Part 2:

SQL Commands:

```
--#1 Genres

CREATE TABLE Genres (
    GenreID SERIAL PRIMARY KEY,
    Genre_Name VARCHAR(255) NOT NULL
);

--#1.1 Insert Data:
\copy Genres FROM 'C:\Users\micha\Desktop\DB_2\Genre.csv' WITH (FORMAT csv,
HEADER true);

--#1.2 SELECT Data:
SELECT *
FROM Genres
LIMIT 5;

--#2 Publishers
CREATE TABLE Publishers (
    PublisherID SERIAL PRIMARY KEY,
    PubName VARCHAR(255) NOT NULL
);
--#2.1 Insert Data:
\copy Publishers FROM 'C:\Users\micha\Desktop\DB_2\Publisher.csv' WITH (FORMAT
csv, HEADER true);
--#2.2 SELECT Data:
SELECT *
FROM Publishers
LIMIT 5;


--#3 Developers
CREATE TABLE Developers (
    DeveloperID SERIAL PRIMARY KEY,
    DevName VARCHAR(255) NOT NULL
);
--#3.1 Insert Data:
\copy Developers FROM 'C:\Users\micha\Desktop\DB_2\Developer.csv' WITH (FORMAT
csv, HEADER true);
--#3.2 SELECT Data:
SELECT *
FROM Developers
LIMIT 5;
```

```sql
--#4 Regions
CREATE TABLE Regions (
    RegionID SERIAL PRIMARY KEY,
    Name VARCHAR(255) NOT NULL
);
--#4.1 Insert Data:
\copy Regions FROM 'C:\Users\micha\Desktop\DB_2\Region.csv' WITH (FORMAT csv,
HEADER true);
--#4.2 SELECT Data:
SELECT *
FROM Regions
LIMIT 5;


--#5 Users
CREATE TABLE Users (
    UserID SERIAL PRIMARY KEY,
    Name VARCHAR(255) NOT NULL,
    RegionID INT REFERENCES Regions(RegionID)
);

--#5.1 Insert Data:
\copy Users FROM 'C:\Users\micha\Desktop\DB_2\Users.csv' WITH (FORMAT csv, HEADER
true);
--#5.2 SELECT Data:
SELECT *
FROM Users
LIMIT 5;


--#6 Console_Generations
CREATE TABLE Console_Generations (
    GenerationID SERIAL PRIMARY KEY,
    Start_Year INT NOT NULL,
    End_Year INT
);
--#6.1 Insert Data:
\copy Console_Generations FROM
'C:\Users\micha\Desktop\DB_2\Console_Generation.csv' WITH (FORMAT csv, HEADER
true);
--#6.2 SELECT Data:
SELECT *
FROM Console_Generations
LIMIT 5;
```

```sql
--#7 Rating_Orgs
CREATE TABLE Rating_Orgs (
    RatingOrgID VARCHAR(255) PRIMARY KEY,
    Title VARCHAR(255) NOT NULL,
    Year_Established INT NOT NULL,
    RegionID INT REFERENCES Regions(RegionID)
);
--#7.1 Insert Data:
\copy Rating_Orgs FROM 'C:\Users\micha\Desktop\DB_2\Rating_Orgs.csv' WITH (FORMAT
csv, HEADER true);
--#7.2 SELECT Data:
SELECT *
FROM Rating_Orgs
LIMIT 5;


--#8 Ratings
CREATE TABLE Ratings (
    RatingOrgID VARCHAR(255) REFERENCES Rating_Orgs(RatingOrgID),
    RatingID VARCHAR(255) PRIMARY KEY,
    Title VARCHAR(255) NOT NULL,
    Descriptions TEXT
);
--#8.1 Insert Data:
\copy Ratings FROM 'C:\Users\micha\Desktop\DB_2\Rating.csv' WITH (FORMAT csv,
HEADER true);
--#8.2 SELECT Data:
SELECT *
FROM Ratings
LIMIT 5;


--#9 Platforms
CREATE TABLE Platforms (
    PlatformID SERIAL PRIMARY KEY,
    PlatformName VARCHAR(255) NOT NULL,
    GenerationID INT REFERENCES Console_Generations(GenerationID),
    Full_Title VARCHAR(255) NOT NULL,
    Year_Launched INT NOT NULL,
    PublisherID INT REFERENCES Publishers(PublisherID)
);
--#9.1 Insert Data:
\copy Platforms FROM 'C:\Users\micha\Desktop\DB_2\Platform.csv' WITH (FORMAT csv,
HEADER true);
--#9.2 SELECT Data:
```

```sql
SELECT *
FROM Platforms
LIMIT 5;


--#10 Games
CREATE TABLE Games (
    GameID SERIAL PRIMARY KEY,
    Name VARCHAR(255) NOT NULL,
    Year INT NOT NULL,
    GenreID INT REFERENCES Genres(GenreID),
    PublisherID INT REFERENCES Publishers(PublisherID),
    RatingID VARCHAR(255) REFERENCES Ratings(RatingID)
);
--#10.1 Insert Data:
\copy Games FROM 'C:\Users\micha\Desktop\DB_2\Game.csv' WITH (FORMAT csv, HEADER
true);
--#10.2 SELECT Data:
SELECT *
FROM Games
LIMIT 5;


--#11 Game_Developers
CREATE TABLE Game_Developers (
    GameID INT REFERENCES Games(GameID),
    DeveloperID INT REFERENCES Developers(DeveloperID),
    PRIMARY KEY (GameID, DeveloperID)
);
--#11.1 Insert Data:
\copy Game_Developers FROM 'C:\Users\micha\Desktop\DB_2\Game_Developers.csv' WITH
(FORMAT csv, HEADER true);
--#11.2 SELECT Data:
SELECT *
FROM Game_Developers
LIMIT 5;


--#12 Reviews
CREATE TABLE Reviews (
    ReviewID SERIAL PRIMARY KEY,
    UserID INT REFERENCES Users(UserID),
    GameID INT REFERENCES Games(GameID),
    Score FLOAT CHECK (Score >= 0 AND Score <=5) NOT NULL,
    Comment TEXT,
```

```
    Recommend BOOLEAN
);
--#12.1 Insert Data:
\copy Reviews FROM 'C:\Users\micha\Desktop\DB_2\Review.csv' WITH (FORMAT csv,
HEADER true);
--#12.2 SELECT Data:
SELECT *
FROM Reviews
LIMIT 5;


--#13 Reviews
CREATE TABLE Public_Reception (
    GameID INT PRIMARY KEY REFERENCES Games(GameID),
    Critic_Score FLOAT,
    Critic_Count INT,
    Consumer_Score FLOAT,
    Consumer_Count INT
);
--#13.1 Insert Data:
\copy Reviews FROM 'C:\Users\micha\Desktop\DB_2\Review.csv' WITH (FORMAT csv,
HEADER true);
--#13.2 SELECT Data:
SELECT *
FROM Public_Reception
LIMIT 5;


--#14 Sales
CREATE TABLE Sales (
    GameID INT REFERENCES Games(GameID),
    RegionID INT REFERENCES Regions(RegionID),
    Sales FLOAT NOT NULL,
    PRIMARY KEY (GameID, RegionID)
);
--#14.1 Insert Data:
\copy Sales FROM 'C:\Users\micha\Desktop\DB_2\Sales.csv' WITH (FORMAT csv, HEADER
true);
--#14.2 SELECT Data:
SELECT *
FROM Sales
LIMIT 5;
```

