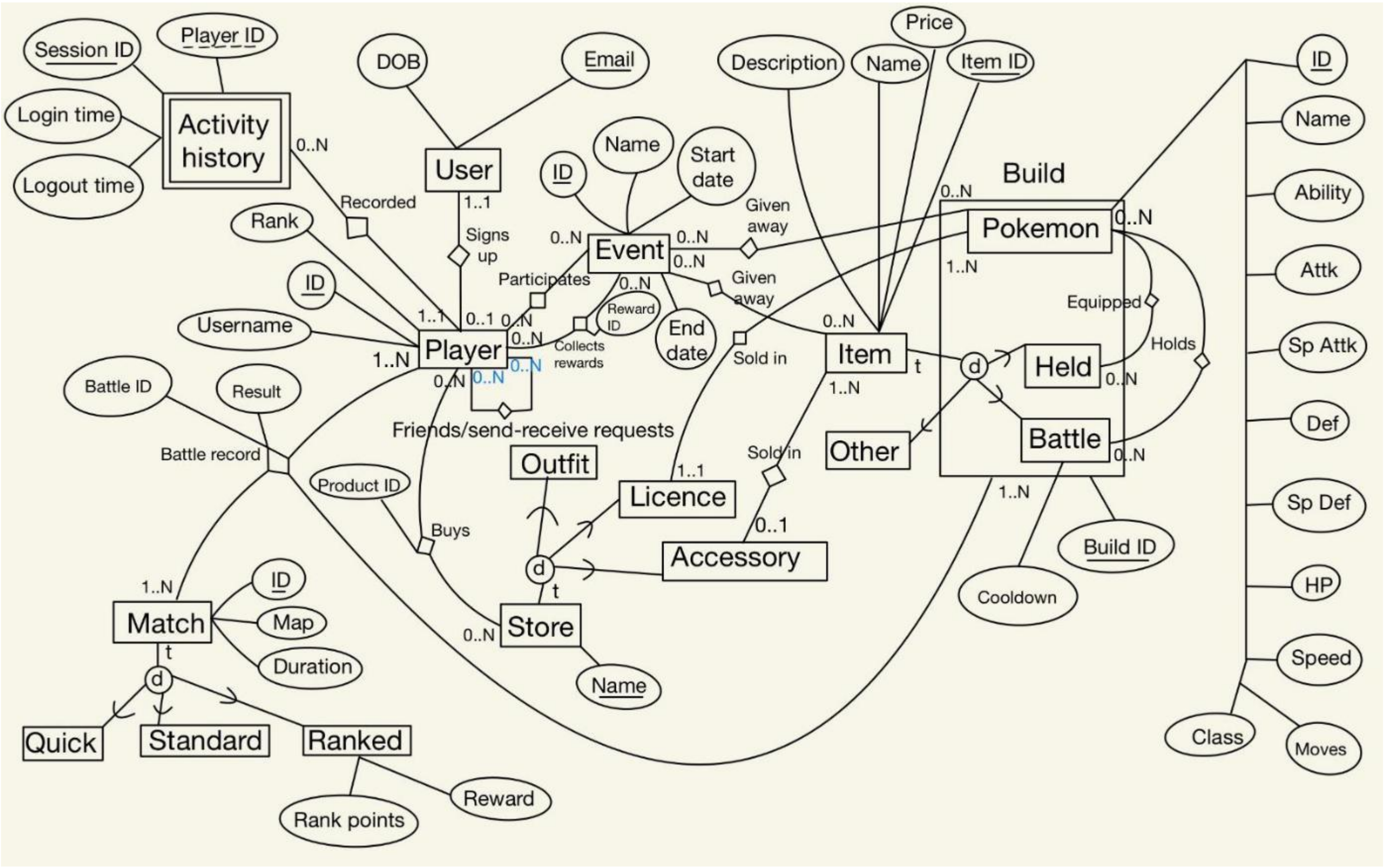
1. A Pokémon can be equipped with a maximum of 1 held item
2. Two players in the same team cannot select the same Pokémon for a given match

The chosen business model involves both transactional and reference data. Examples of transactional data are the user & player database, the activity of players, event data, and a battle record that documents the outcome of each match along with all the players and builds involved. Similarly, examples of reference data are Pokémon data, item data, and store data.

