

Домашнее задание:

Работаем с БД vk и тестовыми данными, которые вы сгенерировали ранее:

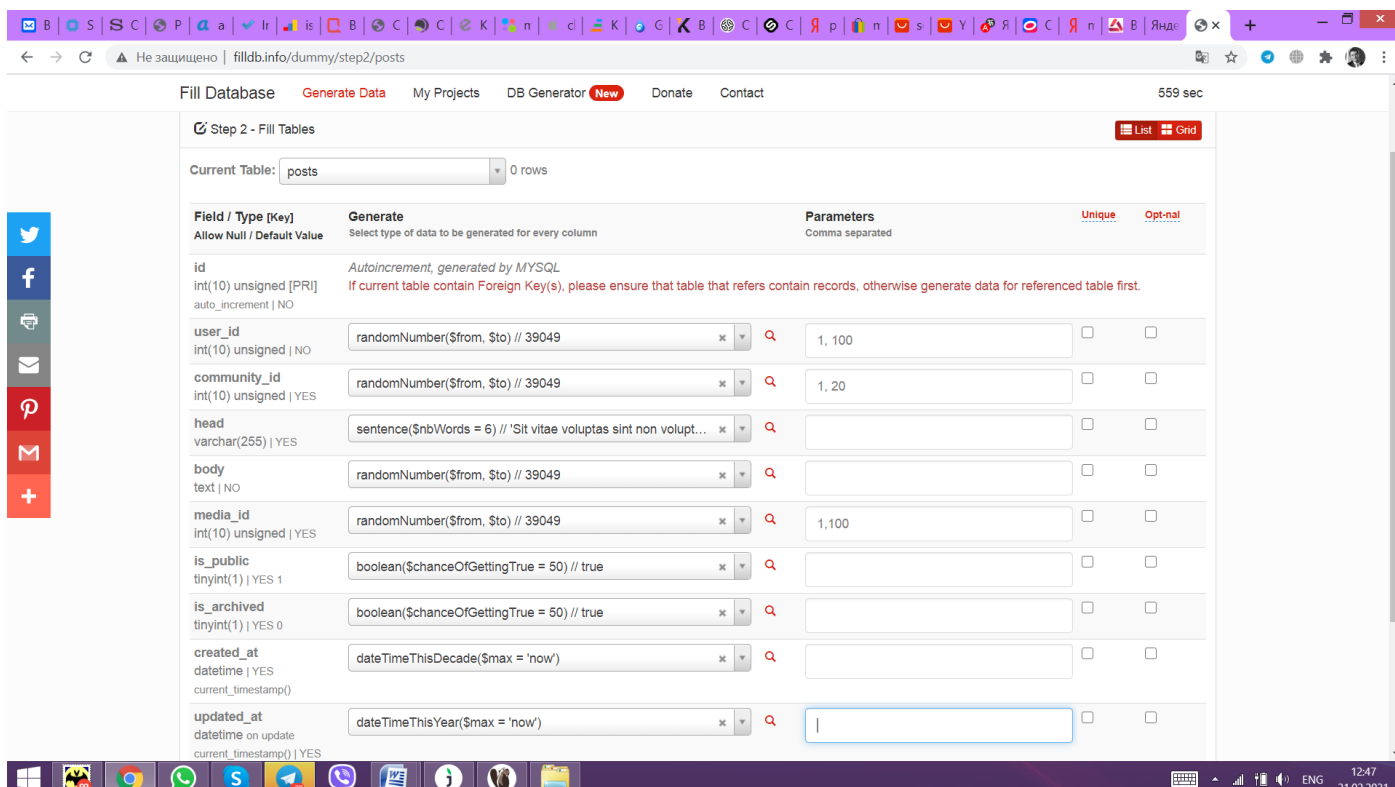
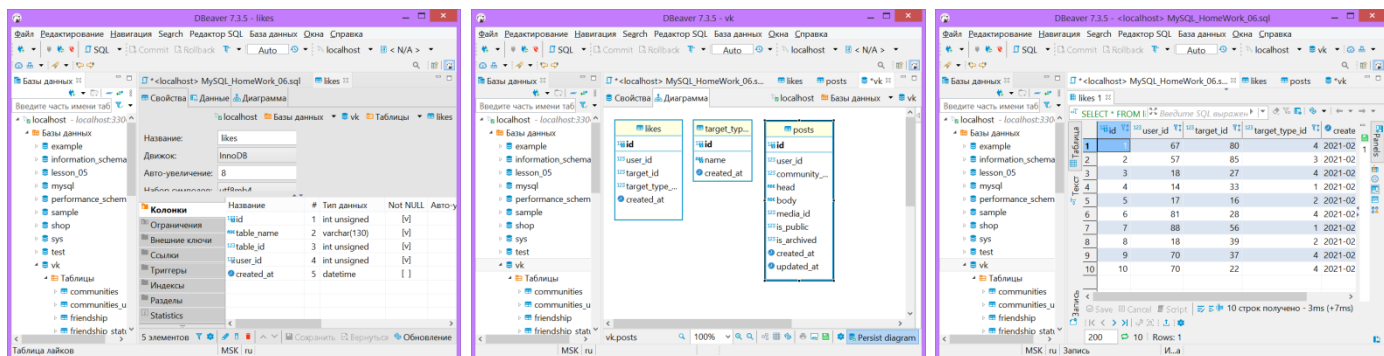
1. Создать и заполнить таблицы лайков и постов.
2. Создать все необходимые внешние ключи и диаграмму отношений.
3. Определить кто больше поставил лайков (всего) - мужчины или женщины?
4. Подсчитать общее количество лайков десяти самым молодым пользователям (сколько лайков получили 10 самых молодых пользователей).
5. Найти 10 пользователей, которые проявляют наименьшую активность в использовании социальной сети (критерии активности необходимо определить самостоятельно).