Screenshots:

```
postgres=# CREATE TABLE Genres (
postgres(#     GenreID SERIAL PRIMARY KEY,
postgres(#     Genre_Name VARCHAR(255) NOT NULL
postgres(# );
CREATE TABLE
postgres=# \copy Genres FROM 'C:\Users\micha\Desktop\DB_2\Genre.csv' WITH (FORMAT csv, HEADER true);
COPY 12
postgres=# SELECT *
postgres-# FROM Genres
postgres-# LIMIT 5;
 genreid |  genre_name
---------+--------------
       1 | Role-Playing
       2 | Platform
       3 | Adventure
       4 | Action
       5 | Racing
(5 rows)
```

```
postgres=# CREATE TABLE Publishers (
postgres(#     PublisherID SERIAL PRIMARY KEY,
postgres(#     PubName VARCHAR(255) NOT NULL
postgres(# );
CREATE TABLE
postgres=# \copy Publishers FROM 'C:\Users\micha\Desktop\DB_2\Publisher.csv' WITH (FORMAT csv, HEADER true);
COPY 585
postgres=# SELECT *
postgres-# FROM Publishers
postgres-# LIMIT 5;
 publisherid |          pubname
-------------+---------------------------
           1 | FuRyu
           2 | Nintendo
           3 | Disney Interactive Studios
           4 | Namco Bandai Games
           5 | Atari
(5 rows)
```

```
postgres=# CREATE TABLE Developers (
postgres(#     DeveloperID SERIAL PRIMARY KEY,
postgres(#     DevName VARCHAR(255) NOT NULL
postgres(# );
CREATE TABLE
postgres=# \copy Developers FROM 'C:\Users\micha\Desktop\DB_2\Developer.csv' WITH (FORMAT csv, HEADER true);
COPY 2138
postgres=# SELECT *
postgres-# FROM Developers
postgres-# LIMIT 5;
 developerid |          devname
-------------+---------------------------
           1 | FuRyu
           2 | Nintendo
           3 | Disney Interactive Studios
           4 | Namco Bandai Games
           5 | Bandai Namco Games
(5 rows)
```

```
postgres=# CREATE TABLE Regions (
postgres(#     RegionID SERIAL PRIMARY KEY,
postgres(#     Name VARCHAR(255) NOT NULL
postgres(# );
CREATE TABLE
postgres=# \copy Regions FROM 'C:\Users\micha\Desktop\DB_2\Region.csv' WITH (FORMAT csv, HEADER true);
COPY 5
postgres=# SELECT *
postgres-# FROM Regions
postgres-# LIMIT 5;
 regionid |  name
----------+--------
        1 | US
        2 | EU
        3 | JP
        4 | Other
        5 | Global
(5 rows)
```

```
postgres=# CREATE TABLE Users (
postgres(#     UserID SERIAL PRIMARY KEY,
postgres(#     Name VARCHAR(255) NOT NULL,
postgres(#     RegionID INT REFERENCES Regions(RegionID)
postgres(# );
CREATE TABLE
postgres=# \copy Users FROM 'C:\Users\micha\Desktop\DB_2\Users.csv' WITH (FORMAT csv, HEADER true);
COPY 1
postgres=# SELECT *
postgres-# FROM Users
postgres-# LIMIT 5;
 userid |  name   | regionid
--------+---------+----------
      1 | Michael |        1
(1 row)
```

```
postgres=# CREATE TABLE Console_Generations (
postgres(#     GenerationID SERIAL PRIMARY KEY,
postgres(#     Start_Year INT NOT NULL,
postgres(#     End_Year INT
postgres(# );
CREATE TABLE
postgres=# \copy Console_Generations FROM 'C:\Users\micha\Desktop\DB_2\Console_Generation.csv' WITH (FORMAT csv, HE
ADER true);
COPY 9
postgres=# SELECT *
postgres-# FROM Console_Generations
postgres-# LIMIT 5;
 generationid | start_year | end_year
--------------+------------+----------
            1 |       1972 |     1980
            2 |       1976 |     1992
            3 |       1983 |     2003
            4 |       1987 |     2004
            5 |       1993 |     2006
(5 rows)
```

```
postgres=# CREATE TABLE Rating_Orgs (
postgres(#     RatingOrgID VARCHAR(255) PRIMARY KEY,
postgres(#     Title VARCHAR(255) NOT NULL,
postgres(#     Year_Established INT NOT NULL,
postgres(#     RegionID INT REFERENCES Regions(RegionID)
postgres(# );
CREATE TABLE
postgres=# \copy Rating_Orgs FROM 'C:\Users\micha\Desktop\DB_2\Rating_Orgs.csv' WITH (FORMAT csv, HEADER true);
COPY 3
postgres=# SELECT *
postgres-# FROM Rating_Orgs
postgres-# LIMIT 5;
 ratingorgid |                              title                              | year_established | regionid
-------------+-----------------------------------------------------------------+------------------+----------
 ESRB        | Entertainment Software Rating Board                             |             1994 |        1
 PEGI        | Pan-European Game Information                                   |             2003 |        2
 CERO        | Computer Entertainment Rating Organization Video Game Rating System |             2002 |        3
(3 rows)
```

```
postgres=# CREATE TABLE Ratings (
postgres(#     RatingOrgID VARCHAR(255) REFERENCES Rating_Orgs(RatingOrgID),
postgres(#     RatingID VARCHAR(255) PRIMARY KEY,
postgres(#     Title VARCHAR(255) NOT NULL,
postgres(#     Descriptions TEXT
postgres(# );
CREATE TABLE
postgres=# \copy Ratings FROM 'C:\Users\micha\Desktop\DB_2\Rating.csv' WITH (FORMAT csv, HEADER true);
COPY 21
postgres=# SELECT *