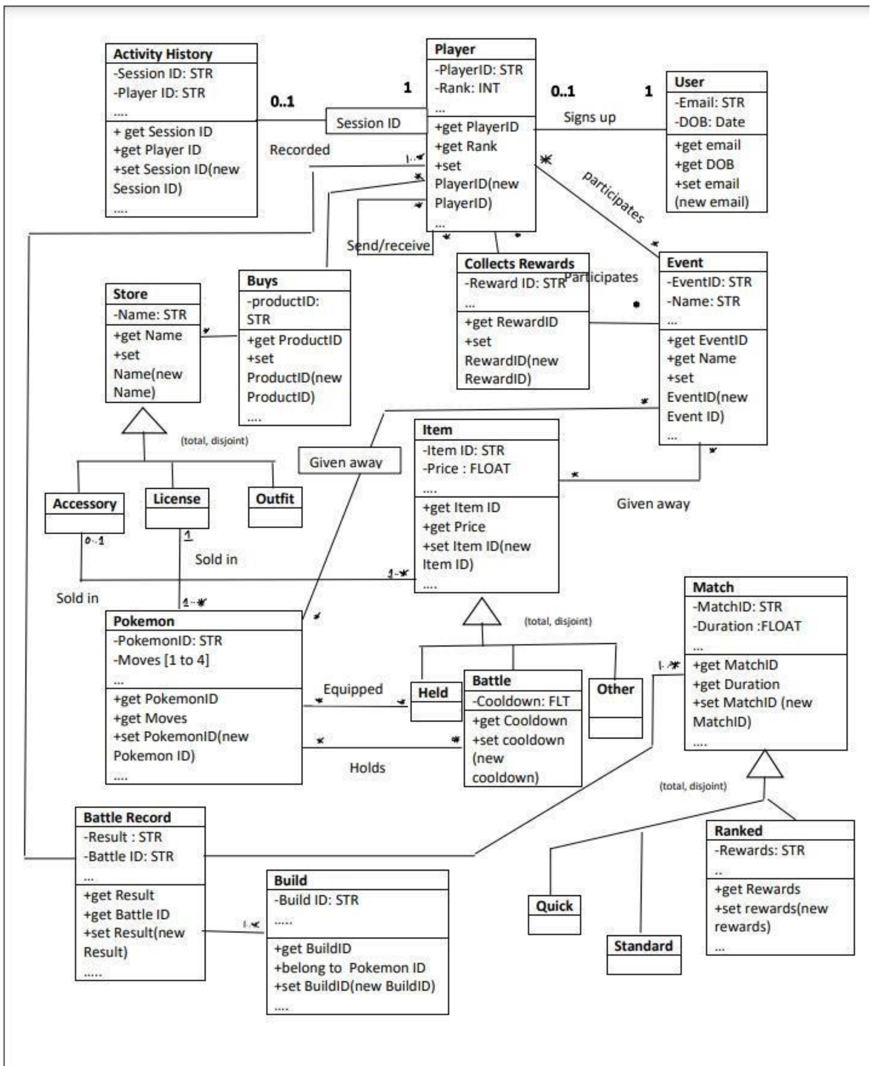
The following are the **limitations of the EER model:**

- Temporal constraints cannot be modeled. These are some of them:
    - Ranked and quick battles are not playable till a player reaches a certain rank
    - A Pokémon's move sets and stats might change over time
    - The number of items a Pokémon can hold gradually increases from 0 to 3 depending on the player's rank
- Consistency among multiple relationship types cannot be enforced. Here are some of the limitations:
    - A player cannot buy a Pokémon that he gets as a reward through events
    - A player cannot send or receive requests to/from themselves
- Attribute type domains or functions cannot be specified. The following domain constraint is a limitation in the above model:
    - A Pokémon cannot hold more than 1 held item in a battle. Similarly, a Pokémon can only hold a maximum of one battle item in a battle

EER Diagram



UML Diagram



Limitations of the UML model:

- It is not always clear who benefits from a UML diagram. As it is a conceptual model presented to business users, users who are not familiar with technical aspects of the UML language might not understand/might feel overwhelmed
- As a documentation tool, it requires too much upfront planning
- Looking at a software scope in a UML diagram can lead to software project stakeholders over-analyzing problems, as well as cause people to lose focus by spending too much time and attention on software features

Mapping Conceptual Data Model to Relational Model

Refer to the relations below, created to map the conceptual data model to the relational model. While normalizing, the steps have been mentioned if required. Cardinalities that cannot be enforced have been called out. The primary key(s) are underlined and the foreign key(s) are italicized. Only if the foreign keys are NOT NULL, it has been called out. Else the foreign key is Nullable. Similarly, all primary keys are non-nullable. All the below relations have been normalized to at least 3.5 Normal Form

User (<u>Email</u>, DOB)
Player (<u>ID</u>, Rank, *Email*)

