**ASSIGNMENT 3**

**1. ABSTRACT**

Database normalization is the process of organizing the attributes and tables of a relational database to minimize data redundancy. In this assignment the database (Movies Database) built in the previous assignment is normalized (1NF, 2NF, 3NF). The final SQL of a normalized database is presented. Views for all of your use-cases are created. Five indexes, functions and procedures are created on the normalized database.

**2. NORMALIZATION:**

The Database before normalization contained four tables (1 for company entity, 1 for producer entity, 2 for consumer entity) and was as follows:

CREATE TABLE company(

title VARCHAR(255) PRIMARY KEY NOT NULL,

Release\_date DATE,

Overview VARCHAR(255),

Adult VARCHAR,

Vote\_count INT,

Vote\_average FLOAT,

Popularity FLOAT

);

CREATE TABLE producer(

title VARCHAR(255) PRIMARY KEY NOT NULL,

production\_companies VARCHAR(255),

production\_countries VARCHAR(255),

revenue INTEGER

);

CREATE TABLE consumer(

Name VARCHAR(255) PRIMARY KEY NOT NULL,

User\_id INTEGER,

Followers INTEGER,

Post STRING,

Hashtags STRING,

Posted\_time STRING,

Favourites INTEGER

);

CREATE TABLE tweets(

title VARCHAR(255) PRIMARY KEY NOT NULL,

name VARCHAR(255),

FOREIGN KEY(name) REFERENCES consumer(Name)

);

**SAMPLE DATABASE TABLES BEFORE NORMALIZATION:**

Company – Movies



Producer – Production house



Consumer – Movie viewers





After Normalization, the database is modified and follows the below schema:

Company Entity:

CREATE TABLE company1 (

Movie\_id INTEGER PRIMARY KEY AUTOINCREMENT

NOT NULL,

Movie\_title VARCHAR (255),

Release\_date DATE,

Overview VARCHAR (255),

Vote\_count INT,

Vote\_average FLOAT,

Popularity FLOAT

);

Production Entity:

CREATE TABLE production1 (

id INTEGER PRIMARY KEY

NOT NULL,

title VARCHAR (255),

revenue INTEGER

);

CREATE TABLE production\_company (

title VARCHAR (255) REFERENCES production1 (title),

production\_company\_name VARCHAR (255),

FOREIGN KEY (title) REFERENCES production1 (title)

);

CREATE TABLE production\_country (

title VARCHAR (255) REFERENCES production1 (title),

production\_country\_name VARCHAR (255),

FOREIGN KEY (title) REFERENCES production1 (title)

);

Consumer Entity:

CREATE TABLE consumer\_user\_info (

User\_id INTEGER PRIMARY KEY

NOT NULL,

Name VARCHAR (255),

Followers INTEGER,

title VARCHAR (255),

User\_name VARCHAR (255)

);

CREATE TABLE consumer\_tweet\_info (

User\_id INTEGER,

Post STRING,

Hashtags STRING,

Posted\_time STRING,

Favourites INTEGER,

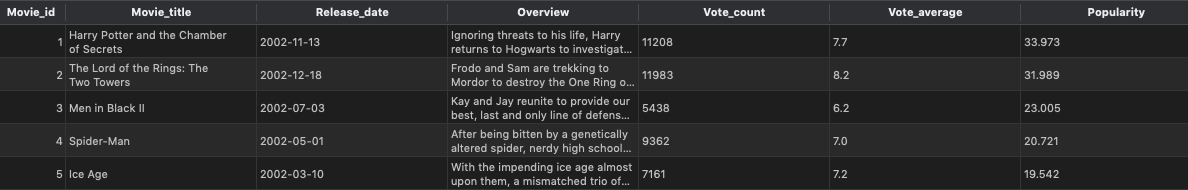
FOREIGN KEY (User\_id) REFERENCES consumer\_user\_info (User\_id)

);

