INSY 661 – Group Project

Creating Database for a

Social Media Company

***Contributors:***

**Marion Laniel (260431778)**

**Ziye Zhang (260766101)**

**Xitong Li (260710002)**

**Vaibhav Vishal (260984038)**

**Agathe Chapelle (260735938)**

**Yulin Hong (260898713)**

**Content:**

1. Section 1
   1. Overview
   2. Mission
   3. Entity-relationship diagram (ERD)
   4. Data Dictionary
      1. Entity information
      2. Attribute information
   5. Relational Schema
2. Section 2
   1. Data Definition Language (DDL)
   2. Queries (objectives, assumption, query, and solution)
3. Section 3

Complex Query Identification

* + 1. Logic
    2. Challenges faced
    3. Overall learning

1. **Section 1**
   1. **Overview:**

In the overviews below, we are only tracking the functionalities in the scope of our database, not the whole social media website. But we will show other advanced functionalities of the website in section three such as friend suggestion

USER:

User Information

Graphical user interface, application

Description automatically generated  

Users can have friends and our database should be able to store on which date the two users start a friendship

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

Users can post, can like, comment

A collage of a house

Description automatically generated with low confidence

Graphical user interface, application, Teams

Description automatically generated

  Groups:

Our website should be able to store group information: title, audience, description

Graphical user interface, text, application

Description automatically generated

A user can join a group, and our database should be able to track on what date one user joined one group

Graphical user interface, text, application, Teams

Description automatically generated

Group members can post inside group discussion

Graphical user interface, text, application, chat or text message

Description automatically generated

  Posts:

  A post should contain author (which is a user), date, type, about, audience

A picture containing text, road, screenshot

Description automatically generated

Event:

Events should be able to display their basic information, Such as the title, date, price, audience, description

Graphical user interface, text, application, chat or text message

Description automatically generated

  Users could show their attitudes towards an event:

Icon

Description automatically generated

An event can have many posts in its discussion, but one post should be only related to one event

Graphical user interface, text, application

Description automatically generated

* 1. **Mission**

The purpose of our simulated database based on Facebook is to maintain the data that is used to support a platform where people can make friends, participate in events, join groups, and share posts and to attract advertisers to post their ads to their designated users or groups in the database.

Mission Objectives

To perform searches on users

To perform searches on events

To perform searches on groups

To perform searches on posts

To report on users

To report on events

To report on groups

To report on posts

To maintain (enter, update, and delete) data on users

To maintain (enter, update, and delete) data on events

To maintain (enter, update, and delete) data on groups

To maintain (enter, update, and delete) data on posts

To track the status of likes and comments

To track the status of users joining groups

To track the status of users’ attitudes towards events (interested & going)

To track the status of users’ friendships

* 1. **Entity Relationship Diagram**

Assumptions:

* A user can make friends with one or many users on a certain date.
* Users can post multiple posts, comments, likes, but each post, comment, like should belong to only one user.
* We only store title, audience, description for groups.
* We only store title, date, price, audience, description for events.
* A user can join multiple groups/events, and a group/events can contain more than one user.
* Every user should have an attitude to an event.
* A user can post multiple posts inside a group/event, but each post should belong to only one group/event. A user can also post outside group/event, that is, user can also describe(i.e.post) posts.



describes

* 1. **Data Dictionary**
     1. **Entity information**

|  |  |  |  |
| --- | --- | --- | --- |
| Entity Name | Description | Aliases | Occurrence |
| Users | An entity that represents the person who creates a Facebook Account. | N/A | A user can indicate his/her “City”, relationship, phone number, email, gender, date of birth, name, password, etc. |
| Posts | An entity that links to a user. This entity usually consists of text and multimedia (not required). | N/A | A user can indicate his/her content/ type of posts, the audience who can look at the posts, the date on which it was posted, etc. |
| Events | An entity that lists down events that can be held by an organization or user. | N/A | A user can indicate his/her title of events, the date on which it will be, price, type of audience, and events details/description |
| Groups | An entity that users can join. This entity can host the event and post information. | N/A | A user can indicate his/her group details, its audience, group information, etc. |
| Friend | An entity that users connect with. | N/A | A user can indicate his/her list of friends and the date on which they connected |
| Comment | An entity that contains comments of users. | N/A | A user can indicate his/her comments and their date. |
| Like | An entity that contains likes done by the users. | N/A | A user can indicate his/her date of liking a post |
| User\_Event | An entity that contains events associated with users. | N/A | A user can indicate his/her status of joining the event |
| User\_Group | An entity that users can be part of. | N/A | A user can indicate his/her date of joining the group |