- Email is the foreign key from the 'User' relation. Email is NOT NULL as each player is signed up by exactly one user
- The maximum cardinality of 1 on the Player side in the signs-up relationship type cannot be enforced as multiple players can have the same email

Username (<u>*Player ID*</u>, <u>Username</u>)

- As the username is also unique, and is also a candidate key, we must bring it to a 3.5 normal form
- Player ID is both the primary and foreign key

Activity History (<u>Session ID</u>, <u>*Player ID*</u>, Login Time, Logout Time)

- Here Player ID is also a foreign key from the Player relation and is NOT NULL

Match (<u>ID</u>, Map, Duration, Type, Rank Points, Reward)

- The entire specialization has been modeled as one relation to address the total disjointedness. Subclass values will be part of the Type domain. Note that, rank points and rewards will be NULL for match types other than ranked

Store (Name)
Outfit (Name)
License (Name)
Accessory (Name)

- The specialization has been modeled by creating separate relations for subclasses and the superclass as some subclasses interact with other entity types
- The total-disjointedness constraint cannot be enforced by doing this. In order to enable this constraint, we will have to model the entire specialization as one relation – Store (Name, Type). Subclass values will be part of the Type domain

Item (Item ID, Name, Price, Description, *Accessory Store Name*)

- Accessory Store Name is a foreign key and can be NULL as some items might not be sold in the store
- The minimum cardinality of 1 on the Item side in sold in relationship type cannot be guaranteed

Held Item (Item ID, Name, Price, Description)
Battle Item (Item ID, Name, Price, Description, Cooldown)
Other Item (Item ID, Name, Price, Description)

- The specialization has been modeled by creating separate relations for subclasses and the superclass as some subclasses interact with other entity types
- The total-disjointedness constraint cannot be enforced by doing this. In order to enable this constraint, we will have to model the entire specialization as one relation – Item (Item ID, Type, Name, Price, Description, Cooldown). Subclass values will be part of the Type domain. Cooldown will be NULL for every item type other than the battle item

Event (ID, Name, Start Date, End Date)
Pokémon (ID, Name, Ability, Attk, Sp Attk, Def, Sp Def, HP, Speed, Class, *Licence Store Name*, Price)

- Licence store name is a foreign key from the accessory store relation. The license store name is NOT NULL as all Pokémon are sold in exactly one license store
- Minimum cardinality of 1 on the Pokémon side of sold in relationship type cannot be enforced

Pokémon – Move (*Pokémon ID*, Move set)

- As a Pokémon has multiple move sets (total 4 combinations), and 1 NF requires the attribute types to be atomic and single-valued, a new relation is being created

Build (Build ID, Pokémon ID, Held Item ID, Battle Item ID, Move set)

- A Pokémon can hold at most 1 held item and at most 1 battle item. This cannot be enforced
- Note that the aggregation has been given a build ID. This is the primary key. Each combination of Pokémon ID, held item ID, and battle item ID is represented by a unique build ID value. Hence, the 3 columns (Pokémon ID, Held Item ID, and Battle item ID) haven't been included in the primary key – as it is normally done in case of mapping an EER aggregation

Player – Event – Participates (*Player ID*, *Event ID*)
Player – Event – Rewards (*Player ID*, *Event ID*, Reward ID)

- Here reward ID is a mapping key and corresponds to Pokémon ID values if the reward is a Pokémon and Item ID values if the reward is an item

Player – Friend (*Player ID*, *Other Player ID*)

- The other player ID is also a player ID from player relation. They are both foreign keys as well and represent the friends/send-receive request relationship type

Player – Store – Buys (*Player ID*, *Store Name*, Product ID)

- Here product ID is a surrogate key and corresponds to Pokémon ID or Item ID depending on what the player purchases

Event – Pokémon – Given Away (*Event ID*, *Pokémon ID*)
Event – Item – Given Away (*Event ID*, *Item ID*)
Pokémon – Held Item (*Pokémon ID*, *Held Item ID*)

- The constraint that a Pokémon can be equipped with a maximum of 1 held item cannot be enforced

Pokémon – Battle Item (*Pokémon ID*, *Battle Item ID*)

- The constraint that a Pokémon can be equipped with a maximum of 1 battle item during a match cannot be enforced

Battle Record (Battle ID, *Match ID*, *Build ID*, *Player ID*, Result)

- Battle ID uniquely identifies the battle for each match, build, and player combination
- Match ID, Build ID and Player ID are the foreign keys and they are all NOT NULL