﻿

**SAMPLE DATABASE TABLES AFTER NORMALIZATION:**

Company – Movies

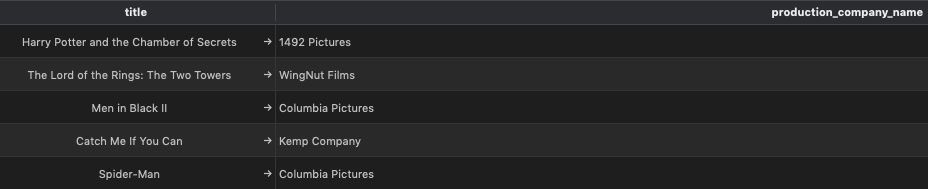


Producer:

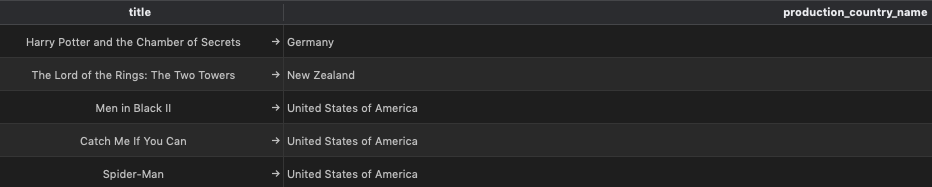
Production -



Production Company -

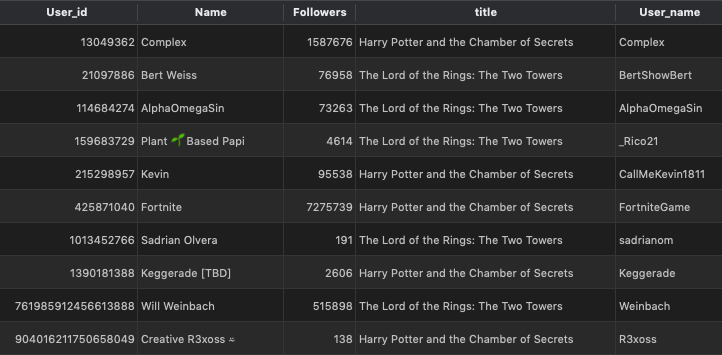


Production Country –

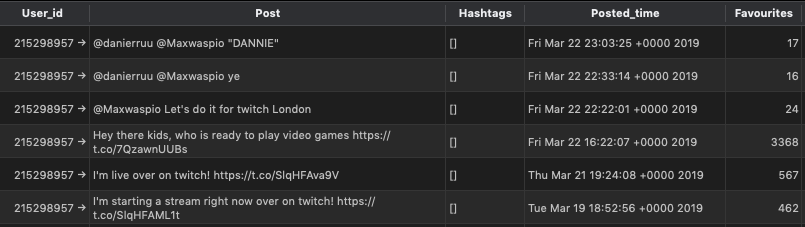


Consumer – Movie viewers

Consumer User Info -



Consumer tweets info -



Normalization steps performed:

Table Producer:

1. The attributes production\_companies and production\_countries were multivalued attributes, so new tables are created for these such as Production Company and Production Country connected to the Production table via Foreign keys.

Table Consumer:

1. Consumer table had information that did not depend on the primary key and hence the information is split to two tables such as Consumer User Info and Consumer tweets info
2. The transitive dependencies in the Consumer table has been eliminated.

Normalization is done to satisfy the following conditions:

First normal form (1NF)

• Each table has a primary key: minimal set of attributes which can uniquely identify a record

• The values in each column of a table are atomic (No multi-value attributes allowed)

• There are no repeating groups: two columns do not store similar information in the same table.

Second normal form (2NF)

• All requirements for 1st NF must be met.

• No partial dependencies.

• No calculated data

Third normal form (3NF)

• All requirements for 2nd NF must be met.

• Eliminate fields that do not directly depend on the primary key; that is no transitive dependencies.