* + 1. **Attribute information**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Entity Name | Attributes | Description | Data Type | Nulls | Multi-valued | Derived | Default |
| Users | userID | User’s unique identifier | INT(10) | No | Yes | No | None |
| name | User first name | VARCHAR(255) | No | No | None |
| surname | User last name | VARCHAR(255) | No | No | None |
| city | User's city | VARCHAR(255) | Yes | No | None |
| relationship\_status | User's relationship status | VARCHAR(255) | Yes | No | Single', 'In a relationship', 'Engaged', 'Married', 'In an open relationship', 'It’s complicated', 'Divorced', 'Widowed' |
| phone\_number | User's phone number | INT(10) | Yes | No | None |
| email | User's email id | VARCHAR(255) | No | No | None |
| gender | User's gender | VARCHAR(255) | Yes | No | None |
| birth\_date | User's Birth date | DATE | No | No | None |
| Posts | postID | Post unique identifier | INT(10) | No | Yes | No | Not Null |
| about | Introduction of post | VARCHAR(255) | Yes | No | None |
| type | Type of post | VARCHAR(255) | No | No | None |
| audience | Audience having access to post | VARCHAR(255) | No | No | 'Public', 'Friends', 'Specific Friends', 'Only me' |
| date | Creation date of post | DATE | No | No | None |
| Events | eventID | Unique event identifier | INT(10) | No | Yes | No | Not Null |
| title | Events title | VARCHAR(255) | No | No | None |
| date | Events date | DATE | No | No | None |
| price | Events ticket/entry price | DECIMAL(10,2) | Yes | No | None |
| audience | Type of audience | VARCHAR(255) | No | No | 'Private', 'Public', 'Friends' |
| description | Events summary | VARCHAR(255) | Yes | No | None |
| Groups | groupID | Unique group identifier | INT(10) | No | Yes | No | Not Null |
| title | Group's name/title | VARCHAR(255) | No | No | None |
| audience | Group members/ audience | VARCHAR(255) | No | No | 'Public', 'Private |
| description | About group | VARCHAR(255) | Yes | No | None |
| Friend | date\_friendship | Date of friendship | DATE | No | No | No | None |
| Comment | about | Content of comment | VARCHAR(255) | Yes | Yes | No | None |
| date\_commented | Date of commenting | DATE | No | No | None |
| Like | date\_liked | Date of liking | DATE | No | No | No | None |
| User\_Event | attitude | going or not or maybe | VARCHAR(255) | No | No | No | ‘Going’, ‘Maybe’, ‘Can’t go’ |
| User\_Group | date\_joined | Date user joined the group | DATE | No | No | No | None |

* 1. **Relational Schema**

EVENT (**EVENTID**, TITLE, DATE, PRICE, AUDIENCE, DESCRIPTION)

GROUP (**GROUPID**, TITLE, AUDIENCE, DESCRIPTION)

USER (**USERID,**NAME, SURNAME, CITY, RELATIONSHIP\_STATUS,

PHONE\_NUMBER, EMAIL, GENDER, BIRTH\_DATE)

POST (**POSTID**, USERID, GROUPID, EVENTID, ABOUT, TYPE, AUDIENCE, DATE)

COMMENT (**POSTID**, **USERID**, ABOUT, DATE\_COMMENTED)

LIKE (**USERID**, **POSTID**, DATE\_LIKED)

FRIEND (**USERID**, **FRIENDID**, DATE\_FRIENDSHIP)

USER\_GROUP (**GROUPID**, **USERID**, DATE\_JOINED)

USER\_EVENT (**EVENTID**, **USERID**, ATTITUDE)

**2. Section 2**

1. **DDL and DML**

Please refer to the SQL script submitted along with this report.

1. **Queries (objectives, assumption, query, and solution)**

Query: 1

Objective: Figure out if relationship (marital status) affects the number of posts posted by people.

Code:

***SELECT***

***relationship\_status AS Status,***

***COUNT(postID) / COUNT(DISTINCT (relationship\_status)) AS 'post per user'***

***FROM***

***(SELECT***

***userID, name, relationship\_status***

***FROM***

***User***

***WHERE***

***relationship\_status IS NOT NULL) u***

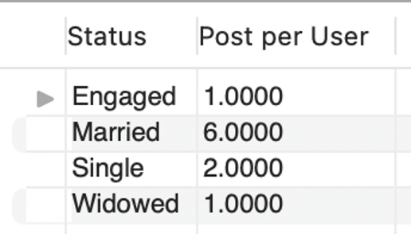
***JOIN***

***Post ON post.userID = u.userID***

***GROUP BY relationship\_status***

***ORDER BY 'post per user' DESC;***

Output Screenshot:



Query: 2

Objective: Figure out if government employees are less likely to comment and like posts due to their job rigidity.