postgres-# FROM Ratings
postgres-# LIMIT 5;
 ratingorgid | ratingid |         title         |                                                                      descriptions
-------------+----------+-----------------------+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 ESRB        | EC       | Early Childhood       | Titles rated EC (Early Childhood) have content that may be suitable for ages 3 and older. Contains no material that parents would find inappropriate
 ESRB        | E        | Everyone              | Titles rated E (Everyone) have content that is generally suitable for all ages. May contain minimal cartoon, fantasy or mild violence and/or infrequent use of mild language.
 ESRB        | E10+     | Everyone 10 and older | Titles rated E10+ (Everyone 10 and older) have content that is generally suitable for ages 10 and up. May contain more cartoon, fantasy or mild violence, mild language and/or minimal suggestive themes.themes.
 ESRB        | T        | Teen                  | Titles rated T (Teen) have content that is generally suitable for ages 13 and up. May contain violence, suggestive themes, crude humor, minimal blood, simulated gambling, and/or infrequent use of strong language.
 ESRB        | M        | Mature                | Titles rated M (Mature) have content that is generally suitable for persons ages 17 and up. May contain intense violence, blood and gore, sexual content and/or strong language.
(5 rows)
```

```
postgres=# CREATE TABLE Platforms (
postgres(#     PlatformID SERIAL PRIMARY KEY,
postgres(#     PlatformName VARCHAR(255) NOT NULL,
postgres(#     GenerationID INT REFERENCES Console_Generations(GenerationID),
postgres(#     Full_Title VARCHAR(255) NOT NULL,
postgres(#     Year_Launched INT NOT NULL,
postgres(#     PublisherID INT REFERENCES Publishers(PublisherID)
postgres(# );
CREATE TABLE
postgres=# \copy Platforms FROM 'C:\Users\micha\Desktop\DB_2\Platform.csv' WITH (FORMAT csv, HEADER true);
COPY 34
postgres=# SELECT *
postgres-# FROM Platforms
postgres-# LIMIT 5;
 platformid | platformname | generationid |      full_title      | year_launched | publisherid
------------+--------------+--------------+----------------------+---------------+-------------
          1 | 3DS          |            8 | Nintendo 3DS         |          2011 |           2
          2 | DS           |            7 | Nintendo DS          |          2004 |           2
          3 | PS3          |            7 | PlayStation 3        |          2006 |          23
          4 | PS2          |            6 | PlayStation 2        |          2000 |          23
          5 | PSP          |            7 | PlayStation Portable |          2005 |          23
(5 rows)


postgres=# |
```

```
postgres=# CREATE TABLE Games (
postgres(#     GameID SERIAL PRIMARY KEY,
postgres(#     Name VARCHAR(255) NOT NULL,
postgres(#     Year INT NOT NULL,
postgres(#     GenreID INT REFERENCES Genres(GenreID),
postgres(#     PublisherID INT REFERENCES Publishers(PublisherID),
postgres(#     RatingID VARCHAR(255) REFERENCES Ratings(RatingID)
postgres(# );
CREATE TABLE
postgres=# \copy Games FROM 'C:\Users\micha\Desktop\DB_2\Game.csv' WITH (FORMAT csv, HEADER true);
ERROR:  insert or update on table "games" violates foreign key constraint "games_ratingid_fkey"
DETAIL:  Key (ratingid)=(E10) is not present in table "ratings".
postgres=# \copy Games FROM 'C:\Users\micha\Desktop\DB_2\Game.csv' WITH (FORMAT csv, HEADER true);
COPY 16716
postgres=# SELECT *
postgres-# FROM Games
postgres-# LIMIT 5;
 gameid |            name             | year | genreid | publisherid | ratingid
--------+-----------------------------+------+---------+-------------+----------
      1 |  Beyblade Burst             | 2016 |       1 |           1 | A
      2 |  Fire Emblem Fates          | 2015 |       1 |           2 |
      3 |  Frozen: Olaf's Quest       | 2013 |       2 |           3 | E
      4 |  Frozen: Olaf's Quest       | 2013 |       2 |           3 | E
      5 |  Haikyu!! Cross Team Match! | 2016 |       3 |           4 |
(5 rows)
```

```
postgres=# CREATE TABLE Game_Developers (
postgres(#     GameID INT REFERENCES Games(GameID),
postgres(#     DeveloperID INT REFERENCES Developers(DeveloperID),
postgres(#     PRIMARY KEY (GameID, DeveloperID)
postgres(# );
CREATE TABLE
```

```
postgres=# \copy Game_Developers FROM 'C:\Users\micha\Desktop\DB_2\Game_Developers.csv' WITH (FORMAT csv, HEADER true);
COPY 17269
postgres=# SELECT *
postgres-# FROM Game_Developers
postgres-# LIMIT 5;
 gameid | developerid
--------+-------------
      1 |           1
      2 |           2
      3 |           3
      4 |           3
      5 |           4
(5 rows)
```

