# Процедуры

## Описание задачи

Напишем процедуру, которая будет предлагать пользователям новых друзей.

Критерии выбора пользователей:

* из одного города
* состоят в одной группе
* друзья друзей

Из выборки будем показывать 5 человек в случайной комбинации.

## Код процедуры

|  |
| --- |
| **drop** **procedure** **if** **exists** friendship\_offers;  delimiter //  **create** **procedure** friendship\_offers(**in** for\_user\_id **INT**)  **begin**  -- общий город  **select** p2.user\_id  **from** profiles p1  **join** profiles p2  **on** p1.hometown = p2.hometown  **where** p1.user\_id = for\_user\_id  **and** p2.user\_id <> for\_user\_id -- исключим себя    **union**    -- общие группы  **select** uc2.user\_id  **from** users\_communities uc1  **join** users\_communities uc2  **on** uc1.community\_id = uc2.community\_id  **where** uc1.user\_id = for\_user\_id  **and** uc2.user\_id <> for\_user\_id -- исключим себя  **union**    -- друзья друзей  -- получим друзей друзей  -- объединяем таблицу саму с собой 3 раза  -- фильтруем «первую» таблицу по for\_user\_id  **select** fr3.target\_user\_id  **from** friend\_requests fr1  **join** friend\_requests fr2  **on** (fr1.target\_user\_id = fr2.initiator\_user\_id  **or** fr1.initiator\_user\_id = fr2.target\_user\_id)  **join** friend\_requests fr3  **on** (fr3.target\_user\_id = fr2.initiator\_user\_id  **or** fr3.initiator\_user\_id = fr2.target\_user\_id)  **where** (fr1.initiator\_user\_id = for\_user\_id **or** fr1.target\_user\_id = for\_user\_id)  **and** fr2.status = 'approved' -- оставляем только подтвержденную дружбу  **and** fr3.status = 'approved'  **and** fr3.target\_user\_id <> for\_user\_id -- исключим себя    **order** **by** rand() -- будем брать всегда случайные записи  **limit** 5; -- ограничим всю выборку до 5 строк  **END**// -- не забываем наш новый разделитель  DELIMITER ; -- вернем прежний разделитель |

## Вызов продедуры / результаты

* Каждый раз при вызове процедуры с одним и тем же параметром мы видим разный результат.

|  |
| --- |
| **CALL** friendship\_offers(1); |
|  |

# Функции

* Наша функция будет только читать данные => **READS** **SQL** **DATA**

|  |
| --- |
| /\*  Направленность дружбы  Кол-во приглашений в друзья к пользователю  /  Кол-во приглашений в друзья от пользователя  Чем больше - популярность выше  Если значение меньше единицы - пользователь инициатор связей.  \*/  **USE** vk;  **DROP** **FUNCTION** **IF** **EXISTS** func\_friendship\_direction;  **DELIMITER //** -- выставим разделитель  **CREATE** **FUNCTION** func\_friendship\_direction(check\_user\_id **INT**)  **RETURNS** **FLOAT** **READS** **SQL** **DATA**  **BEGIN**  -- объявим переменные  **DECLARE** requests\_to\_user **INT**;  **DECLARE** requests\_from\_user **INT**;  **DECLARE** `\_result` **FLOAT**;    -- получим запросы к пользователю  **SET** requests\_to\_user = (  **SELECT** **COUNT**(\*)  **FROM** friend\_requests  **WHERE** target\_user\_id = check\_user\_id);    -- получим запросы от пользователя  /\* SET requests\_from\_user = (  SELECT count(\*)  from friend\_requests  where check\_user\_id = initiator\_user\_id  );\*/  **SELECT** **COUNT**(\*)  **INTO** requests\_from\_user  **FROM** friend\_requests  **WHERE** initiator\_user\_id = check\_user\_id;  -- разделим первое на второе  **if** requests\_from\_user > 0 **then**  **set** `\_result` = requests\_to\_user / requests\_from\_user;  **else**  **set** `\_result` = 99999;  **end** **if**;  -- вернем результат  **return** `\_result`;  **END**// -- не забываем наш новый разделитель  **DELIMITER ;** -- вернем прежний разделитель |