Assumptions: Government people are those whose email ends with .gov

Code:

***SELECT***

***'Government Official',***

***COUNT(postID) / COUNT(DISTINCT (userID)) AS 'Avg num of likes&comments'***

***FROM***

***(SELECT***

***reaction.userID, reaction.postID***

***FROM***

***User***

***JOIN (SELECT***

***userID, postID, date\_liked AS date\_reaction***

***FROM***

***`Like` UNION SELECT***

***userID, postID, date\_commented***

***FROM***

***`Comment`) reaction ON User.userID = reaction.userID***

***WHERE***

***email LIKE '%gov') temp***

***UNION SELECT***

***'Non Government Official',***

***COUNT(postID) / COUNT(DISTINCT (userID)) AS 'Avg num of likes&comments'***

***FROM***

***(SELECT***

***reaction.userID, reaction.postID***

***FROM***

***User***

***JOIN (SELECT***

***userID, postID, date\_liked AS date\_reaction***

***FROM***

***`Like` UNION SELECT***

***userID, postID, date\_commented***

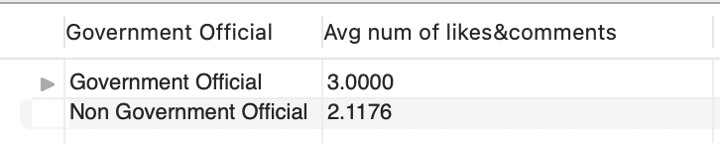
***FROM***

***`Comment`) reaction ON User.userID = reaction.userID***

***WHERE***

***email NOT LIKE '%gov') temp;***

Output Screenshot:



Query: 3

Objective: Find out if the price of an event affects people’s attitude towards it.

Assumptions: Consider ‘Going’ as a positive attitude. The others are negative.

Code:

***SELECT***

***x.level Level,***

***x.total\_attitude 'Total Attitudes',***

***y.num\_Going 'Number of Going',***

***CONCAT(FORMAT(y.num\_Going / x.total\_attitude \* 100,***

***2),***

***'%') AS '% of Going'***

***FROM***

***(SELECT***

***level, COUNT(\*) AS total\_attitude***

***FROM***

***(SELECT***

***`Event`.eventId,***

***User\_event.userID,***

***User\_event.attitude,***

***CASE***

***WHEN price > 100000 THEN 'Extremely High'***

***WHEN price > 200 THEN 'High'***

***WHEN price > 0 THEN 'Low'***

***ELSE 'Free'***

***END `level`***

***FROM***

***`Event`***

***JOIN User\_Event ON `Event`.eventID = User\_event.eventID) temp1***

***GROUP BY level) x***

***JOIN***

***(SELECT***

***level, COUNT(attitude) AS num\_Going***

***FROM***

***(SELECT***

***`Event`.eventId,***

***User\_event.userID,***

***User\_event.attitude,***

***CASE***

***WHEN price > 100000 THEN 'Extremely High'***

***WHEN price > 200 THEN 'High'***

***WHEN price > 0 THEN 'Low'***

***ELSE 'Free'***

***END `level`***

***FROM***

***`Event`***

***JOIN User\_Event ON `Event`.eventID = User\_event.eventID) temp2***

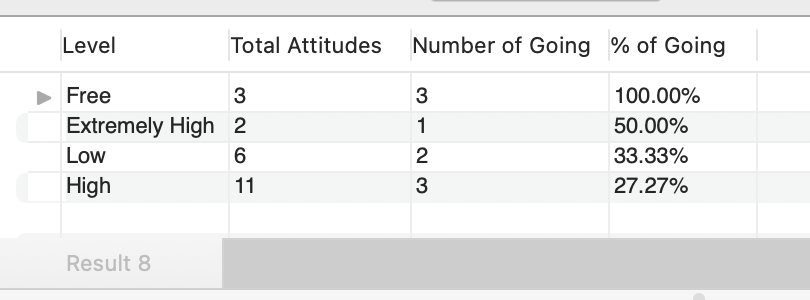
***WHERE***

***attitude LIKE 'Going'***

***GROUP BY level) y ON x.level = y.level***

***ORDER BY y.num\_Going / x.total\_attitude DESC;***

Output Screenshot:



Query: 4

Objective: Find the most active person on the list

Assumptions: The most active person is the one who has the maximum number of events attended, groups joined, and posts posted. That is, Activities = # of groups+# of events+# of posts.