1. Создать и заполнить таблицы лайков и постов.

В соответствии с рекомендацией преподавателя, создадим и заполним таблицы Лайков «likes», типов лайков «target_type» и Постов «posts» (см. [MySQL_HomeWork_06.sql](#)).

Заполним данными и проверим их наличие...

Таблицу «posts» заполним через <http://filldb.info/> и загрузим дамп (см. [fulldb21-02-2021 09-48.sql](#))

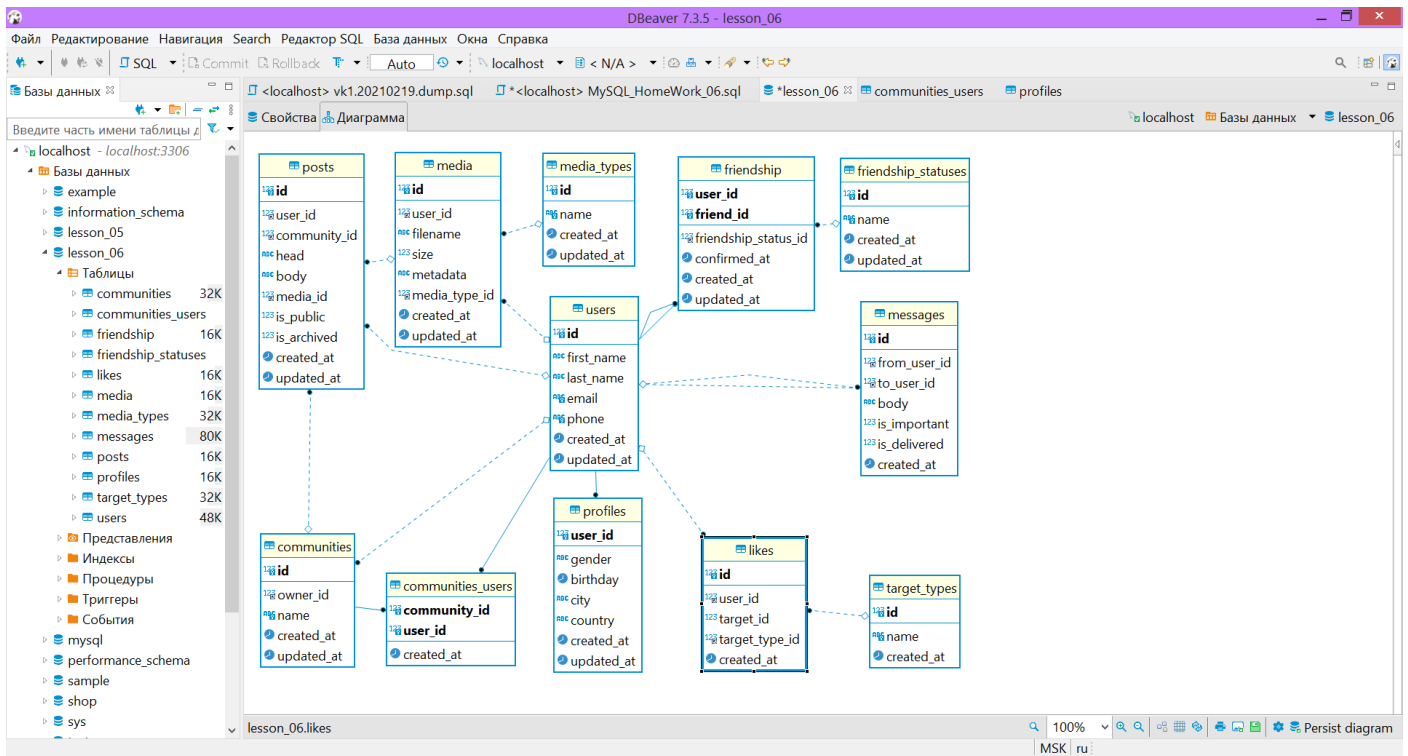


Перед выполнением задания, создадим свежую БД «*CREATE DATABASE lesson_06;*» и прогрузим описание таблиц с данными из дампа через консоль «*mysql lesson_06 < vk1.20210219.dump.sql*». Кстати, по мнению DBBeaver, в дампе была ошибка в строке заполнения данными таблицы ... «*media*» - обратите внимание на апостроф в строке "*Elva O'Reilly*" Загрузка в консоли прошла не с первого раза – ругалась на несоответствие типов ID в users (BIGINT) и from_user_id в messages (INT)... «*ERROR 3780 (HY000) at line 218: Referencing column 'from_user_id' and referenced column 'id' in foreign key constraint 'messages_from_user_id_fk' are incompatible*»

[illegible]

«SELECT * FROM information_schema.KEY_COLUMN_USAGE WHERE TABLE_SCHEMA='lesson_06' AND CONSTRAINT_NAME <>'PRIMARY' AND REFERENCED_TABLE_NAME is not null;» (запрос выдал 12 ключей)

Посмотрим диаграмму связей:



3. Определить кто больше поставил лайков (всего) - мужчины или женщины?

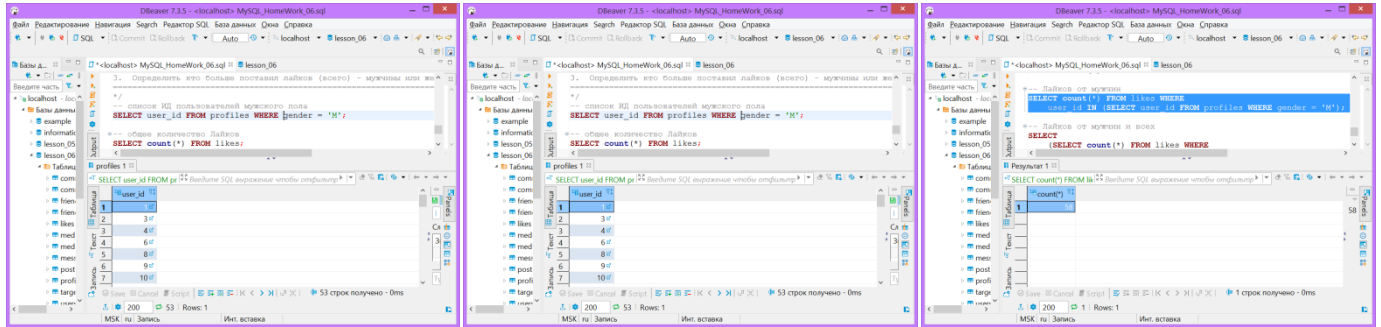
Согласно условиям выполнения ДЗ, решить необходимо путем использования подзапросов. Для начала, получим список ID пользователей мужского пола «*SELECT user_id FROM profiles WHERE gender = 'M';*».