Company:

|  |
| --- |
| Title |
| Release\_date |
| Overview |
| Adult |
| Vote\_count |
| Vote\_average |
| Popularity |

|  |
| --- |
| Movie\_ID |
| Title |
| Release\_date |
| Overview |
| Vote\_count |
| Vote\_average |
| Popularity |

Producer:

|  |
| --- |
| Title |
| Production\_companies |
| Production\_countries |
| Revenue |

|  |
| --- |
| Movie\_ID |
| Title |
| Revenue |

|  |
| --- |
| Title |
| Production\_company\_names |

|  |
| --- |
| Title |
| Production\_country\_names |

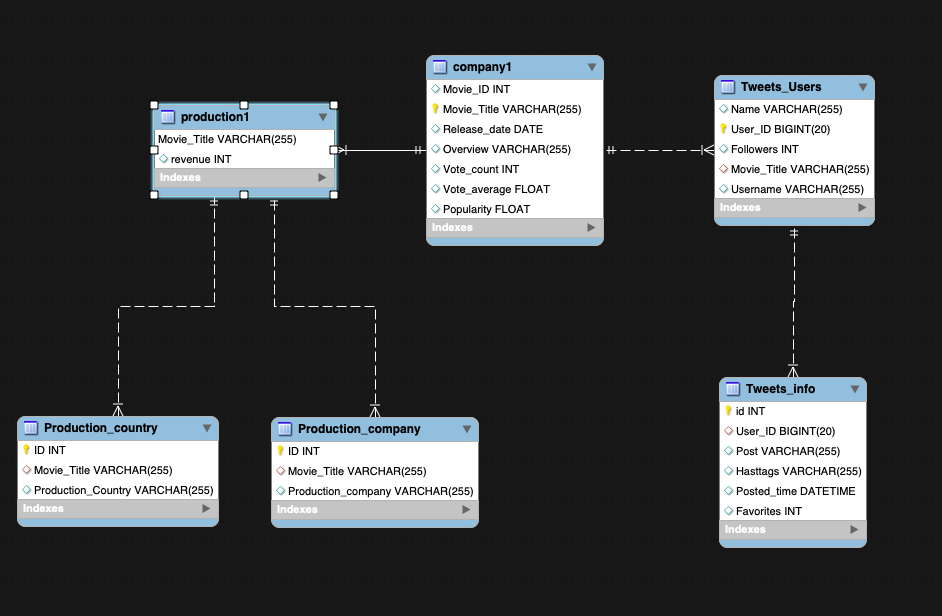
Consumer:

|  |
| --- |
| User\_ID |
| Name |
| Folowers |
| Title |
| User\_Name |

|  |
| --- |
| Name |
| User\_ID |
| Folowers |
| Post |
| Hashtags |
| Posted\_time |
| Favorites |

|  |
| --- |
| User\_ID |
| Post |
| Hashtags |
| Posted\_time |
| Favorites |

|  |
| --- |
| Title |
| Screen\_Name |



**3. VIEWS FOR ALL USE-CASES**

**Use case 1**: Query for finding name and revenue of the movie

View:

create view as ﻿Query\_UC\_1\_Company\_Production\_View\_1

select company1.Movie\_title, company1.Release\_date, production1.revenue

from company1, production1

where company1.Movie\_title = production1.title

**Use case 2:** Query for overview and popularity

View:

create view as ﻿Query\_UC\_2

select company1.Movie\_title, company1.Release\_date, company1.Overview, company1.Popularity

from company1

**Use case 3:** Query for finding release\_date, overview, vote\_count, vote\_average, popularity

View:

create view as ﻿Query\_UC\_3

select company1.Movie\_title, company1.Release\_date, company1.Overview, company1.Vote\_count, company1.Vote\_average, company1.Popularity

from company1

**Use case 4 & 5:** Query for finding the movies and their revenue

View:

create view as ﻿Query\_UC\_4\_5

select production1.title, production1.revenue

from production1

**Use case 6:** Query for finding twitter user with maximum followers

View:

create view as ﻿Query\_UC\_6

select consumer\_user\_info.User\_id, consumer\_user\_info.User\_name, consumer\_user\_info.Followers

from consumer\_user\_info

**Use case 7:** Query to get the tweets and the posted time of the tweets of the user with maximum followers

View:

create view as ﻿Query\_UC\_7

select consumer\_user\_info.User\_id, consumer\_user\_info.User\_name, consumer\_user\_info.Followers, consumer\_tweet\_info.Post, consumer\_tweet\_info.Posted\_time

from consumer\_user\_info, consumer\_tweet\_info

where consumer\_user\_info.User\_id=consumer\_tweet\_info.User\_id

**Use case 8:** Query to find the movie with revenue among the movies tweeted by twitter users

View:

create view as ﻿Query\_UC\_8

select consumer\_user\_info.User\_id, consumer\_user\_info.User\_name, consumer\_user\_info.title, production1.revenue

from consumer\_user\_info, production1

where consumer\_user\_info.title=production1.title

**Use case 9:** Query to get the movie overview, release date and name of the twitter user for movies that the users have tweeted about