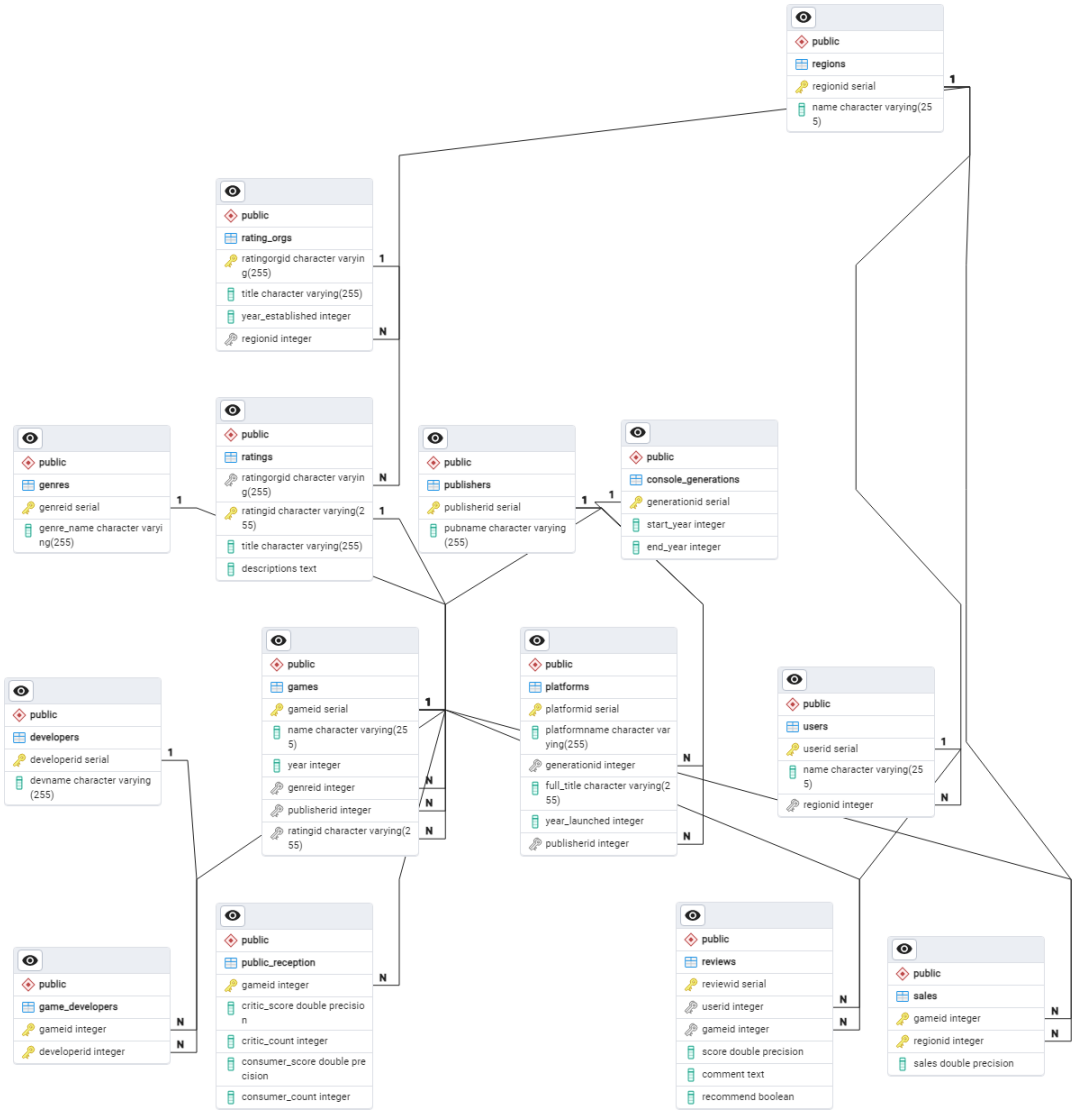
```
postgres=# CREATE TABLE Reviews (
postgres(#     ReviewID SERIAL PRIMARY KEY,
postgres(#     UserID INT REFERENCES Users(UserID),
postgres(#     GameID INT REFERENCES Games(GameID),
postgres(#     Score FLOAT CHECK (Score ≥ 0 AND Score ≤5) NOT NULL,
postgres(#     Comment TEXT,
postgres(#     Recommend BOOLEAN
postgres(# );
CREATE TABLE
postgres=# \copy Reviews FROM 'C:\Users\micha\Desktop\DB_2\Review.csv' WITH (FORMAT csv, HEADER true);
COPY 1
postgres=# SELECT *
postgres-# FROM Reviews
postgres-# LIMIT 5;
 reviewid | userid | gameid | score |                 comment                  | recommend
----------+--------+--------+-------+------------------------------------------+-----------
        1 |      1 |   1506 |     5 | "This is one of my favorite games ever!" | t
(1 row)
```

```
postgres=# CREATE TABLE Public_Reception (
postgres(#     GameID INT PRIMARY KEY REFERENCES Games(GameID),
postgres(#     Critic_Score FLOAT,
postgres(#     Critic_Count INT,
postgres(#     Consumer_Score FLOAT,
postgres(#     Consumer_Count INT
postgres(# );
CREATE TABLE
postgres=# \copy Public_Reception FROM 'C:\Users\micha\Desktop\DB_2\Public_Reception.csv' WITH (FORMAT csv, HEADER true);
COPY 16716
postgres=# SELECT *
postgres-# FROM Public_Reception
postgres-# LIMIT 5;
 gameid | critic_score | critic_count | consumer_score | consumer_count
--------+--------------+--------------+----------------+----------------
      1 |              |              |                |
      2 |              |              |                |
      3 |              |              |             28 |              1
      4 |              |              |             28 |              1
      5 |              |              |                |
(5 rows)
```

```
postgres=# CREATE TABLE Sales (
postgres(#     GameID INT REFERENCES Games(GameID),
postgres(#     RegionID INT REFERENCES Regions(RegionID),
postgres(#     Sales FLOAT NOT NULL,
postgres(#     PRIMARY KEY (GameID, RegionID)
postgres(# );
CREATE TABLE
postgres=# \copy Sales FROM 'C:\Users\micha\Desktop\DB_2\Sales.csv' WITH (FORMAT csv, HEADER true);
COPY 83580
postgres=# SELECT *
postgres-# FROM Sales
postgres-# LIMIT 5;
 gameid | regionid | sales
--------+----------+-------
      1 |        1 |     0
      2 |        1 |  0.81
      3 |        1 |  0.21
      4 |        1 |  0.27
      5 |        1 |     0
(5 rows)
```