Далее, выясним общее количество Лайков «*SELECT count(*) FROM likes;*»

Объединив два запроса, получим количество Лайков поставленных мужчинами

SELECT count() FROM likes WHERE*

user_id IN (SELECT user_id FROM profiles WHERE gender = 'M');



Объединим запросы и в рамках одного обращения выведем количество «мужских» лайков и общее их число *SELECT*

(SELECT count() FROM likes WHERE*

user_id IN (SELECT user_id FROM profiles WHERE gender = 'M')) AS men_,

(SELECT count() FROM likes) AS all_;*

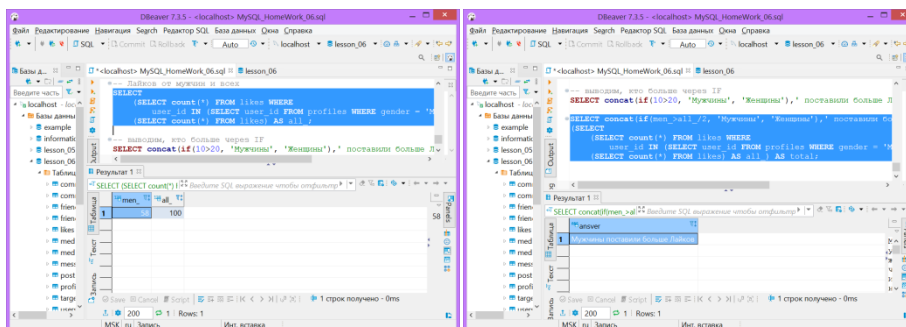
В принципе, уже понятно, что 58 больше половины от 100, но мы хотим «человеческий ответ», поэтому добавим конструкцию IF

SELECT concat(if(men_>all_/2, 'Мужчины', 'Женщины'),' поставили больше Лайков') AS answer FROM (SELECT

(SELECT count() FROM likes WHERE*

user_id IN (SELECT user_id FROM profiles WHERE gender = 'M')) AS men_,

(SELECT count() FROM likes) AS all_) AS total;*



4. Подсчитать общее количество лайков десяти самым молодым пользователям (сколько лайков получили 10 самых молодых пользователей).

Решаем «в лоб». Для начала, список 10-ти ИД самых молодых пользователей

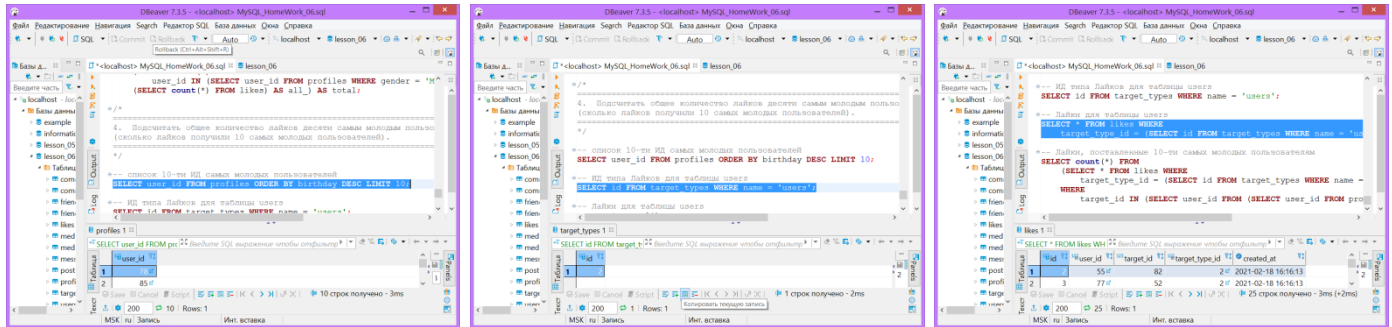
«**SELECT user_id FROM profiles ORDER BY birthday DESC LIMIT 10;**»

Далее, выясним ИД типа Лайков для таблицы users «**SELECT id FROM target_types WHERE name = 'users';**» (2)

И, посмотрим на строки из таблицы Лайков для таблицы users (их всего 25шт)

SELECT * FROM likes WHERE

target_type_id = (SELECT id FROM target_types WHERE name = 'users');



Теперь, объединив ранее проверенные подзапросы, посчитаем Лайки, поставленные 10-ти самым молодым пользователям.