View:

create view as ﻿Query\_UC\_9

select consumer\_user\_info.User\_id, consumer\_user\_info.User\_name, consumer\_user\_info.title, company1.Release\_date, company1.Overview

from consumer\_user\_info, company1

where consumer\_user\_info.title=company1.Movie\_title

**Use case 10:** Query for the twitter user and twitter posts

View:

create view as ﻿Query\_UC\_10

select consumer\_user\_info.User\_id, consumer\_user\_info.User\_name, consumer\_tweet\_info.Post, consumer\_tweet\_info.Posted\_time

from consumer\_user\_info, consumer\_tweet\_info

where consumer\_user\_info.User\_id=consumer\_tweet\_info.User\_id

**Use case 11:** Query for title, overview, release date of the movie the twitter users posted about

View:

create view as ﻿Query\_UC\_11

select consumer\_user\_info.User\_id, consumer\_user\_info.User\_name, consumer\_tweet\_info.Post, consumer\_tweet\_info.Posted\_time

from consumer\_user\_info, consumer\_tweet\_info

where consumer\_user\_info.User\_id=consumer\_tweet\_info.User\_id

**4. INDEXES**

The following indexes are created:

1. Company table which improves the performance for use cases 1, 2, 3:

CREATE INDEX I1\_title\_release\_date ON company1 (

Movie\_title ASC,

Release\_date

);

=> Query execution time reduced from 0.002 second(s) to 0.001 second(s).

2. Company table which improves the performance for use cases 2 & 3:

CREATE INDEX I2\_title\_popularity ON company1 (

Movie\_title ASC,

Popularity

);

=> Query execution time reduced from 0.002 second(s) to 0.001 second(s).

3. Production table which improves the performance for use cases 1, 4 & 5:

CREATE INDEX I1\_Title\_Revenue ON production1 (

title,

revenue

);

=> Query execution time reduced from 0.002 second(s) to 0.001 second(s).

4. Production\_company table:

CREATE INDEX I1\_productionCompany ON production\_company (

production\_company\_name

);

=> Query execution time reduced from 0.002 second(s) to 0.001 second(s).

5. Production\_country table:

CREATE INDEX I1\_ProductionCountry ON production\_country (

production\_country\_name

);

=> Query execution time reduced from 0.002 second(s) to 0.001 second(s).

6. Consumer\_user\_info table which improves the performance for use cases 6,7,8,9,10 & 11:

CREATE INDEX I1\_UserID\_Username ON consumer\_user\_info (

User\_id,

User\_name

);

=> Query execution time reduced from 0.002 second(s) to 0.001 second(s).

7. Consumer\_user\_info table which improves the performance for use cases 8,9 & 11:

CREATE INDEX I2\_UserID\_Username\_title ON consumer\_user\_info (

User\_id,

title,

User\_name

);

=> Query execution time reduced from 0.002 second(s) to 0.001 second(s).

8. Consumer\_tweet\_info table which improves the performance for use cases 7 & 10:

CREATE INDEX I1\_post\_postedtime ON consumer\_tweet\_info (

Post,

Posted\_time

);

=> Query execution time reduced from 0.002 second(s) to 0.001 second(s).

**5. FUNCTIONS**

\* Function (1): number\_of\_tweets() using 'consumer\_user\_info' and 'consumer\_tweet\_info'

#Description: Accepts User\_name as input and returns the total number of tweets made by the user.

#Joins used: Tables 'consumer\_user\_info' and 'consumer\_tweet\_info' are joined using 'User\_ID'.