Code:

***SELECT***

***name Name, Activities***

***FROM***

***(SELECT***

***u.userID,***

***u.name,***

***a.num\_of\_group,***

***b.num\_of\_event,***

***c.num\_of\_post,***

***a.num\_of\_group + b.num\_of\_event + c.num\_of\_post AS Activities***

***FROM***

***User u***

***LEFT JOIN (SELECT***

***userID, COUNT(groupID) num\_of\_group***

***FROM***

***User\_Group***

***GROUP BY userID) a ON u.userID = a.userID***

***LEFT JOIN (SELECT***

***userID, COUNT(EventID) num\_of\_event***

***FROM***

***User\_Event***

***GROUP BY userID) b ON u.userID = b.userID***

***LEFT JOIN (SELECT***

***userID, COUNT(postID) num\_of\_post***

***FROM***

***Post***

***WHERE***

***userID IS NOT NULL***

***GROUP BY userID) c ON u.userID = c.userID) d***

***ORDER BY Activities DESC***

***LIMIT 1;***

Output Screenshot:



Query: 5

Objective: Select people who attend expensive events so that Facebook can target those users and places luxury products’ advertisements on their homepages

Assumptions: Assume users with the attitude ‘going’ are attending these events.

Code:

***SELECT***

***User\_Event.userID,***

***price***

***from Event***

***join***

***User\_Event***

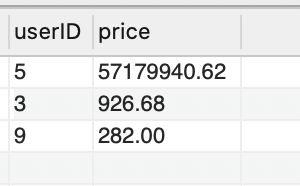
***on Event.eventID = User\_Event.eventID***

***where attitude = 'going'***

***order by price desc***

***limit 3;***

Output Screenshot:



Query: 6

Objective: Suggest friends for the user named Goldie. The suggestion is based on the events Goldie attended. People who attended the same events as Goldie but are not friend with Goldie will be recommended.

Code:

***select***

***concat(User. name,' ', User. surname) as 'suggested friends for attending the same event with Goldie'***

***from***

***User\_Event***

***join***

***User***

***on User\_Event.userID = User.userID***

***where***

***User\_Event.eventID in (select***

***eventID***

***from User\_Event***

***where userID in (select userID***

***from User***

***where name = 'Goldie'))***

***and***

***User\_Event.userID not in (select***

***friendID***

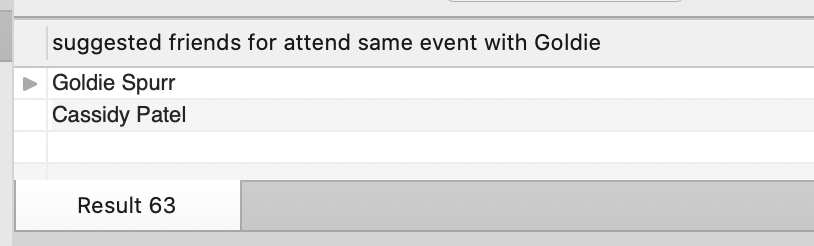
***from friend***

***where userID in (select userID***

***from User***

***where name = 'Goldie'));***

Output Screenshot:



Query: 7

Objective:

Suggest friends for the user named Jacquenetta based on her friends. Her friends’ friends will be recommended if they are not friends with Jacquenetta.

Code:

***select***

***concat(User.name,' ',User.surname) as name,***

***User.city,***

***friends.f\_name as is\_friend\_with***

***from User***

***join(***

***select***

***distinct friendID as ID,***

***friend.userID as f***

***From friend***

***where***

***friend.userID***

***in (***

***select***

***friendID as userID***

***from friend***

***where userID in***

***(select userID***

***from User***

***where name = 'Jacquenetta'))***

***) as suggest\_friends***

***on User.userID = suggest\_friends.ID***

***join***

***(select***

***concat(User.name,' ',User.surname) as f\_name,***

***User.userID as ID***

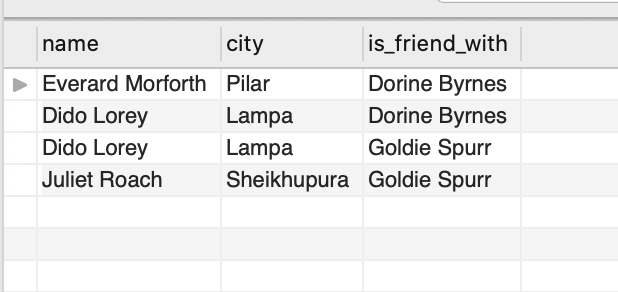
***from***

***user) as friends***

***on suggest\_friends.f = friends.ID***

***;***

Output Screenshot:



Query: 8

Objective:

Suggest groups user Jacquenetta might be interested in, based on her friends’ interests. Order by the number of Jacquenetta’s friends in that group.