Implementation of Relation Model via MySQL

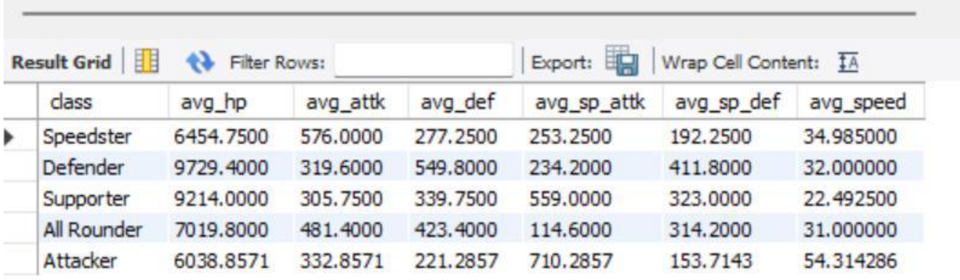
As shown in the screenshots below, the relational model has been implemented in MySQL and MySQL Workbench is used to query the database. Sample data has been populated in all datasets and sample queries have been presented along with the sample output.

Query 1

Analytical Purpose: To get an understanding of the average stats of Pokémon by the class. This can be used to adjust the stats of characters to balance the game when a new character is released

```
select class, avg(hp) as avg_hp, avg(attk) as avg_attk, avg(def) as avg_def, avg(sp_attk) as avg_sp_attk, avg(sp_def) as avg_sp_def, avg(speed) as avg_speed
from game.pokemon
group by class;
```

Output:



class	avg_hp	avg_attk	avg_def	avg_sp_attk	avg_sp_def	avg_speed
Speedster	6454.7500	576.0000	277.2500	253.2500	192.2500	34.985000
Defender	9729.4000	319.6000	549.8000	234.2000	411.8000	32.000000
Supporter	9214.0000	305.7500	339.7500	559.0000	323.0000	22.492500
All Rounder	7019.8000	481.4000	423.4000	114.6000	314.2000	31.000000
Attacker	6038.8571	332.8571	221.2857	710.2857	153.7143	54.314286

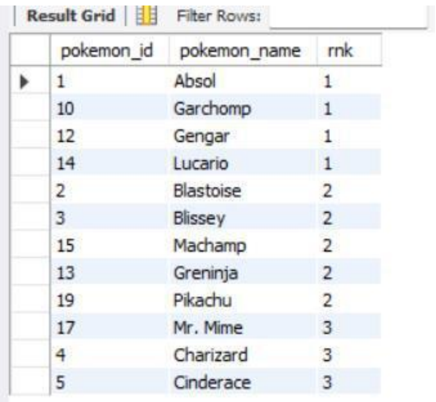
Query 2

Analytical Purpose: Identifying the top 3 highest selling Pokémon in the shop to get an understanding of what kind of Pokémon are customers willing to spend money on.

```
with base as
(select product_id, count(*) as cnt
 from game.player_store_buys
 where store_name = "Unite License Committee"
 group by product_id)

select a.product_id as pokemon_id, b.name as pokemon_name, a.rnk
from (select product_id, dense_rank() over(order by cnt desc) as rnk from base) a
left join game.pokemon b
        on a.product_id = b.id
where a.rnk<=3;
```

Output:



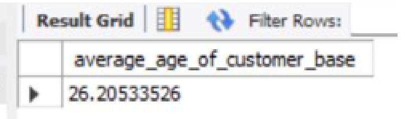
pokemon_id	pokemon_name	rnk
1	Absol	1
10	Garchomp	1
12	Gengar	1
14	Lucario	1
2	Blastoise	2
3	Blissey	2
15	Machamp	2
13	Greninja	2
19	Pikachu	2
17	Mr. Mime	3
4	Charizard	3
5	Cinderace	3

Query 3

Analytical Purpose: Determining the average age of the game's customer base

```
select avg(age) as average_age_of_customer_base
from (select email, datediff(CURDATE(), DOB)/365 as age from game.user) a;
```

Output:

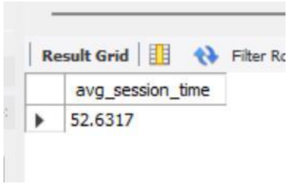


average_age_of_customer_base
26.20533526

Query 4
Analytical Purpose: Determining the average session time to get an understanding of how much time users spend per session. This will help us understand user activity and strategize what the user is exposed to in that interval.

```
### The time here is in minutes
select avg(TIMESTAMPDIFF(MINUTE, login_time, logout_time)) as avg_session_time from game.activity_history;
```

Output:



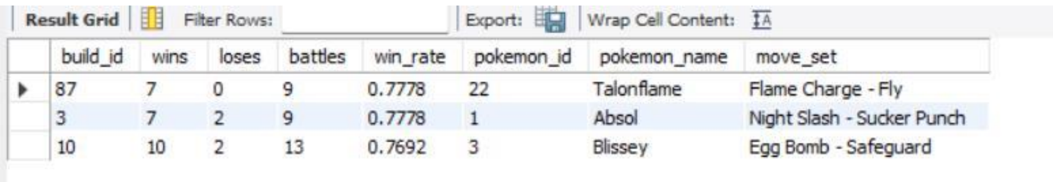
avg_session_time
52.6317

Query 5

Analytical Purpose: Determining the top 3 performing builds. By identifying this, the build can be made nerfed or modified to balance the game. Here, the build should be part of the minimum number of battles (threshold value of 6 in this sample) to judge the quality of the build accurately.

```
with base as
(select build_id, count(case when result="Win" then battle_id else null end) as wins, count(battle_id) as battles,
        count(case when result="Win" then battle_id else null end)/count(battle_id) as win_rate
 from game.battle_records
 group by build_id having battles>6)
select a.*, c.pokemon_id, d.name as pokemon_name, c.move_set
from base a
left join game.build_use c
    on a.build_id = c.build_id
left join game.pokemon d
    on c.pokemon_id = d.id
where 3 > (select count(*) from base b where a.win_rate < b.win_rate)
order by win_rate desc;
```

Output:

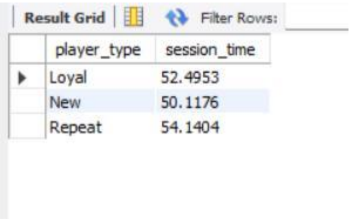


build_id	wins	loses	battles	win_rate	pokemon_id	pokemon_name	move_set
87	7	0	9	0.7778	22	Talonflame	Flame Charge - Fly
3	7	2	9	0.7778	1	Absol	Night Slash - Sucker Punch
10	10	2	13	0.7692	3	Blissey	Egg Bomb - Safeguard

Query 6
Analytical Purpose: Analysing the average time spent on the game by type of players from different ranks.

```
select case when b.rank < 10 then 'New'
                 when b.rank between 11 and 20 then 'Repeat'
                 when b.rank > 20 then 'Loyal' end as player_type,
        avg(TIMESTAMPDIFF(MINUTE, login_time, logout_time)) as session_time
from activity_history a
left join player b
              on a.player_id = b.ID
group by case when b.rank <= 10 then 'New'
                     when b.rank between 11 and 20 then 'Repeat'
                     when b.rank > 20 then 'Loyal'
              end;
```

Output:



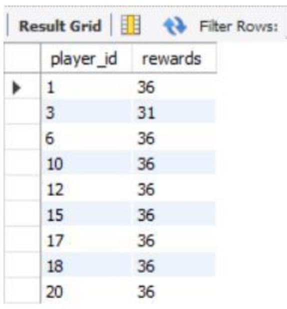
player_type	session_time
Loyal	52.4953
New	50.1176
Repeat	54.1404

Query 7

Analytical Purpose: Getting the list of players who have collected over 30 (threshold value in the sample) rewards. This can be used to flag these players as the ones who participate in events and play regularly. They can be targeted to sell items through events

```
select player_id, count(reward_id) as rewards
from game.player_event_rewards
group by player_id having rewards>=30;
```

Output:


Result Grid
Filter Rows:

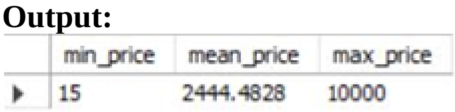
player_id	rewards
1	36
3	31
6	36
10	36
12	36
15	36
17	36
18	36
20	36

Query 8

Analytical Purpose: Getting a sense of the price range of items by calculating aggregate statistics in the item table

```
select min(price) as min_price, avg(price) as mean_price, max(price) as max_price
from game.item;
```

Output:



min_price	mean_price	max_price
15	2444.4828	10000

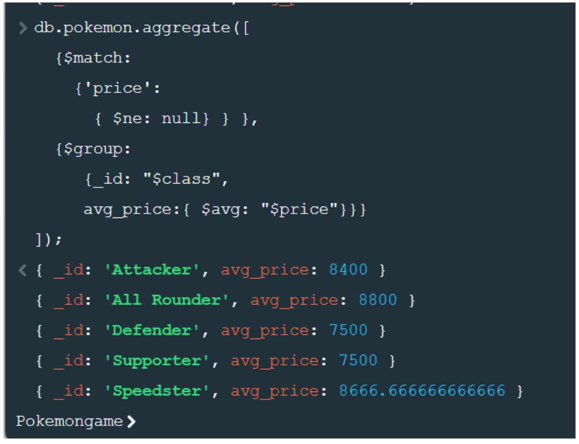
Implementation of NoSQL

Query 1

Analytical Purpose: Finding the average price of Pokémon by class

db.pokemon.aggregate([{$match: {'price':{ $ne: null} } },{$group:{_id: "$class", avg_price:{ $avg: "$price"}}}]);