DELIMITER $$

CREATE DEFINER=`root`@`localhost` FUNCTION `number\_of\_tweets`(user\_name text) RETURNS int

READS SQL DATA

BEGIN

DECLARE number\_of\_tweets INT;

*SELECT count(t.Post) INTO number\_of\_tweets*

*FROM consumer\_tweet\_info t, consumer\_user\_info u*

*WHERE t.User\_ID = u.User\_ID;*

*RETURN number\_of\_tweets;*

*END$$*

*DELIMITER ;*

\* Function (2): number\_of\_posts() using 'consumer\_tweet\_info'

#Description: Accepts user\_id as input and returns the total number of posts made by the user.

#Joins used: Table 'consumer\_tweet\_info' is alone used.

*DELIMITER $$*

*CREATE DEFINER=`root`@`localhost` FUNCTION `*consumer\_tweet\_info*`(user\_id varchar(255)) RETURNS int*

READS SQL DATA

*BEGIN*

*DECLARE number\_of\_posts INT;*

*SELECT count(p.Post) INTO number\_of\_posts*

*FROM consumer\_tweet\_info p*

*WHERE p.User\_ID = user\_id;*

*RETURN number\_of\_posts;*

*END$$*

*DELIMITER ;*

\* Function (3) : dates\_of\_movie\_tweets()

#Description: Accepts movie name as input and returns the concatenated list of dates(extracting only date from datetime stamp),

#Joins used:

#1.Tables 'consumer\_user\_info' and 'consumer\_tweet\_info' are joined using 'User\_ID'.

*DELIMITER $$*

*CREATE DEFINER=`root`@`localhost` FUNCTION `dates\_of\_movie\_tweets`(movie text) RETURNS text*

*READS SQL DATA*

*BEGIN*

*DECLARE dates text;*

*select group\_concat(DISTINCT(Date(t.Posted\_Time))) INTO dates*

*from consumer\_tweet\_info t, consumer\_user\_info p*

*where t.user\_ID = p.user\_ID;*

*RETURN dates;*

*END$$*

*DELIMITER ;*

*\**Function (4) : dates\_of\_user\_tweets()

#Description: Accepts User name as input and returns the concatenated list of dates(extracting only date from datetime stamp),

# Function call for movie\_names\_attached\_to\_posts\_on\_selected\_day() with '2019-03-22' as input

select Movie\_names\_attached\_to\_posts\_on\_selected\_day('2019-03-22') as 'Movie Names';

#on which tweets were made about movies.

#Joins used:

#1.Tables 'consumer\_user\_info' and ''consumer\_tweet\_info ' are joined using 'User\_ID'.

*DELIMITER $$*

*CREATE DEFINER=`root`@`localhost` FUNCTION `dates\_of\_movie\_tweets`(moviename text) RETURNS text*

*READS SQL DATA*

*BEGIN*

*DECLARE dates text;*

*select group\_concat(DISTINCT(Date(t.Posted\_Time))) INTO dates*

*from consumer\_tweet\_info t, consumer\_user\_info p*

*where t.User\_ID = p.User\_ID;*

*RETURN dates;*

*END$$*

*DELIMITER ;*

\* Function (5): Accepts revenue as the input and classifies it as high grossing or low grossing movie

#Description: Accepts revenue of the movie as input and returns if the movie is a high grosser or low grosser based on the below logic

If revenue is greater than 100000000 it is a high grosser else a low grosser

#Tables used: Production1

*DELIMITER $$*

*CREATE DEFINER=`root`@`localhost` FUNCTION `predict\_movie\_type`(revenue)) RETURNS text*

*READS SQL DATA*

*BEGIN*

*DECLARE Movie\_type text;*

*IF revenue > 100000000 THEN*

*SET Movie\_type = “high\_grosser\_movie”;*

*ELSE*

*SET Movie\_type = “low\_grosser\_movie”;*

*END IF;*

*RETURN Movie\_type;*

*END$$*

*DELIMITER ;*

**6. STORED PROCEDURES**

\* stored Procedure (1) : Movie\_specific\_tweets\_made\_on\_selected\_date()

#Description: Consists of Posted\_time as input parameter and returns the list of all tweets made on the

#input date, which is categorized with movie name.

