Курс: Основы реляционных баз данных. MySQL
Урок 10. Транзакции, переменные, представления. Администрирование. Хранимые процедуры и функции, триггеры
Выполнил: Кузнецов Сергей (Факультет Geek University Python-разработки)

Домашнее задание:

1. Проанализировать какие запросы могут выполняться наиболее часто в процессе работы приложения и добавить необходимые индексы.
2. Задание на оконные функции. Построить запрос, который будет выводить следующие столбцы:
   - имя группы;
   - среднее количество пользователей в группах;
   - самый молодой пользователь в группе;
   - самый старший пользователь в группе;
   - общее количество пользователей в группе;
   - всего пользователей в системе;
   - отношение в процентах (общее количество пользователей в группе / всего пользователей в системе) * 100.

---

1. Проанализировать какие запросы могут выполняться наиболее часто в процессе работы приложения и добавить необходимые индексы.

При принятии решения о целесообразности добавления индекса, в условиях невозможности применения классического метода учитывающего статистику частоты запросов к БД и накладные расходы по временным задержкам психологически «мешающим» usability, обратим внимание на субъективно часто-использующийся функционал. При этом, не будем строить индексы, дублирующие уже имеющиеся (все первичные, вторичные ключи и поля с признаком UNIQUE).

Проанализировав интерфейс «Vk» и отталкиваясь от имеющегося описания БД, приходит на ум целесообразность в обеспечении следующих таблиц индексами:

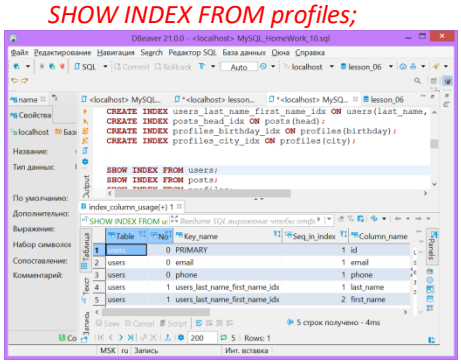
- users (last_name, first_name) – поиск человека
- users (email) – поиск на уникальность, но такой индекс уже есть, как у всех полей с признаком UNIQUE
- posts (head) - просмотр/поиск постов по заголовкам
- media – не нуждается (по внешним ключам и так есть индексы, а в остальном - слабо структурированная информация)
- media_types – не нуждается (минимум записей)
- friendship – не нуждается (внешние ключи и так закрывают все потребности по индексам)
- friendship_statuses - не нуждается (минимум записей)
- communities (name) - поиск на уникальность, но такой индекс уже есть, как у всех полей с признаком UNIQUE
- communities_users - не нуждается (внешние ключи…)
- likes – не нуждается (таблица связей)
- target_types - не нуждается (минимум записей)
- ratings - не нуждается (минимум записей)
- messages – не нуждается
- profiles (birthday) – частая опция поиска
- profiles (city) – частая опция поиска

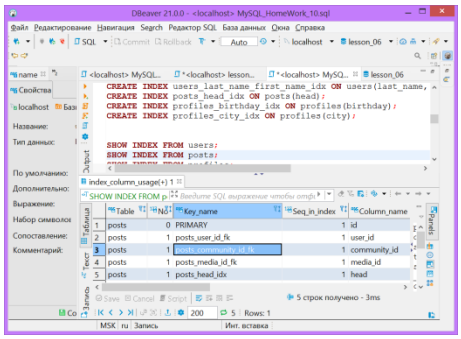
Выполним скрипт создания индексов:

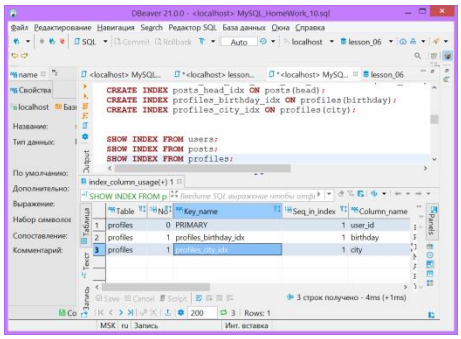
```
CREATE INDEX users_last_name_first_name_idx ON users(last_name, first_name);
CREATE INDEX posts_head_idx ON posts(head);
CREATE INDEX profiles_birthday_idx ON profiles(birthday);
CREATE INDEX profiles_city_idx ON profiles(city);
```

Просмотрим результат создания:

```
SHOW INDEX FROM users;
SHOW INDEX FROM posts;
SHOW INDEX FROM profiles;
```







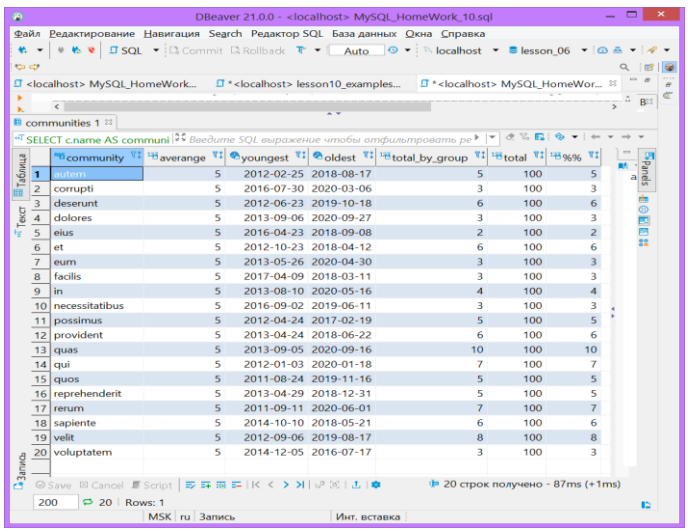
2. Задание на оконные функции. Построить запрос, который будет выводить следующие столбцы:

- имя группы;
- среднее количество пользователей в группах;
- самый молодой пользователь в группе;
- самый старший пользователь в группе;
- общее количество пользователей в группе;
- всего пользователей в системе;
- отношение в процентах (общее количество пользователей в группе / всего пользователей в системе) * 100.

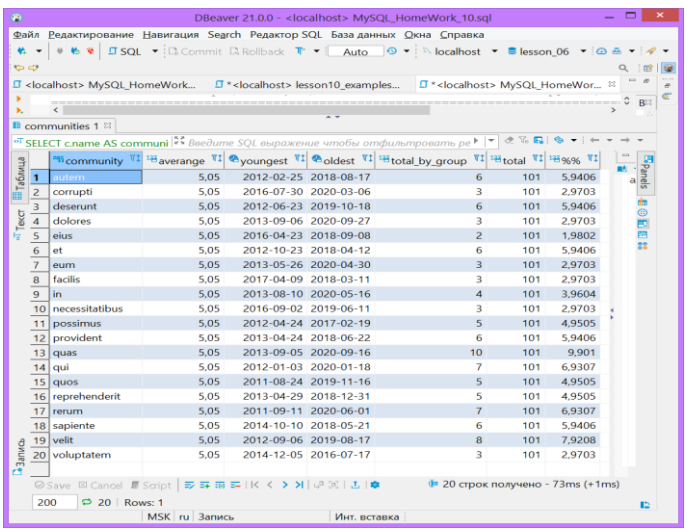
Для начала, составим запрос на «JOIN»:

```
SELECT c.name AS community,
	(SELECT count(*) FROM communities_users) / (SELECT count(*) FROM communities) AS averange,
	min(p.birthday) AS youngest,
	max(p.birthday) AS oldest,
	count(p.user_id) AS total_by_group,
	(SELECT count(*) FROM communities_users) AS total,
	count(p.user_id) / (SELECT count(*) FROM communities_users) * 100 AS "%%"
	FROM communities c
	LEFT JOIN communities_users cu ON cu.community_id = c.id
	JOIN profiles p ON p.user_id = cu.user_id
	GROUP BY c.id
	ORDER BY c.name ;
```