Part 2:

The normalized tables and ERD diagram



The original data set only had the following columns:

It turned out that a majority of the entries for ratings, developer, year_of_release, critc_score/critic_count, user_score/user_count were actually left null and the data set was poor. I had used the website https://www.igdb.com/ and their API to fill out the missing information on those columns I mentioned earlier. It had taken over a day to process. Below is the code that I used to pull extra data from the api: **(I discuss the normalization after the code)**

```
# coding: utf8
import requests
import time
import csv

# Define enums for rating systems
rating_systems_enum = {
    1: "ESRB",
    2: "PEGI",
    3: "CERO",
    4: "USK",
    5: "GRAC",
    6: "CLASS_IND",
    7: "ACB"
}

# Define enums for ratings
ratings_enum = {
    1: "Three",
    2: "Seven",
    3: "Twelve",
    4: "Sixteen",
    5: "Eighteen",
    6: "RP",
    7: "EC",
    8: "E",
    9: "E10",
    10: "T",
    11: "M",
    12: "AO",
    13: "CERO_A",
    14: "CERO_B",
    15: "CERO_C",
    16: "CERO_D",
```

```
    17: "CERO_Z",
    18: "USK_0",
    19: "USK_6",
    20: "USK_12",
    21: "USK_16",
    22: "USK_18",
    23: "GRAC_ALL",
    24: "GRAC_Twelve",
    25: "GRAC_Fifteen",
    26: "GRAC_Eighteen",
    27: "GRAC_TESTING",
    28: "CLASS_IND_L",
    29: "CLASS_IND_Ten",
    30: "CLASS_IND_Twelve",
    31: "CLASS_IND_Fourteen",
    32: "CLASS_IND_Sixteen",
    33: "CLASS_IND_Eighteen",
    34: "ACB_G",
    35: "ACB_PG",
    36: "ACB_M",
    37: "ACB_MA15",
    38: "ACB_R18",
    39: "ACB_RC"
}


def get_game_id(game_name, client_id, access_token):
    url = "https://api.igdb.com/v4/games"
    headers = {
        "Client-ID": client_id,
        "Authorization": access_token,
    }
    query = f"fields=id&search={game_name}"
    response = requests.get(url, headers=headers, params=query)
    if response.status_code == 200:
        data = response.json()
        if data:
            # Assuming the first result is the most relevant
            return data[0]["id"]
    return None

def get_game_data(game_name ,client_id, access_token):
    game_id = get_game_id(game_name, client_id, access_token)
    url = "https://api.igdb.com/v4/games"
    if game_id:
```

```
        url = f"https://api.igdb.com/v4/games/{game_id}"
        headers = {
            "Client-ID": client_id,
            "Authorization": access_token,
        }
        query =
"fields=age_ratings,aggregated_rating,aggregated_rating_count,involved_companies,
name,rating,rating_count,summary"
        #query =
"fields=age_ratings.rating,aggregated_rating,aggregated_rating_count,total_rating
,total_rating_count,summary,involved_companies"
        response = requests.get(url, headers=headers, params=query)
        if response.status_code == 200:
            data = response.json()
            if data:
                time.sleep(1)
                #age rating
                ratings = []
                selected_rating = None
                if "age_ratings" in data[0]:
                    age_ratings = data[0].get("age_ratings")
                    for rating in age_ratings:
                        x=get_ratings(rating, client_id, access_token)
                        if x is not None:
                            if x[0] == "ESRB":
                                selected_rating = x[1]
                                break
                            else:
                                ratings.append(x)

                if selected_rating is None:
                    # Loop through each rating tuple
                    for rating_system, rating_value in ratings:
                        if rating_system == 'ESRB':
                            # If ESRB rating exists, select it and break the loop
                            selected_rating = rating_value
                            break
                        elif rating_system == 'PEGI' and selected_rating is None:
                            # If no ESRB rating but PEGI rating exists, select it
                            selected_rating = rating_value
                        elif rating_system == 'CERO' and selected_rating is None:
                        # If no ESRB or PEGI rating, select CERO rating
                            selected_rating = rating_value

                        # Print the selected rating
```

```
                        print("Selected Rating:", selected_rating)
                            #to align with our data we shall only include ESRB or
PEGI or
                        #print(f"{ratings}")

                #companies
                developer=[]
                if "involved_companies" in data[0]:
                    involved_companies = data[0].get("involved_companies")
                    for company in involved_companies:
                        result = get_involved_name(company, client_id,
access_token)
                        if result is not None:
                            if result[1] == None:
                                #formats out empty descriptions, eh we dont
really need them anyways
                                developer.append(result[0])
                if len(developer) == 1:
                    developer = developer[0]
                            #developer.append(result)
                    #print(f"{developer}")
                Critic_Score = data[0].get("aggregated_rating")
                Critic_Count = data[0].get("aggregated_rating_count")