#Joins used: Tables 'consumer\_user\_info' and 'consumer\_tweet\_info' are joined using 'User\_ID'.

*DELIMITER $$*

*CREATE DEFINER=`root`@`localhost` PROCEDURE `movie\_specific\_tweets\_made\_on\_selected\_date`(IN Posted\_time date)*

*READS SQL DATA*

*BEGIN*

*SELECT p.title as 'Movie Name', t.Content*

*FROM consumer\_user\_info t, consumer\_tweet\_info p*

*WHERE p.User\_ID = t.User\_ID*

*AND t.Posted\_Time between (tweetdate) and timestamp (tweetdate, '23:59:59');*

*END$$*

*DELIMITER ;*

# Procedure call for movie\_specific\_tweets\_made\_on\_selected\_date() with '2019-03-02' as input parameter

Call movie\_specific\_tweets\_made\_on\_selected\_date ('2019-03-02');

drop procedure movie\_specific\_tweets\_made\_on\_selected\_date;

#stored Procedure (2) : User\_specific\_posts\_made\_on\_selected\_date()

#Description: Consists of postdate as input parameter and returns the list of all posts made by user

#Joins used: Tables 'consumer\_user\_info' and 'consumer\_tweet\_info' are joined using 'User\_ID'.

*DELIMITER $$*

*CREATE DEFINER=`root`@`localhost` PROCEDURE `user\_specific \_posts\_made\_on\_selected\_date`(IN posted\_time date)*

*READS SQL DATA*

*BEGIN*

*SELECT posted\_time*

*FROM consumer\_user\_info t, consumer\_tweet\_info p*

*WHERE p.User\_ID = i.User\_ID*

*AND post.Created\_Time between (postdate) and timestamp (postdate, '23:59:59');*

*END$$*

*DELIMITER ;*

#stored Procedure (3) : users\_with\_most\_retweeted\_tweets()

#Description: Consists of number\_of\_users as input parameter and returns the list of twitters users(userID, username and user secreen name),

#where number of results is controlled by the input paramater

#Joins used: Tables 'consumer\_user\_info' and 'consumer\_tweet\_info' are joined using 'User\_ID'.

*DELIMITER $$*

*CREATE DEFINER=`root`@`localhost` PROCEDURE `users\_with\_most\_retweeted\_tweets`(IN number\_of\_users INT)*

*READS SQL DATA*

*BEGIN*

*SELECT sum(t.post\_Count) as 'Post Count', u.User\_ID, u.User\_Name,u.User\_Screen\_Name*

*FROM consumer\_user\_info t, consumer\_tweet\_info p*

*where t.User\_ID = u.User\_ID*

*group by u.User\_ID*

*order by sum(t.post) DESC*

*END$$*

*DELIMITER ;*

*DELIMITER $$*

**7. AUDIT VALIDITY:**

The database had some json objects which has been cleaned and the dataset contains no null values and is clean and complete. Few columns have been renamed for the purpose of making simple, understandable and relatable.

**8. AUDIT COMPLETENESS**

The dataset is up to date, needs no more cleaning and matches the quality of the real world data.

**9. AUDIT CONSISTENCE/UNIFORMITY:**

The possible range of the dataset is covered from the new resultant data set. The data does not have any null values, limitations, negative values.

**10. CONCLUSION:**

In the assignment, all of the database tables were populated with real-word data collected from three sources: TMDB API, Twitter API, and kaggle. A conceptual schema was designed for Movies domain involving companies, producers, consumers and subsequently a physical model was also established. The collected data was reformatted, cleaned to fit the database schema and populated into the database. The database was further normalized, views were created for the use cases. Indexes were created to improve the performance of the Queries. User defined functions and procedures are defined.

**11. CITATIONS AND REFERENCES:**

1. Geeks for Geeks - https://www.geeksforgeeks.org

2. Github - https://github.com/nikbearbrown/INFO\_6210

3. Google

4. Kaggle

5. Youtube

6. Stack Overflow