Introduction to Database Systems - HW3

Name: 成俊宏
ID: 109550157

Motivation

I am an aficionado of games. In my daily life, I love to investigate gaming skills and enjoy playing new games. However, there is always an end to each game. I feel depressed when I finished a game because I cannot discover anything else from this game. Although I love playing games, I do not like to find new games to play. Since there are too many boring games in the twenty-first century, finding new games is annoying. Meanwhile, I come up with the database final project. I would like to build an application that can help me search for new games and enjoy the games!

Application Description

I use a LINE bot to be the prototype of my application because LINE is a popular application.

The name of the LINE bot is GameKing. It mainly provides two functions to query for interesting games, `"Game Information (遊戲資訊)"` and `"Game Ranking (遊戲排名)"`.

The first function is called `"Game Information (遊戲資訊)"`. The main use of this function is to know more about some games. After querying for the `"Game Information"`, GameKing will list some popular games and provide the details information about those games. Also, you can provide the game that you want to search and the bot will give the information to you. However, this function only provides the details of a designated game, it does not provide any new games for the users if they don't know the exact name of new games.

Hence, there is the second function is called `"Game Ranking (遊戲排名)"`. The main use of this function is to recommend some games that are the rage recently in different categories. For example, `"Top Three Sales"` is a category in the LINE bot. It will recommend the games whose sales are the top three recently. Therefore, it can provide popular and new games. In addition, if you want to know more about these games, you can use the `"Game Searching"` function to obtain more information and decide to download and enjoy these games!

Data Sources and Import Method

Ways of connecting databases in Python

First, I save the secrets in `rds_config.py`, and use it like the following way.

```
# rds settings
rds_host  = rds_config.rds_host
username  = rds_config.db_username
password  = rds_config.db_password
db_name   = rds_config.db_name
db_port   = rds_config.db_port
```

I use two library to connect the database: `psycopg2` and `sqlalchemy`.

`psycopg2` can use `connect()` to establish the connection.

```
conn = psycopg2.connect( """database credentials here..""" )
```

And it can use `connect().cursor().execute()` to run raw SQL.

```
with conn.cursor() as cur:
    cur.execute(open("raw.sql", "r").read())
    conn.commit()
```

`sqlalchemy` provides ORM to perform operations on the table.

`declarative_base` is used to provide the base class for the declarative object of our table, and `sessionmaker` is used to establish a session for perform operations.

```
from sqlalchemy.orm import (
    declarative_base,
    sessionmaker
)
```

The object should be obtained by the following steps:

- `create_engine`: create an engine to establish the connection with the provided credentials.

  ```
  def connect(user, password, db, host='localhost', port=5432):
      '''Returns a connection and a metadata object'''
      url = 'postgresql://{}:{}@{}:{}/{}'
      url = url.format(user, password, host, port, db)

      con = sqlalchemy.create_engine(url, client_encoding='utf8')
      return con

  con = connect(username, password, db_name, rds_host, db_port)
  ```

- `class <Name>`: bind on the engine and auto load the table.

  ```
  Base = declarative_base()
  class Video_Games(Base):
      __table__ = Table('video_games', Base.metadata, autoload=True,
  autoload_with=con)
  ```

- `sessionmaker` : bind on the engine and make a session.

```
Session = sessionmaker(bind=con)
session = Session()
```

And the query can be perform using the object:

```
for instance in session.query(Video_Games):
    cnt += 1
    data = instance.__dict__ # return as dictionary
```

Import Records From csv File

At the beginning, I use the csv file from this website as the data source:

You can notice that I first create a table called `temp` without any constraints.
(Referring to `importData/preprocess.sql` .)

```
temp (
    "Name"                TEXT,
    "Platform"            TEXT,
    "Year_of_Release"     TEXT,  --- INT
    "Genre"               TEXT,
    "Publisher"           TEXT,
    "NA_Sales"            NUMERIC(7, 3),
    "EU_Sales"            NUMERIC(7, 3),
    "JP_Sales"            NUMERIC(7, 3),
    "Other_Sales"         NUMERIC(7, 3),
    "Global_Sales"        NUMERIC(7, 3),
    "Critic_Score"        INT,
    "Critic_Count"        INT,
    "User_Score"          TEXT,  --- NUMERIC(7, 3),
    "User_Count"          INT,
    "Developer"           TEXT,
    "Rating"              TEXT
);
```

Then I import the data to `temp` table for preprocessing.
(Referring to `importData/utils.py/recoverFromCSV` .)

```
def recoverFromCSV(conn):
    # Create temp Table
    # ...

    # Import from csv to temp
    copy_sql = """
          COPY temp FROM stdin WITH CSV HEADER
          DELIMITER as ','
          """

    file_name = "Video_Games.csv"
    with open(file_name, 'r') as f:
        # next(f)
        conn.cursor().copy_expert(sql=copy_sql, file=f)
        conn.commit()

    # import temp to Video_Games
    # ...
```

And then move the records from `temp` to the constrained table `video_games`.
(Referring to `importData/importData.sql`.)

Updates Records From Web Scraper

Apart from importing from csv, I also tried to use a web scraper to update my database.

The link is the main website for the web scraper to grab the data.

And the details of the web scraper can refer to `importData/scrape.py`.

```
def scrapeInsert(conn):
    # use web scraper to fetch records
    df = scrapeTodf()

    # import dataframes into temp table
    # to_sql function can easily handle this
    conn = fetchConn().connect()
    df.to_sql('temp', con=conn, if_exists='replace', index=False)

    # import temp to Video_Games
    with conn.cursor() as cur:
        cur.execute(open("importData.sql", "r").read())
        conn.commit()
```

Database Schema

The database schema only contains one table, also known as the original data.

The table is defined as below:

```
Video_Games (
    "Name"                TEXT,
    "Platform"            TEXT,
    "Year_of_Release"     INT,
    "Genre"               TEXT,
    "Publisher"           TEXT,
    "NA_Sales"            NUMERIC(7, 3),
    "EU_Sales"            NUMERIC(7, 3),
    "JP_Sales"            NUMERIC(7, 3),
    "Other_Sales"         NUMERIC(7, 3),
    "Global_Sales"        NUMERIC(7, 3),
    "Critic_Score"        INT,
    "Critic_Count"        INT,
    "User_Score"          NUMERIC(7, 3),
    "User_Count"          INT,
    "Developer"           TEXT,
    "Rating"              TEXT,

    PRIMARY KEY ("Name", "Platform", "Year_of_Release")
);

CREATE INDEX indexName
ON Video_Games ("Name");

CREATE INDEX indexYear
ON Video_Games ("Year_of_Release");

CREATE INDEX indexGenre
ON Video_Games ("Genre");
```

I use the original attributes because of the following reasons:

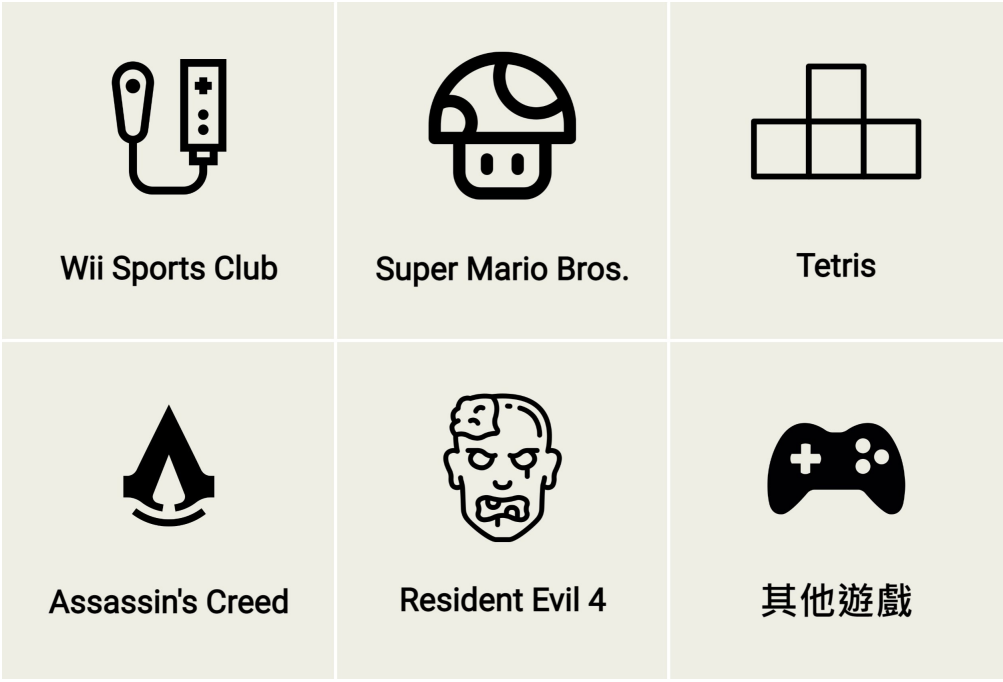
- I've tested for functional dependencies, but those functional dependencies I tried fail to hold.
  So, I **don't** need to perform **normalization** on the table.
  - `"Platform"` → `"Publisher"`
  - `"Platform"` → `"Developer"`
  - `"Publisher"` → `"Developer"`
- Also, the primary key is not just "Name", because the same game may be distributed on different platforms. I chose **`("Name", "Platform", "Year_of_Release")` as the primary key.**
  - There is **no constraint needed to be added**.
- For indexing, I only **index on `"Name"`, `"Genre"`, and `"Year_of_Release"`**, separately.
  - Because they are frequently used in `WHERE` clause.
  - Our application frequently reads, not writes. So indexing will help a lot.
  - Before the web scraper is added, the series of `Sales` contains a few `NULL` values. But tons of `NULL` in a series of `Sales` come when I use web scraper to fetch the data.
    So I **don't index on series of `Sales` or the performance will degrade**.
  - I don't use `UNIQUE INDEX` because the attributes I chose to index have duplicates.
- There are some different types between `temp` and `Video_Game` table, and we need to cast them when transiting the records.
  - Cast from one type to another type

Application Functions

There are mainly two functions in this application: `"Game Information (遊戲資訊)"` and `"Game Ranking (遊戲排名)"`.

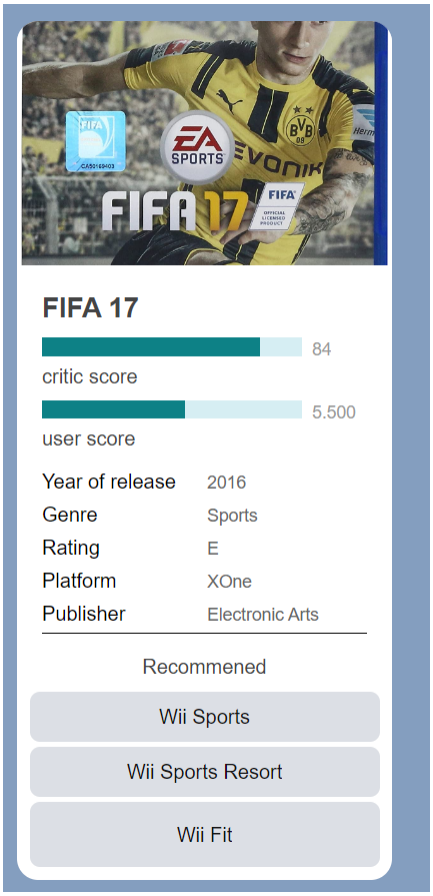
Game Information

For `"Game Information"`, it provides a way to query for the detail of the game by the name of it.



If we press the button, the user will send a message `q1 <msgText>`, which is the way to trigger the function.

For example, if we type `q1 FIFA 17`, the line bot will send the following flex message:



The ways to get the information is easy, just filter by `"Name"` :

```
# Assume msgText = 'FIFA 17'
for instance in session.query(Video_Games).filter(Video_Games.Name==msgText):
    data = instance.__dict__
```

The corresponding SQL code is:

```
SELECT *
FROM Video_Games
WHERE "Name" = <msgText>
```

We can notice that there are 3 recommended games of the same `"Genre"` in the foot section.

The way to implement this, just filter it by the same `"Genre"` and order the records by `"Global_Sales"` .

```
cnt = 4
for instance in session.query(Video_Games).\
    filter(Video_Games.Genre==data['Genre'], Video_Games.Global_Sales != None).\
    order_by(Video_Games.Global_Sales)[-3:]:
    cnt -= 1
    name = instance.Name
    data['Name' + str(cnt)] = name
```

The corresponding SQL code is:

```
SELECT *
FROM Video_Games
WHERE "Genre" = <msgText's "Genre"> AND "Global_Sales" IS NOT NULL
ORDER BY "Global_Sales" DESC
LIMIT 0, 3
```

Game Ranking

For `"Game Ranking"` , provides a way to find the top three by some attribute like `"Global_Sales"` , `"User_Score"` , and so on.

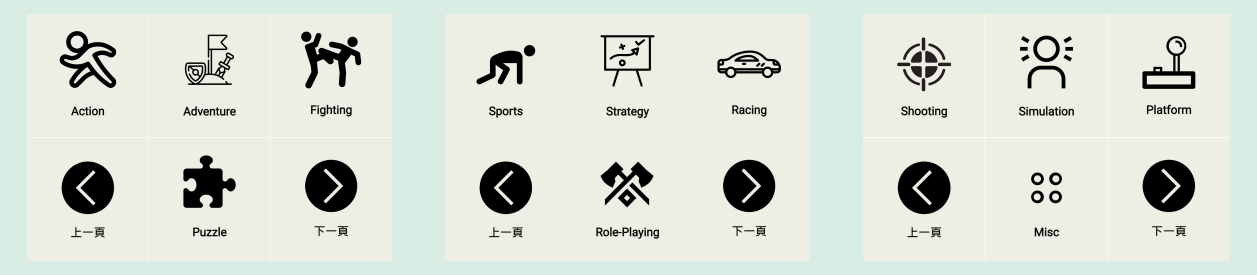
For `"總銷量前三名"`, `"評論者評分前三名"`, and `"使用者評分前三名"`, we can use the method which is similar to what we have done for recommending another 3 games.

```
cnt = 4
for name in session.query(Video_Games.Name).filter(orderby !=
None).order_by(orderby)[-3:]:
    cnt -= 1
    name = name[0]
    data['Name' + str(cnt)] = name
```

The corresponding SQL code is:

```
SELECT *
FROM Video_Games
WHERE <orderby> IS NOT NULL
ORDER BY <orderby> DESC
LIMIT 0, 3
```

For `"按種類排名"`, the users can select the `"Genre"` they want to filter by.



Also easy to filter them by `"Genre"`.

```
# Assume msgText is <Genre>
for name in session.query(Video_Games.Name).\
    filter(Video_Games.Genre == msgText, Video_Games.Global_Sales != None).\
    order_by(Video_Games.Global_Sales)[-3:]:
        cnt -= 1
        name = name[0]
        data['Name' + str(cnt)] = name
```

The corresponding SQL code is:

```
SELECT *
FROM Video_Games
WHERE "Genre" = <msgText> AND "Global_Sales" IS NOT NULL
ORDER BY "Global_Sales" DESC
LIMIT 0, 3
```

For `"地區銷量前三名"`, the following images will be shown:

We need to support 3 regions for querying.

We can use the mechanism like `"總銷量前三名"` to handle these cases.

Last, for `"年份後前三名"`, it will provide a hint message `q5 <年份>` to query.

This type of query should rank the games after the provided years.

For example, if we type `q5 2016`, they it will perform the following query:

```
for name in session.query(Video_Games.Name).\
    filter(Video_Games.Year_of_Release >= year, Video_Games.Global_Sales !=
None).\
    order_by(Video_Games.Global_Sales)[-3:]:
        cnt -= 1
        name = name[0]
        data['Name' + str(cnt)] = name
```

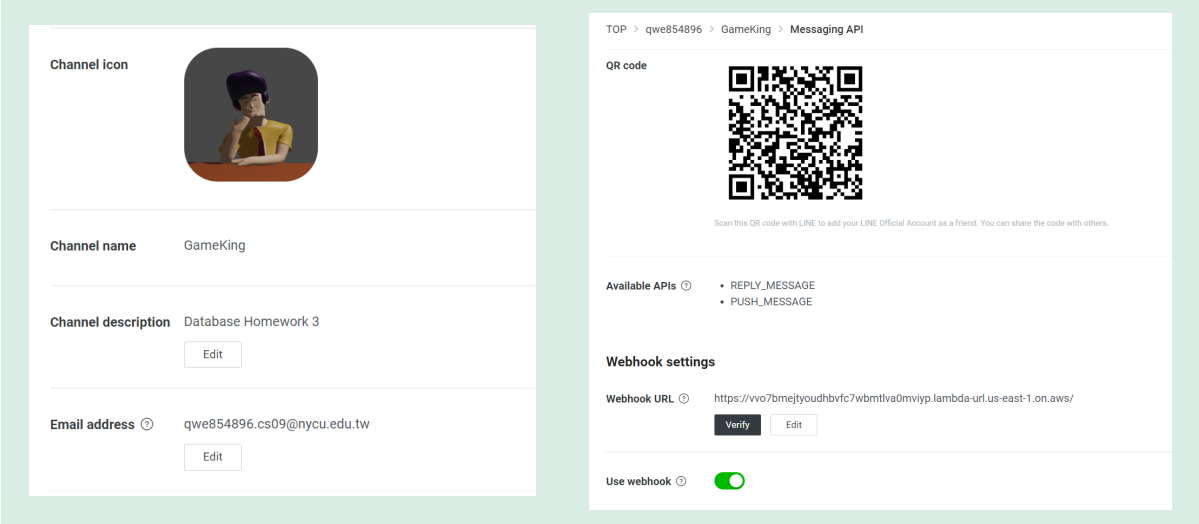
The corresponding SQL code is:

```
SELECT *
FROM Video_Games
WHERE "Year_of_Release" >= <year> AND "Global_Sales" IS NOT NULL
ORDER BY "Global_Sales" DESC
LIMIT 0, 3
```

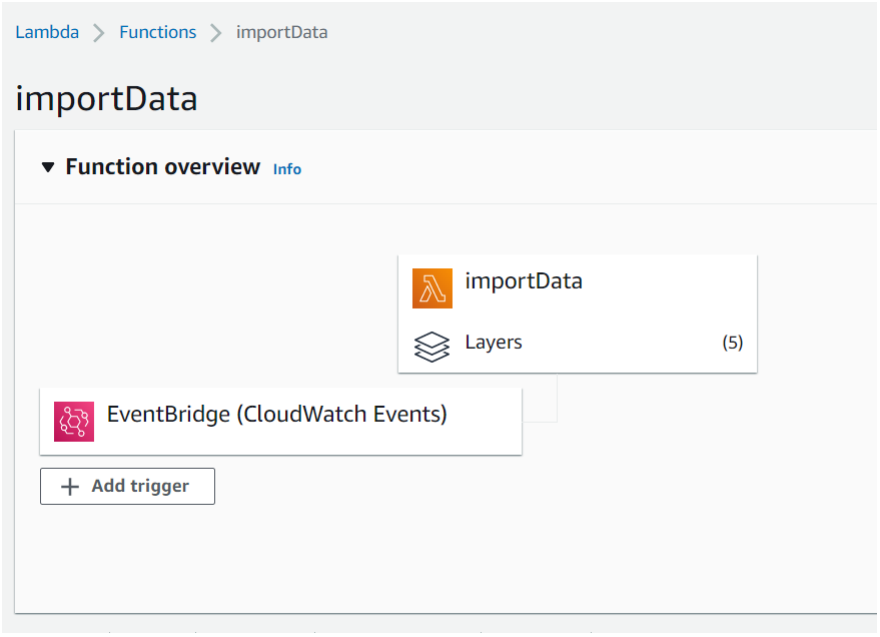
Services on AWS

The following steps describe the way I finish this homework:

- Use RDS to create PostgreSQL database
- Use VPC to set up the suitable network access
- Use Systems Manager > Automation and CloudWatch to debug.
- Use Lambda to deploy the database, the records, and the application.



  - On the Line Developer page, I need to provide the URL as a webhook.
  - It can be evidence that I have deployed my line bot on AWS.
  - Also, you can try to interact with the line bot from the above QR code.
  - The packages I used are provided in the source code.
- Use EventBridge to trigger update on database once a week.



The Works I've done and I Want to Mention

- Design flex message, a type of message LINE provided, to present the information of games.
- Design richmenus for users to easily interact with linebot.
- Use `jinja2` to substitute the variables in a json template.