Output:



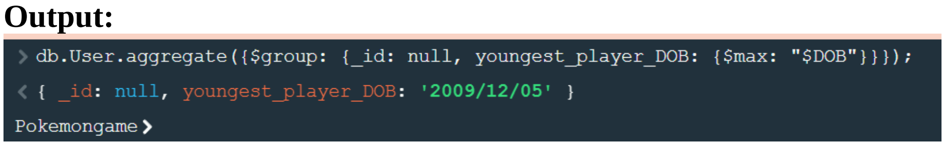
```
> db.pokemon.aggregate([
    {$match:
      {'price':
        { $ne: null} } },
    {$group:
       {_id: "$class",
       avg_price:{ $avg: "$price"}}}
  ]);
< { _id: 'Attacker', avg_price: 8400 }
  { _id: 'All Rounder', avg_price: 8800 }
  { _id: 'Defender', avg_price: 7500 }
  { _id: 'Supporter', avg_price: 7500 }
  { _id: 'Speedster', avg_price: 8666.666666666666 }
Pokemongame >
```

Query 2

Analytical Purpose: Find the DOB of the youngest player in order to gauge the limits of the age group of users and strategize accordingly

db.User.aggregate({$group: {_id: null, youngest_player_DOB: {$max: "$DOB"}}});

Output:



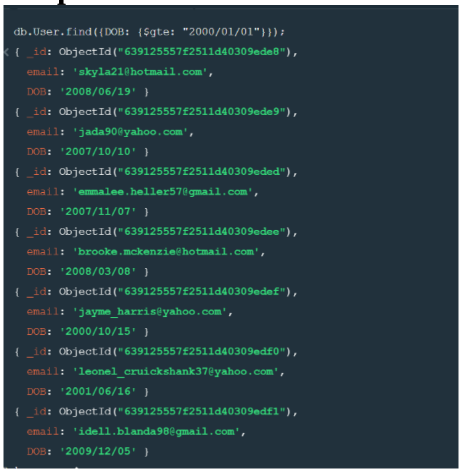
```
> db.User.aggregate({$group: {_id: null, youngest_player_DOB: {$max: "$DOB"}}});
< { _id: null, youngest_player_DOB: '2009/12/05' }
Pokemongame >
```

Query 3

Analytical Purpose: Getting the list of users who are born after the year 2000 in order to implement retention strategies targeting a younger audience

db.User.find({DOB: {$gte: "2000/01/01"}});

Output:



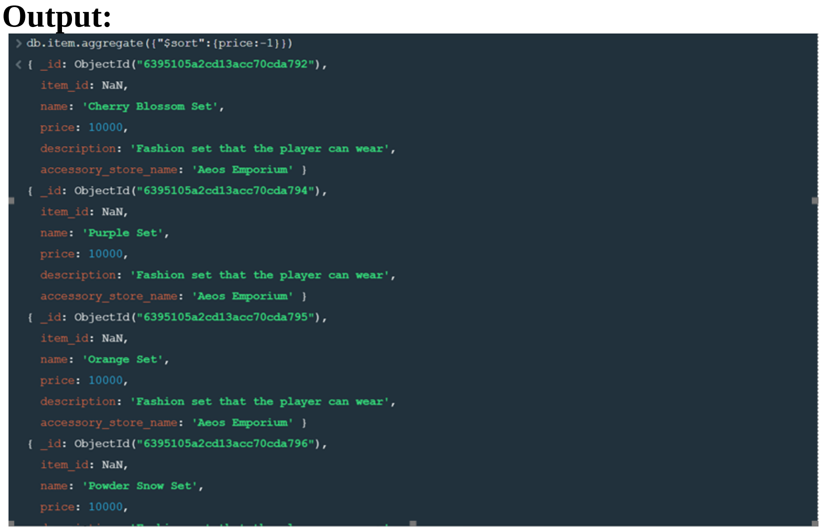
```
db.User.find({DOB: {$gte: "2000/01/01"}});
< { _id: ObjectId("639125557f2511d40309ede8"),
    email: 'skyla21@hotmail.com',
    DOB: '2008/06/19' }
  { _id: ObjectId("639125557f2511d40309ede9"),
    email: 'jada90@yahoo.com',
    DOB: '2007/10/10' }
  { _id: ObjectId("639125557f2511d40309eded"),
    email: 'emmalee.heller57@gmail.com',
    DOB: '2007/11/07' }
  { _id: ObjectId("639125557f2511d40309edee"),
    email: 'brooke.mckenzie@hotmail.com',
    DOB: '2008/03/08' }
  { _id: ObjectId("639125557f2511d40309edef"),
    email: 'jayme_harris@yahoo.com',
    DOB: '2000/10/15' }
  { _id: ObjectId("639125557f2511d40309edf0"),
    email: 'leonel_cruickshank37@yahoo.com',
    DOB: '2001/06/16' }
  { _id: ObjectId("639125557f2511d40309edf1"),
    email: 'idell.blanda98@gmail.com',
    DOB: '2009/12/05' }
```