## Вызов функции / результаты

|  |
| --- |
| **SELECT** TRUNCATE(func\_friendship\_direction(1), 2); |
|  |
| **SELECT** TRUNCATE(func\_friendship\_direction(11), 2); |
|  |

# Транзакции

## Добавление нового пользователя

Когда нам нужно добавить полную инфу о пользователе, то проблема/особенность данной операции в необходимости внесения записей сразу в 2 таблицы (**users, profiles**). Соответственно, надо либо сделать все, либо – ничего (отменить начатое в случае невозможности завершить действие).

* Добавим нового пользователя (сразу в 2 таблицы)

|  |
| --- |
| -- Транзакция по добавлению нового пользователя  **START** **TRANSACTION**;  **INSERT** **INTO** users (firstname, lastname, email, phone)  **VALUES** ('New', 'User', 'new@mail.com', 454545456);  **SELECT** @last\_user\_id := (**SELECT** **MAX**(id) **FROM** users); -- опасный способ  **SELECT** @last\_user\_id := **last\_insert\_id**(); -- лучше так  **INSERT** **INTO** profiles (user\_id, gender, birthday, hometown)  **VALUES** (@last\_user\_id, 'M', '1999-10-10', 'Moscow');    **COMMIT**; |
| * Результат пойдем искать в 2 таблицах: users, profiles * Должна была добавиться новая запись в кждоой из них |

## Откат транзакции

* Поменяем последнюю строку скрипта с **COMMIT** на **ROLLBACK**

|  |
| --- |
| -- Транзакция по добавлению нового пользователя  **START** **TRANSACTION**;  **INSERT** **INTO** users (firstname, lastname, email, phone)  **VALUES** ('New', 'User', 'new3@mail.com', 454545456);  **SELECT** @last\_user\_id := (**SELECT** id **FROM** users **WHERE** email = 'new@mail.com');  **INSERT** **INTO** profiles (user\_id, gender, birthday, hometown)  **VALUES** (@last\_user\_id, 'M', '1999-10-10', 'Moscow');    **ROLLBACK**; |
| * Скрипт исполнится без ошибок * Новых строк в таблицы добавлено не будет |

## Ошибки внутри транзакции

* Следующая транзакция вызовет ошибку

|  |
| --- |
| -- Транзакция с ошибкой в дате рождения  **START** **TRANSACTION**;  **INSERT** **INTO** users (firstname, lastname, email, phone)  **VALUES** ('New', 'User', 'new@mail.com', 454545456);  **SELECT** @last\_user\_id := **last\_insert\_id**();  **INSERT** **INTO** profiles (user\_id, gender, birthday, hometown)  **VALUES** (@last\_user\_id, 'M', '1999-13-13', 'Moscow'); -- некорректная дата    **COMMIT**; |
|  |

* Добавм код транзакции в процедуру
  + Добавим входные параметры (фио и прочие данные пользователя)
  + Научим ее возвращать инфу об ошибке