Code:

***select***

***User\_Group.groupID,***

***`Group`.title,***

***count(User\_Group.userID) as 'number of friends in this group'***

***from User\_Group***

***join `Group`***

***on User\_Group.groupID = `Group`.groupID***

***where User\_Group.userID in(***

***select***

***friendID***

***From Friend***

***where Friend.userID in***

***(select userID***

***from User***

***where name = 'Jacquenetta'))***

***AND***

***User\_Group.groupID not in(***

***select***

***User\_Group.groupID***

***From User\_Group***

***where User\_Group.userID in***

***(select userID***

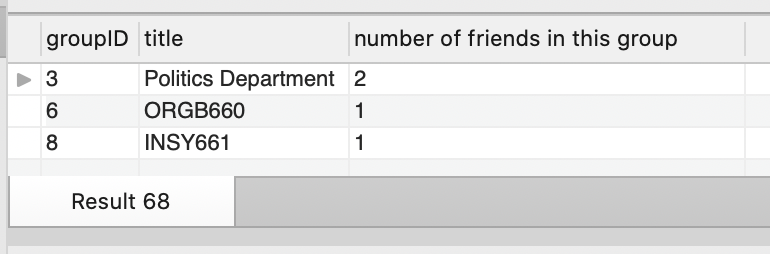
***from User***

***where name = 'Jacquenetta'))***

***group by groupID***

***order by count(User\_Group.userID) desc;***

Output Screenshot:



Query 9

Objective: Select the event and show its participants, if anyone is going to it. By doing so, we can analyze those events that tend to attract customers

Code:

***select event.eventID, event.title, user.name, user.surname***

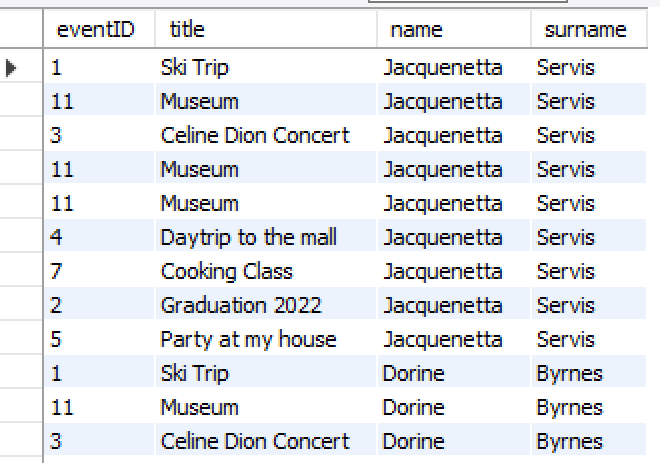
***from user\_event***

***inner join event on user\_event.eventID = event.eventID***

***inner join user on user\_event.userID = user\_event.userID***

***where user\_event.attitude = 'Going';***

Output Screenshot:



Query: 10

Objective: Suggest groups Dido might be interested in, we knew that Dido has joined the group “McGill”, we could suggest all the other groups contain the keyword “McGill” based on this.

Code:

***select \* from `group`***

***where title like (***

***concat("%", (select title from (***

***select Dido.name, `group`.title from user\_group***

***inner join(***

***select \* from user***

***where name = 'Dido'***

***)as Dido***

***on user\_group.userID = Dido.userID***

***inner join `group` on `group`.groupID = user\_group.groupID***

***)as a), "%")***

***) AND TITLE <> (***

***select title from (***

***select Dido.name, `group`.title from user\_group***

***inner join(***

***select \* from user***

***where name = 'Dido'***

***)as Dido***

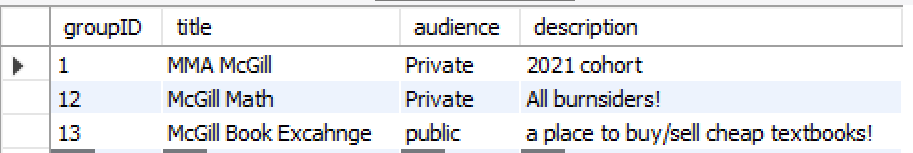
***on user\_group.userID = Dido.userID***

***inner join `group` on `group`.groupID = user\_group.groupID***

***)as a***

***);***

Output Screenshot:



Query: 11

Objective: Find the author of the post which is commented by most users, display the post\_ID and the number of comments it received as well.