Query 4

Analytical Purpose: Getting item information sorted by the most expensive to least expensive

db.item.aggregate({“$sort”:{price: -1}},{_id: 0, name: 1, price: 1}))

Output:



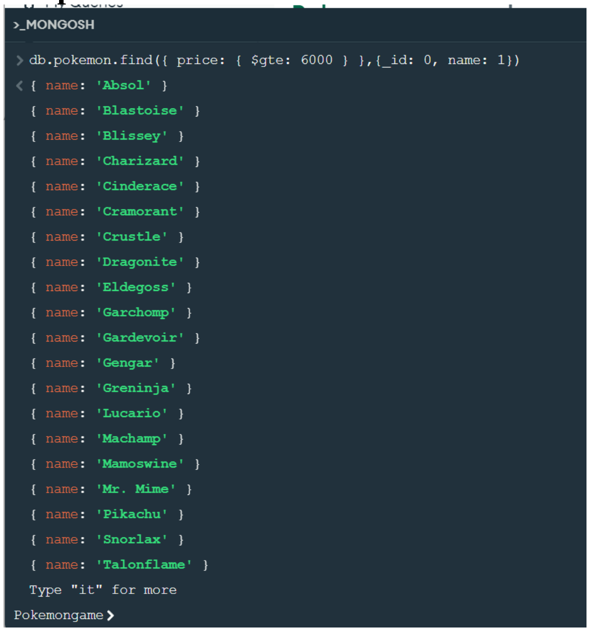
```
> db.item.aggregate({"$sort":{price:-1}})
< { _id: ObjectId("6395105a2cd13acc70cda792"),
    item_id: NaN,
    name: 'Cherry Blossom Set',
    price: 10000,
    description: 'Fashion set that the player can wear',
    accessory_store_name: 'Aeos Emporium' }
  { _id: ObjectId("6395105a2cd13acc70cda794"),
    item_id: NaN,
    name: 'Purple Set',
    price: 10000,
    description: 'Fashion set that the player can wear',
    accessory_store_name: 'Aeos Emporium' }
  { _id: ObjectId("6395105a2cd13acc70cda795"),
    item_id: NaN,
    name: 'Orange Set',
    price: 10000,
    description: 'Fashion set that the player can wear',
    accessory_store_name: 'Aeos Emporium' }
  { _id: ObjectId("6395105a2cd13acc70cda796"),
    item_id: NaN,
    name: 'Powder Snow Set',
    price: 10000,
```

Query 5

Analytical Purpose: To get the list of pokemon that cost at least 6000 coins

db.pokemon.find({ price: { $gte: 6000 } },{_id: 0, name: 1})

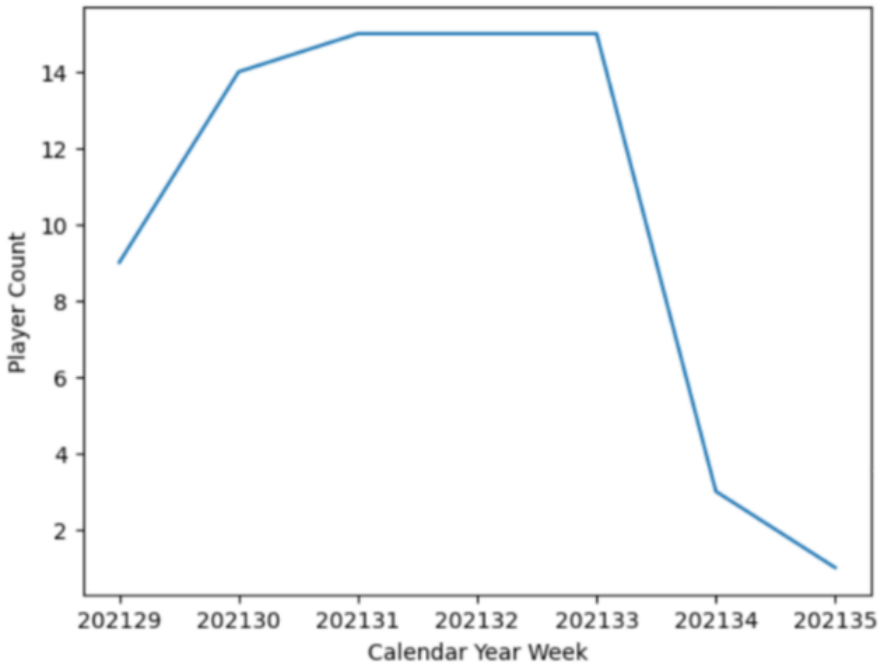
Output:



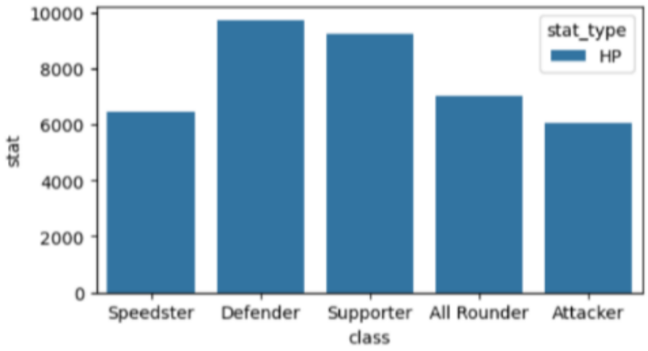
```
>_MONGOSH
> db.pokemon.find({ price: { $gte: 6000 } },{_id: 0, name: 1})
< { name: 'Absol' }
  { name: 'Blastoise' }
  { name: 'Blissey' }
  { name: 'Charizard' }
  { name: 'Cinderace' }
  { name: 'Cramorant' }
  { name: 'Crustle' }
  { name: 'Dragonite' }
  { name: 'Eldegoss' }
  { name: 'Garchomp' }
  { name: 'Gardevoir' }
  { name: 'Gengar' }
  { name: 'Greninja' }
  { name: 'Lucario' }
  { name: 'Machamp' }
  { name: 'Mamoswine' }
  { name: 'Mr. Mime' }
  { name: 'Pikachu' }
  { name: 'Snorlax' }
  { name: 'Talonflame' }
  Type "it" for more
Pokemongame >
```

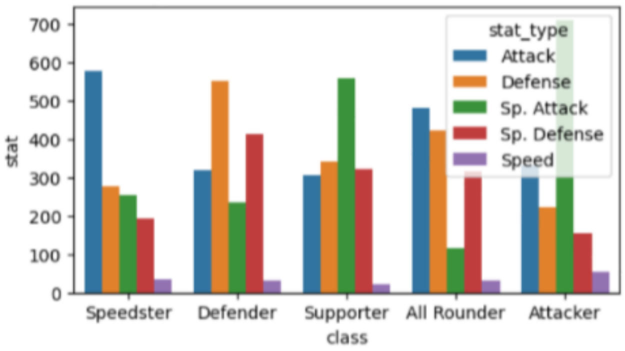
Database access via Python

A connection was set up between the MySQL server and Jupyter notebook in order to access the database via Python and generate insights. The line plot provided below tracks the overall users playing the game over the weeks. It can be noticed that the number of active users has declined over time. This would help the company keep track of the activity and evaluate how the strategies implemented impact the activity over time.



The bar graphs below are the average stats of Pokémon by class. It can be utilized by comparing the stats of a given Pokémon to its class averages and evaluating the deviation from the average. This could help balance the game, which in turn would result in a better user experience.





Summary and Recommendation

Through the study, it can be seen that user activity has been declining over the weeks due to poor user experience. It can also be seen that certain builds perform better than others and some builds need to be improved. By providing the business with this information, the game can be balanced. This will help users using the weaker builds win more often, resulting in lesser frustration and improved customer retention.

Similarly, recommendations regarding events, items, and Pokémon were made by analyzing the best-selling products and events with maximum participation. By providing the developers with these insights, the business can efficiently implement resources improvising and developing similar concepts. This maximizes the monetization opportunity. Coupled with better user experience, an increase in the ROI and profit can be expected.

Customer cohort analysis was also briefly demonstrated by categorizing customers based on age and rank. This can help the business devise different strategies based on the target audience.