|  |
| --- |
| -- обработка ошибки в транзакции  **DROP** **PROCEDURE** **IF** **EXISTS** `sp\_add\_user`;  **DELIMITER $$**  **CREATE** **PROCEDURE** `sp\_add\_user`(firstname **varchar**(100), lastname **varchar**(100), email **varchar**(100), phone **varchar**(12), hometown **varchar**(50), photo\_id **INT**, **OUT** tran\_result **varchar**(200))  **BEGIN**  **DECLARE** `\_rollback` **BOOL** **DEFAULT** 0;  **DECLARE** code **varchar**(100);  **DECLARE** error\_string **varchar**(100);  **DECLARE** last\_user\_id **int**;  **DECLARE** **CONTINUE** HANDLER **FOR** **SQLEXCEPTION**  **begin**  **SET** `\_rollback` = 1;  **GET** stacked **DIAGNOSTICS** **CONDITION** 1  code = RETURNED\_SQLSTATE, error\_string = MESSAGE\_TEXT;  **set** tran\_result := **concat**('Error occured. Code: ', code, '. Text: ', error\_string);  **end**;    **START** **TRANSACTION**;  **INSERT** **INTO** users (firstname, lastname, email, phone)  **VALUES** (firstname, lastname, email, phone);    **INSERT** **INTO** profiles (user\_id, hometown, photo\_id)  **VALUES** (**last\_insert\_id**(), hometown, photo\_id);    **IF** `\_rollback` **THEN**  **ROLLBACK**;  **ELSE**  **set** tran\_result := 'ok';  **COMMIT**;  **END** **IF**;  END$$  **DELIMITER ;**  -- вызываем процедуру  **call** sp\_add\_user('New', 'User', 'new87@mail.com', 454545456, 'Moscow', -1, @tran\_result);  -- смотрим результат  **select** @tran\_result; |
| * Теперь в случае ошибки мы видим ее текст |

# Временно выключить проверку внешних ключей

* Отключим проверку внешних ключей

SET FOREIGN\_KEY\_CHECKS = 0;

* Выполним удаление

|  |
| --- |
| **delete** **from** users  **where** id = ... |

* Включим обратно проверку ключей

SET FOREIGN\_KEY\_CHECKS = 1;

* Посмотреть текущее значение этой переменной можно так:

|  |
| --- |
| -- посмотреть значение переменной foreign\_key\_checks  **SELECT** @@**GLOBAL**.foreign\_key\_checks, @@**SESSION**.foreign\_key\_checks; |
|  |

# Представления

* Напишем представление, выбирающее друзей пользователей

|  |
| --- |
| -- представление, выбирающее друзей пользователей  **CREATE** **or** **replace** **VIEW** view\_friends  **AS**  **select** \*  **FROM** users u  **JOIN** friend\_requests fr **ON** u.id = fr.target\_user\_id  **WHERE**  fr.status = 'approved'  **union**    **select** \*  **FROM** users u  **JOIN** friend\_requests fr **ON** u.id = fr.initiator\_user\_id  **WHERE**  fr.status = 'approved' |
|  |

* Теперь можно обращаться к представлению следующим образом

|  |
| --- |
| -- получим друзей пользователя из представления  **select** \*  **from** view\_friends  **where** id = 1 |
|  |

## Удаление представления

* Для удаления представления используется команда **DROP VIEW**

|  |
| --- |
| -- удаление представления  **drop** **view** view\_friends; |
|  |

# Триггеры

* Напишем триггер для операции обновления профиля (дата рождения должна быть в прошлом)

|  |
| --- |
| -- триггер для проверки возраста пользователя перед обновлением  **DELIMITER //**  **CREATE** **TRIGGER** check\_user\_age\_before\_update **BEFORE** **UPDATE** **ON** profiles  **FOR** **EACH** **ROW**  **begin**  **IF** **NEW**.birthday >= **CURRENT\_DATE**() **THEN**  **SIGNAL** **SQLSTATE** '45000' **SET** MESSAGE\_TEXT = 'Update Canceled. Birthday must be in the past!';  **END** **IF**;  **END**//  **DELIMITER ;** |
|  |

* Напишем триггер для корректировки возраста пользователя

|  |
| --- |
| -- триггер для корректировки возраста пользователя при вставке новых строк  **drop** **TRIGGER** **if** **exists** check\_user\_age\_before\_insert;  **DELIMITER //**  **CREATE** **TRIGGER** check\_user\_age\_before\_insert **BEFORE** **INSERT** **ON** profiles  **FOR** **EACH** **ROW**  **begin**  **IF** **NEW**.birthday > **CURRENT\_DATE**() **THEN**  **SET** **NEW**.birthday = **CURRENT\_DATE**();  **END** **IF**;  **END**//  **DELIMITER ;** |
|  |