                name = data[0].get("name")
                consumer_score = data[0].get("rating")
                consumer_rating = data[0].get("rating_count")
                summary = data[0].get("summary")
                #other data
                #name, Critic_Score Critic_Count, User_Score,   User_Count,
Developer,  Rating,

                return (name, Critic_Score, Critic_Count, consumer_score,
consumer_rating, developer, selected_rating, summary)
            else:
                return None
        elif response.status_code == 429:
            print(f"Sleeping on '{game_name}'")
            return get_game_data(game_name, client_id, access_token)  # Retry the
request
        else:
            print(f"ERROR '{game_name}': {response.text}")
            return None
    else:
        return None
```

```
def get_ratings(rating_id, client_id, access_token):
    url = f"https://api.igdb.com/v4/age_ratings/{rating_id}"
    headers = {
        "Client-ID": client_id,
        "Authorization": access_token,
    }
    query = "fields=category,rating"
    response = requests.get(url, headers=headers, params=query)
    if response.status_code == 200:
        data = response.json()
        if data:
            rating_number = data[0].get("rating")
            category = data[0].get("category")
            organization = rating_systems_enum.get(category)
            rating = ratings_enum.get(rating_number)
            if organization=="ESRB" or organization=="PEGI" or
organization=="CERO":
                return (organization, rating)
        return None
    elif response.status_code == 429:
        print("Rate limited, retrying...")
        time.sleep(1)
        return get_ratings(rating_id, client_id, access_token)  # Retry the
request
    else:
        print(f"ERROR: Failed to fetch for rating: {rating_id}")
        return None




def get_involved_name(company, client_id, access_token):
    url = f"https://api.igdb.com/v4/involved_companies/{company}"
    headers = {
        "Client-ID": client_id,
        "Authorization": access_token,
    }
    query = "fields=company,developer,porting,publisher,supporting"
    response = requests.get(url, headers=headers, params=query)
    if response.status_code == 200:
        data = response.json()
        if data:
            company_id = data[0].get("company")
            developer = data[0].get("developer")
            if developer:
```

```
                return get_developer_name(company_id, client_id, access_token)
        return None #no dev found, we already have publishers
    elif response.status_code == 429:
        print("Rate limited, retrying...")
        time.sleep(1)
        return get_involved_name(company, client_id, access_token)  # Retry the
request
    else:
        print(f"ERROR: Failed to fetch company name for ID {company}")
        return None

def get_developer_name(company_id, client_id, access_token):
    url = f"https://api.igdb.com/v4/companies/{company_id}"
    headers = {
        "Client-ID": client_id,
        "Authorization": access_token,
    }
    query = "fields=description,name"
    response = requests.get(url, headers=headers, params=query)
    if response.status_code == 200:
        data = response.json()
        if data:
            description = data[0].get("description")
            name = data[0].get("name")
            return (name,description)
    elif response.status_code == 429:
        print("Rate limited, retrying...")
        time.sleep(1)
        return get_developer_name(company_id, client_id, access_token)  # Retry
the request
    else:
        print(f"ERROR: Failed to fetch dev name for ID {company_id}")
        return None

def query(game_list,client_id, access_token):
    all_data = []
    for game_name in game_list:
        print(f"queuing'{game_name}'")
        try:
            game_data = get_game_data(game_name, client_id, access_token)
            if game_data:
                game_data = ['' if value is None else value for value in
game_data]
                all_data.append(game_data)
                print(f"Found info for'{game_name}' is:")
```

```
                time.sleep(0.25)  # Introduce a delay of 0.25 seconds between
each request to respect the rate limit
        except Exception as e:
            print(f"Error occurred for '{game_name}': {e}")
            all_data.append((game_name,))
    # Write data to CSV file
    if all_data:
        with open('game_data.csv', 'w', newline='', encoding='utf-8') as csvfile:
            writer = csv.writer(csvfile)
            writer.writerow(['Name', 'Critic Score', 'Critic Count', 'Consumer
Score', 'Consumer Rating', 'Developer', 'Rating', 'Summary'])
            writer.writerows(all_data)
        print("Data written to game_data.csv")
    else:
        print("No data to write to CSV")

    return all_data


# Example usage
game_list = ["Hangman", "Home Run", "Housekeeping", "Phantasy Star Online 2
Episode 4: Deluxe Package", "Brothers Conflict: Precious Baby", "Phantasy Star
Online 2 Episode 4: Deluxe Package", "ZombiU", "The Elder Scrolls V: Skyrim",
"Resident Evil Zero",....] #over 16,000 entries

client_id = "[client id]"
access_token = "Bearer [token]"
info = query(my_list, client_id, access_token)
```

Normalization

To create normalize the data I split the data into several csv files so that I could directly upload them to in postgresql. I used multiple python scripts to split the data in the original kaggle dataset with the new information.