Добавим строку в «communities_users» для наглядности (чтобы количество членов в группе не ровнялось %%) «*INSERT INTO communities_users (community_id, user_id) value (20, 77);*»



	community	averange	youngest	oldest	total_by_group	total	%%
1	autem	5	2012-02-25	2018-08-17	5	100	5
2	corrupti	5	2016-07-30	2020-03-06	3	100	3
3	deserunt	5	2012-06-23	2019-10-18	6	100	6
4	dolores	5	2013-09-06	2020-09-27	3	100	3
5	eius	5	2016-04-23	2018-09-08	2	100	2
6	et	5	2012-10-23	2018-04-12	6	100	6
7	eum	5	2013-05-26	2020-04-30	3	100	3
8	facilis	5	2017-04-09	2018-03-11	3	100	3
9	in	5	2013-08-10	2020-05-16	4	100	4
10	necessitatibus	5	2016-09-02	2019-06-11	3	100	3
11	possimus	5	2012-04-24	2017-02-19	5	100	5
12	provident	5	2013-04-24	2018-06-22	6	100	6
13	quas	5	2013-09-05	2020-09-16	10	100	10
14	qui	5	2012-01-03	2020-01-18	7	100	7
15	quos	5	2011-08-24	2019-11-16	5	100	5
16	reprehenderit	5	2013-04-29	2018-12-31	5	100	5
17	rerum	5	2011-09-11	2020-06-01	7	100	7
18	sapiente	5	2014-10-10	2018-05-21	6	100	6
19	velit	5	2012-09-06	2019-08-17	8	100	8
20	voluptatem	5	2014-12-05	2016-07-17	3	100	3