Code:

***select post.postID as mostPopularPost, user.userID as mostPopularAuthor, c.NumberOfComments from post***

***inner join (***

***select \* from (***

***select count(distinct(userID)) as NumberOfComments, postID***

***from `Comment`***

***group by postID***

***)as a***

***where a.NumberOfComments = (***

***select max(NumberOfComments) from (***

***select count(userID) as NumberOfComments, postID***

***from `Comment`***

***group by postID***

***)as b***

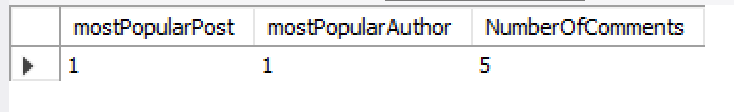
***)***

***)as c***

***on post.postID = c.postID***

***inner join user on post.userID = user.userID***

Output Screenshot:



Query: 12

Objective: count the number of potential attendees to an upcoming event

Assumption: Potential attendees include “maybe” and “not going”

Code:

***Select Event.eventID, Event.title, A.NumberOfAttendees***

***From Event***

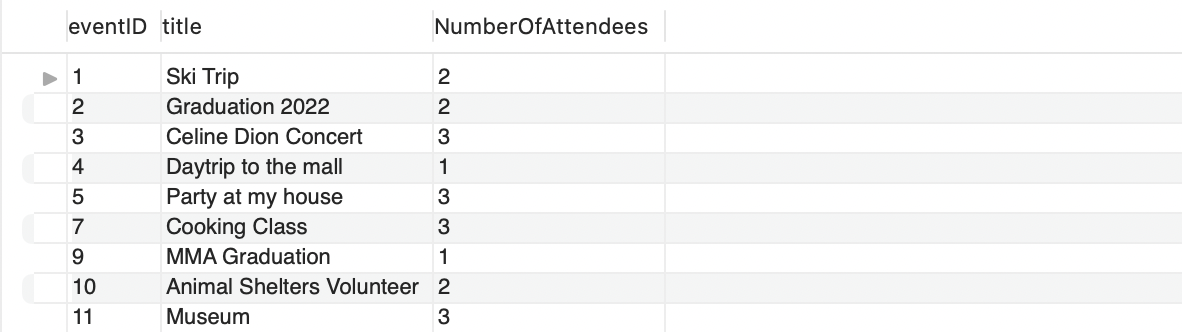
***Join (Select count(User\_Event.userID) NumberOfAttendees, User\_Event.eventID***

***From User\_Event***

***Where User\_Event.attitude != "Can't go"***

***Group by eventID) as A on Event.eventID=A.eventID***

Output:



Query: 13

Objective: Determine the average age of a Facebook user

Code:

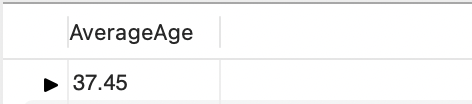
***Select Avg(B.Age) as AverageAge***

***From (Select ("2021"- A.BirthYear) as Age***

***From (Select extract(Year from birth\_date) as BirthYear***

***From `User`) as A) as B***

Output:



Query: 14

Objective: Find the users who have more male friends than females.

Code:

***SELECT***

***user.name, b.userID, b.Male\_friend, c.Female\_friend***

***FROM***

***(SELECT***

***userID, COUNT(gender) Male\_friend***

***FROM***

***(SELECT***

***friend.userID, friend.friendID, user.gender***

***FROM***

***Friend***

***JOIN user ON user.userID = Friend.friendID) a***

***WHERE***

***gender = 'Male'***

***GROUP BY userID) b***

***LEFT JOIN***

***(SELECT***

***userID, COUNT(gender) Female\_friend***

***FROM***

***(SELECT***

***friend.userID, friend.friendID, user.gender***

***FROM***

***Friend***

***JOIN user ON user.userID = Friend.friendID) a***

***WHERE***

***gender = 'Female'***

***GROUP BY userID) c ON b.userID = c.userID***

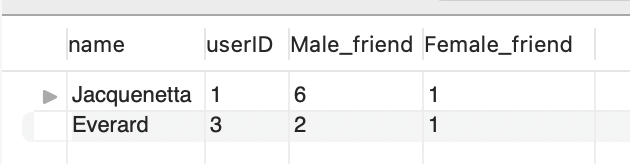
***JOIN***

***user ON b.userID = user.userID***

***WHERE***

***b.Male\_friend > c.Female\_friend;***

Output Screenshot:



Query 15

Objective: Suggest a friend for the user named Jacquenetta, based on strangers who commented on Jacquenetta’s post.

Assumption: a person who is not friends with Jacquenetta is considered to be a stranger.

Code:

***select***

***concat(User.name,' ',User.surname) as 'suggested friend',***

***Comment.date\_commented***

***from***

***Comment***

***join User***

***on Comment.userID = User.userID***

***where***

***Comment.postID in***

***(select***

***Post.postID***

***from***

***Post***

***where Post.userID in***

***(select User.userID***

***from User***

***where name = 'Jacquenetta'))***

***AND***

***Comment.userID NOT IN***

***(select***

***Friend.friendID***

***from***

***Friend***

***where Friend.userID in***

***(select User.userID***

***from User***

***where name = 'Jacquenetta'))***

***AND***

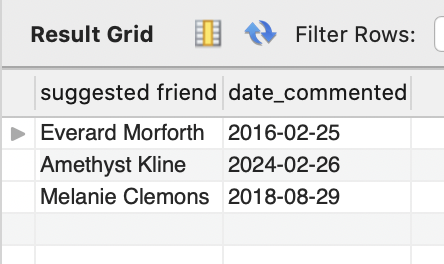
***Comment.userID NOT IN***

***(select User.userID***

***from User***

***where name = 'Jacquenetta')***

Output Screenshot:



Query 16

Objective: identifying the upcoming event in the next month

Code:

***select eventID***

***from `Event`***

***where extract(month from `date`) = (select (extract(month from sysdate())) + 1***  ***as next\_month)***

***and extract(year from `date`) = (select (extract(year from sysdate())))***

Output:



Query 17

Objective: identifying users who are engaging the most with posts by evaluating their number of likes attributed to different posts.

Code:

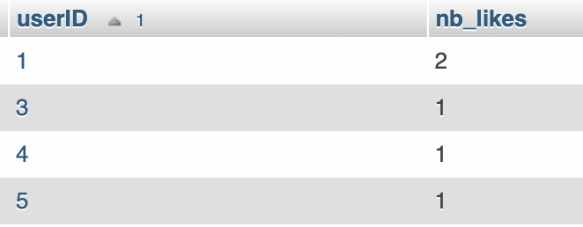
***select userID, count(distinct postID) as nb\_likes***

***from `Like`***

***group by userID***

***order by userID***

Output:



Query 18

Objective: determining the number of friends per user.

Code:

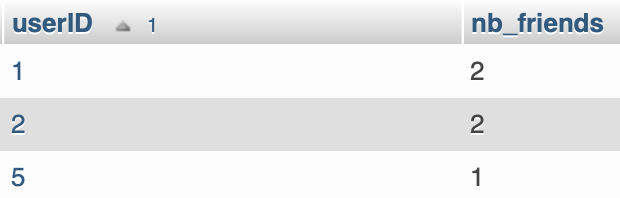
***select userID, count(distinct friendID) as nb\_friends***

***from Friend***

***group by userID***

***order by userID***

Output:



Query 19

Objective: determining the proportion of posts that contain a photo attachment.

Code:

***select t.nb\_photos/s.nb\_posts as proportion\_posts\_photos***

***from(***

***select count(distinct postID) as nb\_posts***

***from Post) as s,***

***(***

***select count(distinct postID) as nb\_photos***

***from Post***

***where type like 'Photo') as t***

Output:



Query 20

Analyze the influence of marital status on people's consumption perceptions through our social media platform

Code:

***select Count(Marital\_Status)/(***

***select count(\*) as total\_people\_going from (***

***select Marital\_status, userID, eventID,***

***CASE***

***WHEN price > 900 THEN 'High'***

***ELSE 'Low'***

***END `Price\_level` ,***

***CASE***

***when attitude = "Can't go" then "not interested"***

***else "interested"***

***end "attitude\_level"***

***from (***

***select user\_event.userID, user\_event.eventID, u.Marital\_Status, user\_event.attitude, event.price***

***from user\_event***

***inner join***

***(***

***SELECT***

***userID,***

***gender,***

***CASE***

***WHEN relationship\_status = 'Married' THEN 'Married'***

***ELSE 'Unmarried'***

***END `Marital\_Status`***

***FROM user***

***)as u***

***on u.userID = user\_event.userID***

***inner join event on user\_event.eventID = event.eventID***

***)as a***

***)as b***

***where attitude\_level <> 'not interested'***

***) as percent\_married\_people\_going\_to\_expensive\_event***

***from (***

***select \* from (***

***select Marital\_status, userID, eventID,***

***CASE***

***WHEN price > 900 THEN 'High'***

***ELSE 'Low'***

***END `Price\_level` ,***

***CASE***

***when attitude = "Can't go" then "not interested"***

***else "interested"***

***end "attitude\_level"***

***from (***

***select user\_event.userID, user\_event.eventID, u.Marital\_Status, user\_event.attitude, event.price***

***from user\_event***

***inner join***

***(***

***SELECT***

***userID,***

***gender,***

***CASE***

***WHEN relationship\_status = 'Married' THEN 'Married'***

***ELSE 'Unmarried'***

***END `Marital\_Status`***

***FROM user***

***)as u***

***on u.userID = user\_event.userID***

***inner join event on user\_event.eventID = event.eventID***

***)as a***

***)as b***

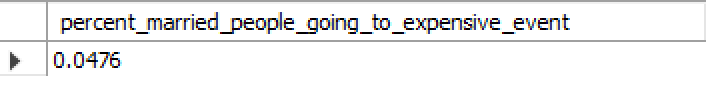
***where attitude\_level <> 'not interested'***

***)as c***

***where Marital\_Status = 'Married'***

***and price\_level = 'High';***

Output:



**3. Section 3**

**Complex / Interesting Query Identification**

**Two interesting queries: Query 3 and Query 10**

Query 3

* + 1. Idea: When analyzing users' behaviors towards events, we think the price of an event is one of the factors that affect people's attitude (i.e., one of the options among "Going", "Maybe", "Can't go"). Therefore, it would be useful to query the relationship between price levels and percentage of Going, where %Going =#\_of\_Going /(#\_of\_Going+#\_of\_Maybe+#\_of\_Can'tGo)
    2. Logic: Since it is harder for SQL language to analyze continuous price data than using other programming languages where we can run linear regressions at ease such as Python and R, we divide the continuous price data into four levels using Case functions, which are "Extremely High", "High", "Low", and "Free". We first query and count the total number of attitudes towards each event, then count the number of going towards each event, finally we inner join these two tables based on the 4 price levels and derive the percentage of going in descending order.
    3. Challenges faced
       1. People who do not go may fully ignore the event posts and do not express their attitudes on the platform. This may decrease the number of “Can’t go” and thus the % of going may falsely seem higher
       2. As mentioned in logic, linear regression might be more useful for the continuous data unless there are strong patterns for clusters
       3. Data is not enough, especially for 'extremely high'. This may cause the % of Going to seem high in the result
       4. The two counts cannot be done using a single table because the latter needs additional conditions.
    4. Overall learning:
       1. The MySQL built-in function Case function can be helpful for grouping continuous data into clusters when the cutoff values are determined
       2. MySQL is a DBMS and SQL language is a query language. People should not be constrained to SQL when they work in data analysis. Data can always be loaded to other platforms for further exploration

Query 10

i. Idea: When browsing the Facebook site, we found that Facebook uses text-matching to suggest groups that users might like, so we wanted to implement this feature on our social platform as well.

ii. Logic: We first find the groups that the target user has joined, then use MYSQL's LIKE statement and regular expression to find other groups with similar names, and don't forget to exclude the groups that the target user has joined in the final recommendation.

iii. Challenges: While splitting a string in python is easy, it's very difficult to do in MySQL, so we can only suggest groups for users that contain the name of their existing group, and it's very difficult to cut out a part of the existing group name for recommendation

iv. Learnings: When using subquery inside the MYSQL like statement, we must remember to add "%" to the concat statement, otherwise the subquery return will not be used properly, and we will not get the desired output. (for example, LIKE (CONCAT(“%”, (….),”%”)))

**One complex query: Query 20**

Query 20

i. Idea: People are likely to become frugal when they get married because most of them have to think about raising children, so we wanted to examine this trend through our social platforms.

ii. Logic: we first join user\_event with user and event table, from these we only need marital status, userID, eventID, attitude, price. Then we select out those records with high price, and count how many of them are married, finally use the previous number we got to divide by the whole number.

iii. Challenges: When group by, not only marital status needs to be taken into account, but also price, that is, the impact of both on people, which is somewhat tricky here. Because only one column can be grouped by when group by.

* + 1. Learnings: When faced with the tricky problem of having two or even more columns that need to be grouped by, we can instead consider representing the result in probabilistic form, so that we can reduce the number of columns that need to be grouped by. And it does not affect our analysis, because after deriving one probability, we can subtract it from 1 to get another probability, and this comparison also gives us a sense of who has more influence on the result.

**Appendix #1:** Draft ERD – first attempt at developing the ERD. We modified this ERD in order to accurately capture the many to many relationships.