BCNF PROOF:

1. Genres Table
    a. GenreID -> GenreName
    b. GenreID is the primary key and the only functionally dependency from the primary key to other attributes. This makes GenereID the super key
2. Publisher Table:
    a. PublisherID -> Publisher Name
    b. PublisherID is the primary key and the only functionally dependency from the primary key to other attributes. This makes PublisherID the super key
3. Developer Table:
    a. DeveloperID-> Developer Name
    b. DeveloperID is the primary key and the only functionally dependency from the primary key to other attributes. This makes DeveloperID the super key
4. Region Table:
    a. RegionID -> Region Name
    b. RegionID is the primary key and the only functionally dependency from the primary key to other attributes. This makes RegionID the super key
5. User Table:
    a. UserID -> Name, RegionID
    b. UserID is the primary key, and all dependies originate from UserID.
6. Console Generations Table
    a. GenerationID -> Start_Year, End_Year
    b. GenerationID is the primary key, and all dependencies originate from GenerationID.
7. Rating Organizations Table
    a. RatingOrgID -> Title, Year_Established, RegionID
    b. RatingOrgID is the primary key, and all attirbutes depend on it.
8. Rating Table
    a. All attributes dpend on RatingID and since it RatingID is a superkey. RatingOrgID only links each rating to the organization that it belongs to.
9. Platform Table
    a. All attributes depend on the primary key PlatformID
10. Games Table
    a. Each game is uniquely identified by GameID. This table consists of many foreign keys, but every single game is identifiable.
11. Game Developer Table
    a. Contains a composite primary key (GameID and DeveloperID).Allows the many to many relationships to be satisfied and maintains BCNF.
12. Reviews Table
    a. ReviewID is the primary key and all other columns depend on to to relate to the foreign keys, maintains BCNF.

13. Public Reception Table
    a. All columns depend on GameID which is its foreign key and primary key, maintains BCNF.
14. Sales Table
    a. Uses a composte primary key of (GameID and RegionID), this ensures all functional dependencies are from a super key, maintains BCNF

All tables already in BCNF, and maintains BCNF.

Part 3: All the queries are outlined in my CLI code

View Data Menu:

```
View Data Menu
Genres:
- SELECT genreid, genre_name FROM public.genres ORDER BY genreid;
Publishers:
- SELECT publisherid, pubname FROM public.publishers ORDER BY publisherid;
Developers:
- SELECT developerid, devname FROM public.developers ORDER BY developerid;
Platforms:
- SELECT platformid, platformname, generationid FROM public.platforms ORDER BY
platformid;
Console Generations:
- SELECT generationid, start_year, end_year FROM public.Console_Generations ORDER
BY generationid;
Games:
- SELECT g.gameid, g.name, g.year, gr.genre_name, p.pubname FROM public.games g
LEFT JOIN public.genres gr ON g.genreid = gr.genreid LEFT JOIN public.publishers
p ON g.publisherid = p.publisherid ORDER BY g.name;
Reviews:
- SELECT * FROM public.reviews; (With optional filtering by game ID)
Regions:
- SELECT regionid, name FROM public.regions;
User Management Menu
Create User:
- INSERT INTO public.users (name, regionid) VALUES (%s, %s);
- UPDATE User Info:
- UPDATE public.users SET name = COALESCE(%s, name), regionid = COALESCE(%s,
regionid) WHERE userid = %s;
- DELETE User:
- DELETE FROM public.users WHERE userid = %s;
Game Management Menu
Add New Game:
```

```
- INSERT INTO public.games (name, year, genreid, publisherid, ratingid) VALUES
(%s, %s, %s, %s, %s);
- UPDATE Game Info:
- UPDATE public.games SET name = COALESCE(%s, name), year = COALESCE(%s, year),
genreid = COALESCE(%s, genreid), publisherid = COALESCE(%s, publisherid),
ratingid = COALESCE(%s, ratingid) WHERE gameid = %s;
- DELETE Game:
- DELETE FROM public.games WHERE gameid = %s;
Review Management Menu
Write Review:
- INSERT INTO public.reviews (userid, gameid, score, comment, recommend) VALUES
(%s, %s, %s, %s, %s);
Edit Review:
- UPDATE public.reviews SET score = COALESCE(%s, score), comment = COALESCE(%s,
comment), recommend = COALESCE(%s, recommend) WHERE reviewid = %s;
- DELETE Review:
- DELETE FROM public.reviews WHERE reviewid = %s;
Sales Data Menu
View Sales by Game:
- SELECT regionid, sales FROM public.sales WHERE gameid = %s;
View Sales by Region:
- SELECT gameid, sales FROM public.sales WHERE regionid = %s;
Best Selling Games by Criteria (Region, Genre, Rating):
Queries depend on the criteria - SELECTed but generally follow the structure of
joining the sales and games tables and ordering by sales.
Rating Information Menu
View Ratings:
- SELECT ratingid, title, descriptions FROM public.ratings; (With optional
filtering by rating organization ID)
View Rating Organizations:
- SELECT ratingorgid, title, year_established FROM public.rating_orgs;
Analytics and Reports Menu
Top Rated Games:
- SELECT g.name, AVG(r.score) as avg_score FROM public.games g JOIN
public.reviews r ON g.gameid = r.gameid GROUP BY g.name ORDER BY avg_score DESC
LIMIT 10;
Most Reviewed Games:
- SELECT g.name, COUNT(r.reviewid) as review_count FROM public.games g JOIN
public.reviews r ON g.gameid = r.gameid GROUP BY g.name ORDER BY review_count
DESC LIMIT 10;
Developer and Publisher Relations Menu
Games by Developer:
- SELECT g.gameid, g.name FROM public.games g JOIN public.game_developers gd ON
g.gameid = gd.gameid WHERE gd.developerid = %s;
Games by Publisher:
```

```
- SELECT gameid, name FROM public.games WHERE publisherid = %s;
Systems by Publisher:
- SELECT platformid, platformname, year_launched FROM public.platforms WHERE
publisherid = %s;
```