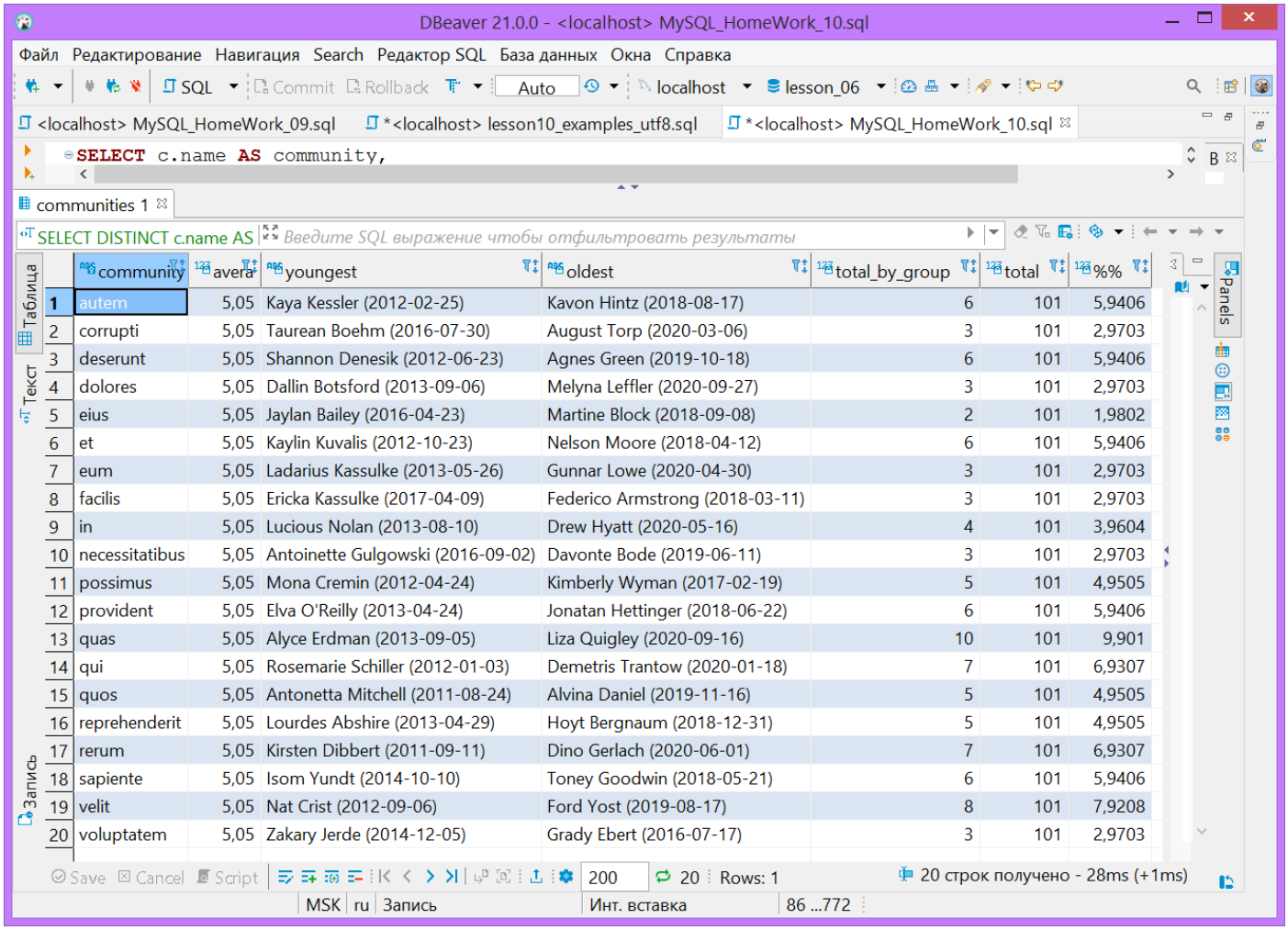


	community	averange	youngest	oldest	total_by_group	total	%%
1	autem	5,05	2012-02-25	2018-08-17	6	101	5,9406
2	corrupti	5,05	2016-07-30	2020-03-06	3	101	2,9703
3	deserunt	5,05	2012-06-23	2019-10-18	6	101	5,9406
4	dolores	5,05	2013-09-06	2020-09-27	3	101	2,9703
5	eius	5,05	2016-04-23	2018-09-08	2	101	1,9802
6	et	5,05	2012-10-23	2018-04-12	6	101	5,9406
7	eum	5,05	2013-05-26	2020-04-30	3	101	2,9703
8	facilis	5,05	2017-04-09	2018-03-11	3	101	2,9703
9	in	5,05	2013-08-10	2020-05-16	4	101	3,9604
10	necessitatibus	5,05	2016-09-02	2019-06-11	3	101	2,9703
11	possimus	5,05	2012-04-24	2017-02-19	5	101	4,9505
12	provident	5,05	2013-04-24	2018-06-22	6	101	5,9406
13	quas	5,05	2013-09-05	2020-09-16	10	101	9,901
14	qui	5,05	2012-01-03	2020-01-18	7	101	6,9307
15	quos	5,05	2011-08-24	2019-11-16	5	101	4,9505
16	reprehenderit	5,05	2013-04-29	2018-12-31	5	101	4,9505
17	rerum	5,05	2011-09-11	2020-06-01	7	101	6,9307
18	sapiente	5,05	2014-10-10	2018-05-21	6	101	5,9406
19	velit	5,05	2012-09-06	2019-08-17	8	101	7,9208
20	voluptatem	5,05	2014-12-05	2016-07-17	3	101	2,9703

Теперь, воспользуемся оконными функциями:

```
SELECT DISTINCT c.name AS community,
        count(cu.user_id) OVER() / (SELECT count(*) FROM communities) AS average,
        FIRST_VALUE(concat(u.first_name, ' ', u.last_name, ' (', p.birthday, ')')) OVER w AS youngest,
        LAST_VALUE(concat(u.first_name, ' ', u.last_name, ' (', p.birthday, ')')) OVER (PARTITION BY c.id
    RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS oldest,
        count(p.user_id) OVER (PARTITION BY c.id) AS total_by_group,
        count(c.id) OVER() AS total,
        count(p.user_id) OVER (PARTITION BY c.id) /      count(c.id) OVER() * 100 AS "%%"
                FROM (communities c
                LEFT JOIN communities_users cu ON cu.community_id = c.id
        JOIN profiles p ON p.user_id = cu.user_id
        JOIN users u ON u.id = p.user_id
        )
        WINDOW w AS (PARTITION BY c.id ORDER BY c.id, p.birthday)
        ORDER BY community;
```

Не сказать, чтобы стало более кратко и понятно, но удалось вытащить ФИО самого молодого и самого «старого» в группе. Остался вопрос, как избавиться от подзапроса «*(SELECT count(*) FROM communities)*». При озвучивании ДЗ, преподаватель упомянул «…без группировки…» - что, в принципе, выполняется.



	community	avera	youngest	oldest	total_by_group	total	%%
1	autem	5,05	Kaya Kessler (2012-02-25)	Kavon Hintz (2018-08-17)	6	101	5,9406
2	corrupti	5,05	Taurean Boehm (2016-07-30)	August Torp (2020-03-06)	3	101	2,9703
3	deserunt	5,05	Shannon Denesik (2012-06-23)	Agnes Green (2019-10-18)	6	101	5,9406
4	dolores	5,05	Dallin Botsford (2013-09-06)	Melyna Leffler (2020-09-27)	3	101	2,9703
5	eius	5,05	Jaylan Bailey (2016-04-23)	Martine Block (2018-09-08)	2	101	1,9802
6	et	5,05	Kaylin Kuvalis (2012-10-23)	Nelson Moore (2018-04-12)	6	101	5,9406
7	eum	5,05	Ladarius Kassulke (2013-05-26)	Gunnar Lowe (2020-04-30)	3	101	2,9703
8	facilis	5,05	Ericka Kassulke (2017-04-09)	Federico Armstrong (2018-03-11)	3	101	2,9703
9	in	5,05	Lucious Nolan (2013-08-10)	Drew Hyatt (2020-05-16)	4	101	3,9604
10	necessitatibus	5,05	Antoinette Gulgowski (2016-09-02)	Davonte Bode (2019-06-11)	3	101	2,9703
11	possimus	5,05	Mona Cremin (2012-04-24)	Kimberly Wyman (2017-02-19)	5	101	4,9505
12	provident	5,05	Elva O'Reilly (2013-04-24)	Jonatan Hettinger (2018-06-22)	6	101	5,9406
13	quas	5,05	Alyce Erdman (2013-09-05)	Liza Quigley (2020-09-16)	10	101	9,901
14	qui	5,05	Rosemarie Schiller (2012-01-03)	Demetris Trantow (2020-01-18)	7	101	6,9307
15	quos	5,05	Antonetta Mitchell (2011-08-24)	Alvina Daniel (2019-11-16)	5	101	4,9505
16	reprehenderit	5,05	Lourdes Abshire (2013-04-29)	Hoyt Bergnaum (2018-12-31)	5	101	4,9505
17	rerum	5,05	Kirsten Dibbert (2011-09-11)	Dino Gerlach (2020-06-01)	7	101	6,9307
18	sapiente	5,05	Isom Yundt (2014-10-10)	Toney Goodwin (2018-05-21)	6	101	5,9406
19	velit	5,05	Nat Crist (2012-09-06)	Ford Yost (2019-08-17)	8	101	7,9208
20	voluptatem	5,05	Zakary Jerde (2014-12-05)	Grady Ebert (2016-07-17)	3	101	2,9703