SELECT count(*) FROM

(SELECT * FROM likes WHERE

target_type_id = (SELECT id FROM target_types WHERE name = 'users')) AS ll

WHERE

target_id IN (SELECT user_id FROM profiles ORDER BY birthday DESC LIMIT 10);

Но, запрос выдает ошибку «SQL Error [1235] [42000]: This version of MySQL doesn't yet support 'LIMIT & IN/ALL/ANY/SOME subquery'» Это ошибка оптимизатора запросов, который пытается индексировать столбцы.

Решается обертыванием подзапроса в запрос (ответ – 7):

SELECT count(*) FROM

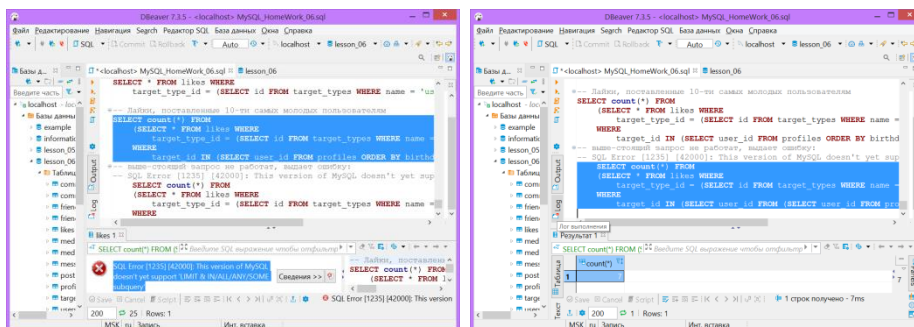
(SELECT * FROM likes WHERE

target_type_id = (SELECT id FROM target_types WHERE name = 'users')) AS ll

WHERE

target_id IN (SELECT user_id FROM

(SELECT user_id FROM profiles ORDER BY birthday DESC LIMIT 10) ttt);



5. Найти 10 пользователей, которые проявляют наименьшую активность в использовании социальной сети (критерии активности необходимо определить самостоятельно)

Определим критерии активности добавив новую сущность «ratings». В ней определим весовые коэффициенты для каждого вида активности. Можно указывать любую таблицу, в которой указываем поле с ID пользователя и вес (коэффициент значимости активности в данной сущности). Так, к примеру, таблице «friendship» мы поставили два разных веса оценки: «1» - для инициатора дружбы и «0.5» - для «получателя» приглашения дружить.

```
CREATE TABLE ratings (  
    id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY COMMENT "Идентификатор строки",  
    table_name VARCHAR(130) NOT NULL COMMENT "Имя таблицы, для которой имеется рейтинг",  
    table_col_name VARCHAR(130) NOT NULL COMMENT "Имя столбца для users(id) в таблице, для которой имеется рейтинг",  
    ratio FLOAT UNSIGNED NOT NULL COMMENT "Весовой коэффициент (множитель) при расчете рейтинга",  
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP COMMENT "Дата создания"  
) COMMENT "Таблица рейтингов";
```

Разработал хранимую процедуру вычисления рейтинга пользователя на основе построения динамического SQL-запроса. Процедура просматривает все оцениваемые сущности из таблицы ratings (posts, media, friendship,...). В общем, любые таблицы по любому полю связи с user_id и произвольному весовому коэффициенту (разные активности оцениваются индивидуально). На входе – «user_id», на выходе итоговый рейтинг пользователя с учетом весовых коэффициентов. Единственная проблема, что логичнее было сделать это функцией, но есть ограничение «...mysql Dynamic SQL is not allowed in stored function...». Пришлось воспользоваться параметром с признаком OUT

«CREATE PROCEDURE getRating (IN user_id INT, OUT user_rating_out FLOAT)»

Для получения итогового ответа, создал еще одну процедуру заполнения временной таблицы с рейтингами всех пользователей. Процедура последовательно перебирает всех пользователей и рассчитывает по каждому его рейтинг путем запуска процедуры «getRating». Результат записывается во временную таблицу «rating_tmp». Итогом является запрос «SELECT u.first_name, u.last_name, r.rating, id FROM users u, rating_tmp r WHERE u.id = r.user_id ORDER BY r.rating LIMIT 10;»

The first screenshot shows the 'ratings' table structure and data. The table has columns: id, table_name, table_col_name, ratio, and created_at. The data includes rows for 'posts', 'media', 'friendship', 'likes', and 'communities_users'.

The second screenshot shows the 'getRating' stored procedure code. The procedure takes 'user_id' as an input and 'user_rating_out' as an output. It iterates through the 'ratings' table, calculates the rating for each table, and returns the total rating.

The third screenshot shows the 'rating_tmp' table results. The table has columns: first_name, last_name, rating, and id. The results show the top 10 users with their ratings, ordered by rating in descending order.

После замечания преподавателя «...очень хорошо, хотя пока задания были на применение вложенных запросов...», дополнил решение альтернативным вариантом (без использования хранимых процедур). Для начала, собрал рейтинг по всем пользователям по одной таблице posts с учетом весового коэффициента

```
SELECT id, u.first_name, u.last_name,
       (SELECT count(*) FROM posts WHERE user_id = u.id) *
       (SELECT ratio FROM ratings WHERE table_name = 'posts') AS rating
FROM users u;
```

Далее, внесем в запрос все таблицы, влияющие на рейтинг

```
SELECT u.id, u.first_name, u.last_name,
       (SELECT count(*) FROM posts WHERE user_id = u.id) *
       (SELECT ratio FROM ratings WHERE table_name = 'posts') +
       (SELECT count(*) FROM media WHERE user_id = u.id) *
       (SELECT ratio FROM ratings WHERE table_name = 'media') +
       (SELECT count(*) FROM friendship WHERE user_id = u.id) *
       (SELECT ratio FROM ratings WHERE table_name = 'friendship' AND table_col_name = 'user_id') +
       (SELECT count(*) FROM friendship WHERE friend_id = u.id) *
       (SELECT ratio FROM ratings WHERE table_name = 'friendship' AND table_col_name = 'friend_id') +
       (SELECT count(*) FROM communities WHERE owner_id = u.id) *
       (SELECT ratio FROM ratings WHERE table_name = 'communities') +
       (SELECT count(*) FROM communities_users WHERE user_id = u.id) *
       (SELECT ratio FROM ratings WHERE table_name = 'communities_users') +
       (SELECT count(*) FROM likes WHERE user_id = u.id) *
       (SELECT ratio FROM ratings WHERE table_name = 'likes')
       AS rating
FROM users u ORDER BY rating, first_name, last_name LIMIT 10;
```

Результат абсолютно идентичен ранее предложенной реализации через хранимые процедуры. Кстати, времени на эту реализацию потрачено в разы меньше. Конечно, есть и минус у такого варианта. Для расширения перечня таблиц и/или столбцов в варианте с процедурами было достаточно добавить соответствующую запись в таблицу «*ratings*». В случае с запросом, придется изменить и сам запрос.

The first screenshot shows a query for the 'posts' table. The query is:
`SELECT id, u.first_name, u.last_name, (SELECT count(*) FROM posts WHERE user_id = u.id) * (SELECT ratio FROM ratings WHERE table_name = 'posts') AS rating FROM users u;`
 The result shows 11 rows of user data with their ratings for the 'posts' table.

The second screenshot shows a query for the 'likes' table. The query is:
`SELECT id, u.first_name, u.last_name, (SELECT count(*) FROM likes WHERE user_id = u.id) * (SELECT ratio FROM ratings WHERE table_name = 'likes') AS rating FROM users u;`
 The result shows 11 rows of user data with their ratings for the 'likes' table.

The third screenshot shows a query for the 'ratings' table. The query is:
`CALL fillRating;`
 The result shows 11 rows of user data with their ratings for the 'ratings' table.