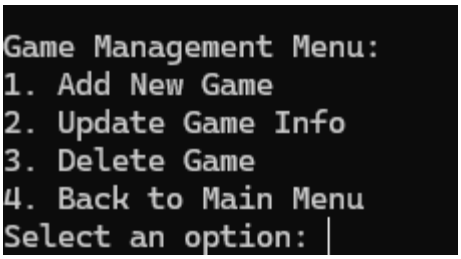
Part 4:

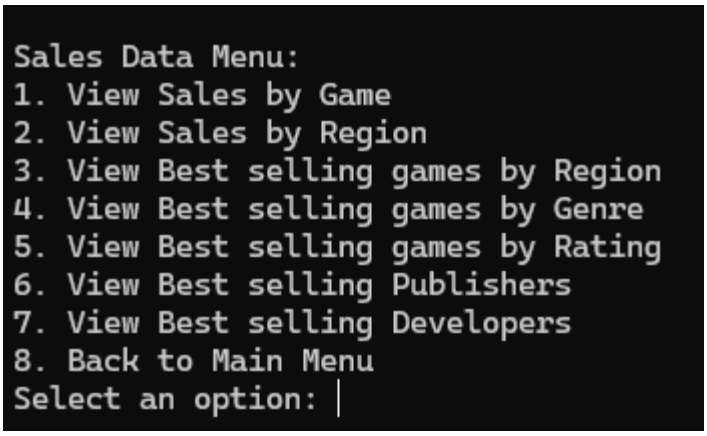
CLI menus:

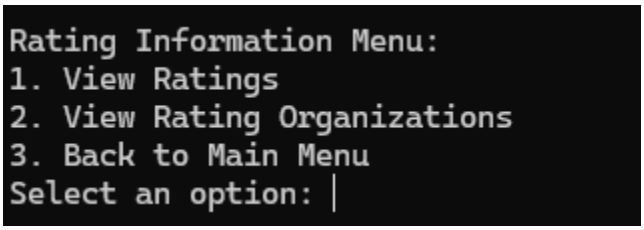
```
Main Menu:
1. View Data
2. User Management
3. Game Management
4. Review Management
5. Sales Data
6. Rating Information
7. Analytics and Reports
8. Search
9. Developer and Publisher Relations
10. Exit
Select an option:
```

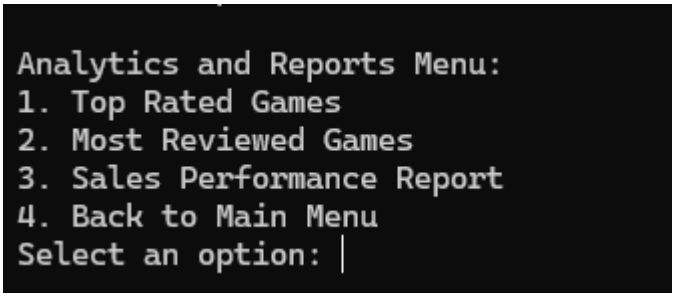
```
View Data Menu:
1. Genres
2. Publishers
3. Developers
4. Platforms
5. Console Generations
6. Games
7. Reviews
8. Regions
9. Back to Main Menu
Select an option:
```

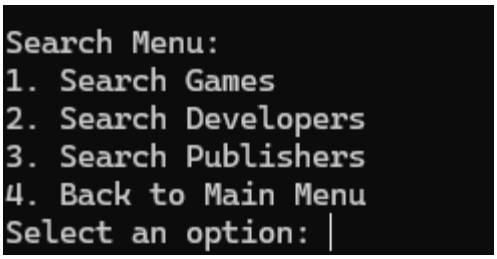
```
User Management Menu:
1. Create User
2. Update User Info
3. Delete User
4. Back to Main Menu
Select an option:
```

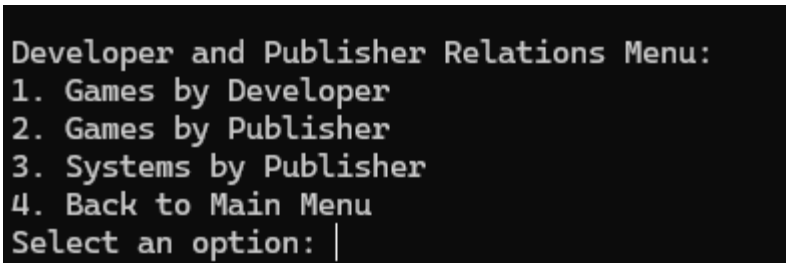

Game Management Menu:
1. Add New Game
2. Update Game Info
3. Delete Game
4. Back to Main Menu
Select an option:


Sales Data Menu:
1. View Sales by Game
2. View Sales by Region
3. View Best selling games by Region
4. View Best selling games by Genre
5. View Best selling games by Rating
6. View Best selling Publishers
7. View Best selling Developers
8. Back to Main Menu
Select an option:


Rating Information Menu:
1. View Ratings
2. View Rating Organizations
3. Back to Main Menu
Select an option:


Analytics and Reports Menu:
1. Top Rated Games
2. Most Reviewed Games
3. Sales Performance Report
4. Back to Main Menu
Select an option:


Search Menu:
1. Search Games
2. Search Developers
3. Search Publishers
4. Back to Main Menu
Select an option:


Developer and Publisher Relations Menu:
1. Games by Developer
2. Games by Publisher
3. Systems by Publisher
4. Back to Main Menu
Select an option: