Software Documentation

המידע (הדטה)

המידע נלקח ממאגר המידע: Anime Datasets with reviews - MyAnimeList.

https://www.kaggle.com/datasets/marlesson/myanimelist-dataset-animes-profiles-reviews?resource=download&select=reviews.csv

אשר נמצא באתר Kaggle.

- ניקוי ופלטור הטבלאות.

הפלטורים והניקויים נעשו ע"י פקודות מניפולציית טקסט פשוטות של לינוקס כגון **Sed** ו-**Awk** כמו כן גם בשימוש פשוט באקסל לטובת מחיקת עמודות בהן לא היה לנו צורך.
כמו כן השתמשנו גם בפונקציונליות של פקודות החלפה במעבד התמלילים **VIM** לטובת החלפות בתוך הקובץ.

אתן שתי דוגמאות:

1. שינוי שמות בחודשים למספרים

בטבלה profile, ישנה עמודה של תאריכי לידה.
אשר מופיעים בצורה:

Nov 10, 1994

ע"י פעולת ההחלפה במעבד התמלילים VIM, נחליף בין השם Nov למספר 11.

:%s/Nov/11/g

פקודה זו מחליפה בין כל מחרוזת "Nov" למחרוזת "11", בכל הקובץ.
פקודה זו בוצעה עבור שאר החודשים כמובן.

2. <u>שינוי הסדר של התאריך כך שיופיע כyyyy-mm-dd.</u>

גם כאן נשתמש בפעולת החלפה של VIM המבוססת על ביטויים רגולרים:
המצב ההתחלתי:

11 10, 1994

הפקודה:

%s/(\d\d\) (\d\d), (\d\d\d\d)/\3 \1 \2

- לנוחות לא כתבתי את הסלאשים של שעושים escape לתווים מיוחדים.

בנוסף, לכל אנימה בדטה סט יש רשימה של ג'אנרים.
אותה הוצאנו מהטבלה הראשית, ויצרנו טבלה חדשה כך שלכל אנימה יהיו מספר שורות עם הג'אנרים השונים.
כלומר, אם בטבלה הראשית היה:

Anime_id, [“Action”, “Comedy”]

ע"י סקריפט פשוט יצרנו את הטבלה הבאה:

Anime_id	anime_genre
1	Action
1	Comdey

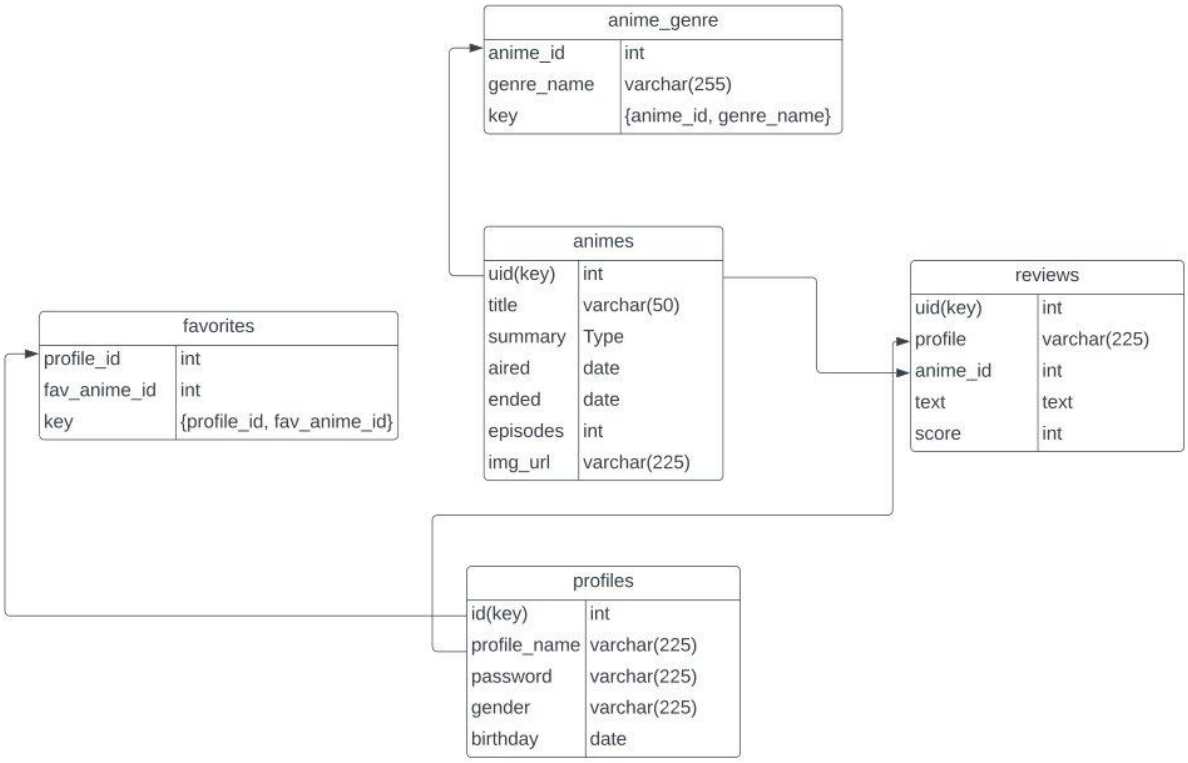
כך אנו מתמודדים עם עמודות בהן הערכים הינם רשימות.

- **<u>נתונים פקטיבים</u>**

כיוון שיצרנו מערכת הרשמה לאתר, הוספנו עמודת סיסמא בטבלת היוזרים של האתר.

הערה: כפי שנאמר לעיל, הסקריפטים לעיבוד המידע היו פעולות פשוטות של Sed ופעולות החלפה בVIM ולא הופעלו סקריפטים חיצונים מורכבים.

סכמת מסד הנתונים:



שאילתות:

- ***getUserId***

```
SELECT id FROM profiles
WHERE profile_name = ?
```

params: [profile_name]

returns the id of given profile name.

- ***getAllAnimesNames***

```
SELECT uid,title,img_url
FROM animes
ORDER BY title
```

returns: all anime names (with uid and img url) ordered by title.

- ***getLastReviewUid***

```
SELECT uid FROM reviews
ORDER BY uid DESC
LIMIT 1
```

return the last largest uid of a profile in profiles table.

- ***addReview***

```
INSERT INTO reviews (uid, profile, anime_uid, text, score)
VALUES(?,?,?,?,?)
```

params: [uid, profile, anime_uid, text,score)

inserts a new review.

- ***get profile name from id***

```
SELECT profile_name
FROM profiles
WHERE id = ?
```

params: [id]

returns the profile name of a that corresponds to the given profile id.

- ***updateUserReview***

```
UPDATE reviews
SET text = ?, score = ?
WHERE uid = ?
```

params: [rev_text, score, rev_id]

updates user review according to its id, and sets it to new ‘text’ and ‘score’.

- ***getNumOfUsers***

```
SELECT COUNT(profile_name) AS total
```

```
FROM profiles
```

return total number of users

- ***getUserReviews***

```
SELECT profile,reviews.uid AS rev_id,title,img_url,score,text
FROM reviews
JOIN animes
ON reviews.anime_uid = animes.uid
WHERE profile=?
```

params: [profile]

returns gets given profile's reviews, together with the review id, the profile name, the reviewed anime img url and the review text.

- ***isUsernameTaken***

```
SELECT * FROM profiles
WHERE profile_name = ?
```

params: [username]

by returning one row or none, we can check where given profile name exits in the database.

- ***createUser***

```
INSERT INTO profiles(id,profile_name,password,gender,birthday)
VALUES(?,?,?,?,?)
```

params: [id, profile_name, password, gender, birthday]

adding a new user into profiles table.

- ***addAnimeToFav***

```
INSERT INTO favorites (profile_id, fav_anime_id)
VALUES(?,?)
```

params: [user_id, anime_id]

adding a new favorite anime to a user. both are given as arguments.

- ***check credentials***

```
SELECT * FROM profiles
WHERE profile_name = ? AND password = ?
```

params: [username, password]

return wheter a given username and password exits in the db when trying to login.

- ***getAnimeAvgScroe***

```
SELECT avg(score) AS avg FROM reviews
```

```
WHERE anime_uid = (
SELECT uid FROM animes
WHERE title=?
)
```

params: [anime_name]

return anime avg score based on users scores in their reviews.

- ***getTopAnimes***

```
SELECT title, avg_score, img_url FROM
   (SELECT anime_uid, avg_score FROM
      (SELECT anime_uid,COUNT(DISTINCT(profile)) AS
times_reviewed, AVG(score) AS avg_score
      FROM reviews
      GROUP BY anime_uid
      having times_reviewed > ?
      ORDER BY avg_score DESC
      LIMIT ?
) AS t ) AS t2
inner JOIN animes
on t2.anime_uid = animes.uid
ORDER BY avg_score DESC ;
```

params: [min_rev, k]

given min_rev, and k, this query will return the top K animes with the best avg score according to users review's scores. to be a 'top anime' a minimum number of reviews must be met. ('min_rev'). all

animes with at least ‘min_rev’ can be calculated and checked for their score against other ‘top animes’ candidates. *

- ***getBestScoreReview***

```
SELECT title,img_url,text FROM animes
JOIN (
SELECT anime_uid,text,score FROM reviews
JOIN (
SELECT profile_name FROM profiles WHERE id = ?
) AS t
ON reviews.profile = t.profile_name
) AS t2
ON anime_uid = uid
ORDER BY score DESC LIMIT 1
```

params: [profile_id]

returns the review with the highest score of a user (based on it’s id as an argument).

- ***getUserFavAnimes***

```
SELECT title, img_url FROM favorites
inner JOIN animes
ON favorites.fav_anime_id = animes.uid
WHERE profile_id = ?
```

params: [id]

return user’s (according to given user’s id) favorite animes.

- ***getAllGenres***

```
SELECT DISTINCT genre_name
FROM anime_genre
```

returns all genres in the db.

- ***getAnimeByGenreList***

```
SELECT uid, title, summary, aired, ended, episodes, img_url
FROM (
   SELECT anime_id FROM anime_genre
   WHERE genre_name in (?)
) AS t
JOIN animes ON t.anime_id = animes.uid;
```

params: [genres]

return all anime that correspondes to a given genre list.

- ***getMostActiveUsers***

```
SELECT profile, COUNT(profile) AS num_of_reviews
FROM reviews
GROUP BY profile
ORDER BY num_of_reviews DESC LIMIT ?
```

params: [k]

return the most active users based on their number of reviews

- ***getUsersNumberOfReviews***

```
SELECT COUNT(*) AS total
FROM reviews
JOIN(
SELECT profile_name FROM profiles WHERE id = ?
) AS t
ON reviews.profile = t.profile_name
```

params: [id]

returns number of reviews of a user based on it's given id.

- ***getUserTotalFavAnimes***

```
SELECT COUNT(*) AS total
FROM favorites
WHERE profile_id = ?
```

params: [id]

returns total number of favorites animes of a user (that correspondes to given id)

- ***isAnimeExists***

```
SELECT * FROM animes
WHERE UPPER(title)
LIKE UPPER(?)
```

params: [anime_name]

returns all anime that are similar (case insensitive) to given anime name

- ***inert anime genre***

```
INSERT INTO anime_genre (anime_id, genre_name)
VALUES(?,?)
```

params: [id, g]

inseret a new anime and it's genre into the anime_genre table.

- ***get last anime uid***

```
SELECT uid
FROM animes
ORDER BY uid DESC LIMIT 1
```

returns the larget anime uid in the db.

- ***insert new anime***

```
INSERT INTO animes (uid, title, summary, aired, ended,
episodes, img_url)
VALUES (?,?,?,?,?,?,?)
```

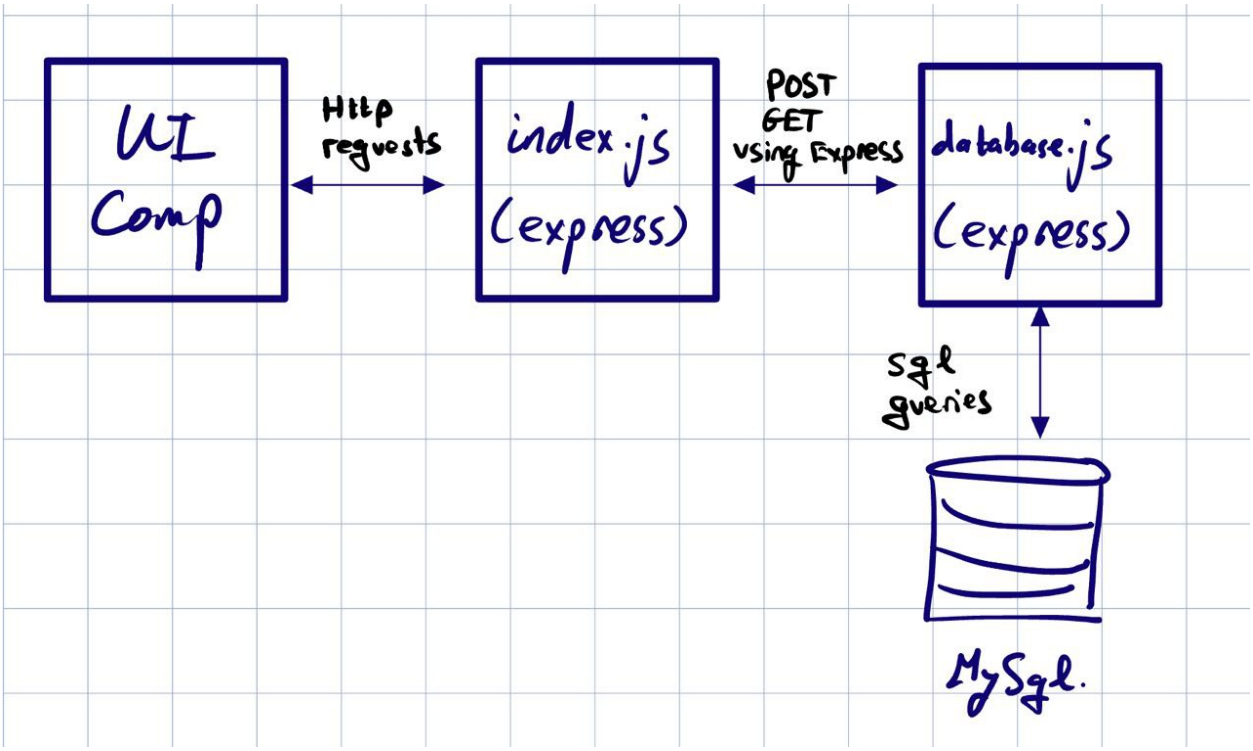
params: [uid, title, summary, aired date, end date, number of episodes, img url]

inserts a new anime into the animes table.

- **Get total animes in all of the genres**

SELECT count(*) As total_animes,genre_name FROM anime_genre GROUP BY genre_name ORDER BY total_animes Desc

אופן ביצוע השאילתות:



נסביר כיצד עובדות השאילתות באפלקציה שלנו.
כאמור, האפלקציה הינה אפלקציית אינטרנט. החלק של הUI מבוצע ע"י react framwork ובעזרת הספריה axios אנו מבצעים שאילתות HTTP לסרבר שלנו (index.js).

הסרבר שלנו משתמש ב Express framwork.
כאשר נבצע שאילתת HTTP מהUI, הסרבר יקלוט את הבקשה, וישלח בקשה למסד הנתונים, זאת בעזרת שכבה מקשרת, database.js אשר מרכזת בתוכה פונקציות אשר מבצעות שאילתות sql לסרבר mysql.