# 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); |
|  |

# 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)
* Заодно потренируемся в работе с переменными

|  |
| --- |
| -- 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, но пользователь будет добавлен (только в таблицу **users**)

|  |
| --- |
| -- 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**; |
|  |

* Для обработки таких ошибок есть 2 сценария:
  + Обрабатывать их на сервере, который обращается к БД
  + Писать транзакции в хранимых процедурах со следующей оберткой
* Напишем процедуру с примером обработки ошибок в транзакциях

|  |
| --- |
| -- обработка ошибки в транзакции  **DROP** **PROCEDURE** **IF** **EXISTS** `sp\_add\_user`;  **DELIMITER $$**  **CREATE** **PROCEDURE** `sp\_add\_user`(**OUT** tran\_result **varchar**(20))  **BEGIN**  **DECLARE** `\_rollback` **BOOL** **DEFAULT** 0;  **DECLARE** **CONTINUE** HANDLER **FOR** **SQLEXCEPTION** **SET** `\_rollback` = 1;  **START** **TRANSACTION**;    **INSERT** **INTO** users (firstname, lastname, email, phone)  **VALUES** ('New', 'User', 'new37@mail.com', 454545456);    **INSERT** **INTO** profiles (user\_id, gender, birthday, hometown)  **VALUES** (**last\_insert\_id**(), 'M', '1999-11-11', 'Moscow');    **IF** `\_rollback` **THEN**  **set** tran\_result = 'ROLLBACK';  **ROLLBACK**;  **ELSE**  **set** tran\_result := 'COMMIT';  **COMMIT**;  **END** **IF**;  END$$  **DELIMITER ;**  -- вызываем процедуру  **call** sp\_add\_user(@tran\_result);  -- смотрим результат  **select** @tran\_result; |
| * Транзакция откатилась * Проверим таблицу users, убедимся, что нет новой записи * Но в данном случае мы не знаем кода и текста ошибки |

* Поправим 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 |

## Удаление пользователя

* Напишем транзакцию, удаляющую запись о пользователе из обеих таблиц (**users** и **profiles**)

|  |
| --- |
| -- Транзакция по удалению пользователя  **START** **TRANSACTION**;  **SELECT** @last\_user\_id := **last\_insert\_id**();  **delete** **from** users  **where** id = @last\_user\_id;  **delete** **from** profiles  **where** user\_id = @last\_user\_id;  **COMMIT**; |
| * Ошибка за-за наличия внешнего ключа |

* поменяем местами порядок удаления в таблицах users <-> profiles

|  |
| --- |
| -- Транзакция по удалению пользователя  **START** **TRANSACTION**;  **SELECT** @last\_user\_id := **last\_insert\_id**();  **delete** **from** profiles  **where** user\_id = @last\_user\_id;  **delete** **from** users  **where** id = @last\_user\_id;  **COMMIT**; |
| * Пользователя с почтой 'new@mail.com' больше нет в БД |

### Временно выключить проверку внешних ключей

Но если бы у удаляемого пользователя были посты, фотки, лайки, сообщения и т.д., то мы не смогли бы его удалить. В этом случае можно отключить проверку внешних ключей.

Это фича именно MySQL. В других СУБД возможность отключения проверки внешних ключей реализована по-другому (синтаксис отличается).

* 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; |
|  |

* Поменяем значение и увидим, что повлияло это только на @@session

|  |
| --- |
| **SET FOREIGN\_KEY\_CHECKS = 0;**  **SELECT** @@**GLOBAL**.foreign\_key\_checks, @@**SESSION**.foreign\_key\_checks; |
|  |

### Изменить тип внешнего ключа

Внешний ключ может иметь разную логику поведения при удалении основной записи (на которую он ссылается):

|  |
| --- |
| *reference\_option*:  RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT |

* Попробуем удалить сообщество, в котором есть члены

|  |
| --- |
| -- НЕ сможем удалить  **delete** **from** communities  **where** id = 10; |
| * Не можем этого сделать из-за ограничений внешнего ключа |

Поменяем логику поведения внешнего ключа при удалении записей с запрета (**RESTRICT**) на каскадное удаление (**CASCADE**). В этом случае нам необходимо:

* Удалить существущий внешний ключ

|  |
| --- |
| -- удалим внешний ключ  **ALTER** **TABLE** vk.users\_communities  **DROP** **FOREIGN** **KEY** <fk\_name>; |
|  |

* Заново его добавить с требуемой новой логикой работы

|  |
| --- |
| -- создадим заново внешний ключ с правилом каскадного удаления зависимых записей  **ALTER** **TABLE** vk.users\_communities  **ADD** **CONSTRAINT** users\_communities\_fk  **FOREIGN** **KEY** (community\_id)  **REFERENCES** vk.communities(id) **ON** **DELETE** **cascade**; |
|  |

* Теперь сможем удалить любое сообщество (но вместе с его членами)

|  |
| --- |
| -- сможем удалить и сразу удалим зависимые записи  **delete** **from** communities  **where** id = 10; |
| * Сообщество удалено успешно * Инфа о членстве пользователей в этом сообществе тоже удалена (табл. **users\_communities**) |

# Функции

## Описание задачи

Напишем функцию, которая будет считать коэффициент направленности дружбы.

Формула: Кол-во приглашений в друзья к пользователю / (разделим на) Кол-во заявок в друзья от пользователя.

Из результата (которым будет некоторое вещественное число) можно будет делать некоторые выводы:

1. Чем больше значение, тем популярность пользователя выше.
2. Если значение меньше единицы - пользователь инициатор связей (и наоборот).

## Код фукнции

* 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** friendship\_direction;  DELIMITER // -- выставим разделитель  **CREATE** **FUNCTION** friendship\_direction(check\_user\_id **INT**)  **RETURNS** **FLOAT** **READS** **SQL** **DATA**  **BEGIN**  -- объявим переменные  **DECLARE** requests\_to\_user **INT**;  **DECLARE** requests\_from\_user **INT**;    -- 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** initiator\_user\_id = check\_user\_id);    -- разделим первое на второе и вернем результат  **RETURN** requests\_to\_user / requests\_from\_user;  **END**// -- не забываем наш новый разделитель  DELIMITER ; -- вернем прежний разделитель |

### Опции при создании функции

Офф. дока: <https://dev.mysql.com/doc/refman/8.0/en/stored-programs-logging.html>

## Вызов функции / результаты

* Из результатов видно, что пользователь чаще сам инициатор дружбы

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

# Отличия процедур от функций

* По-разному вызываются
* Процедуры могут иметь in, out, inout параметры
* Функция должна возвращать значение
* Внутри другого выражения можно вызвать только функцию
* В хранимых процедурах можно использовать динамический SQL, но не в функциях или триггерах.
* Функции обычно используются для вычислений, процедуры - для выполнения бизнес-логики

# Представления

## Синтаксис

Офф. Дока:

<https://dev.mysql.com/doc/refman/8.0/en/create-view.html>

|  |
| --- |
| CREATE  [OR REPLACE]  [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]  [DEFINER = *user*]  [SQL SECURITY { DEFINER | INVOKER }]  VIEW *view\_name* [(*column\_list*)]  AS *select\_statement*  [WITH [CASCADED | LOCAL] CHECK OPTION] |

Использование алгоритма **MERGE** требует соответствия 1 к 1 между строками таблицы и основанного на ней представления. Алгоритм **TEMPTABLE** заносит содержимое представления во временную таблицу

**WITH CHECK OPTION** — при использовании данной конструкции все добавляемые или изменяемые строки будут проверяться на соответствие определению представления. В случае несоответствия данное изменение не будет выполнено. Обратите внимание, что при указании данной конструкции для необновляемого представления возникнет ошибка и представление не будет создано. (подробнее речь об этом пойдет ниже).

## Пример представления с БД vk

* 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

ENG docs:

<https://dev.mysql.com/doc/refman/8.0/en/create-trigger.html>

RUS docs:

Отсутствует офф. Дока

## Синтаксис

|  |
| --- |
| CREATE  [DEFINER = *user*]  TRIGGER *trigger\_name*  *trigger\_time* *trigger\_event*  ON *tbl\_name* FOR EACH ROW  [*trigger\_order*]  *trigger\_body*  *trigger\_time*: { BEFORE | AFTER }  *trigger\_event*: { INSERT | UPDATE | DELETE }  *trigger\_order*: { FOLLOWS | PRECEDES } *other\_trigger\_name* |

## Примеры с БД vk

* 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 ;** |
|  |

* Проверим работу триггера обновления

|  |
| --- |
| -- обновим ДР пользователя с некорректной датой  **UPDATE** profiles  **SET** birthday='2020.10.10'  **WHERE** user\_id = 1; |
| * Видим наше сообщение об ошибке |

* Проверим работу триггера на вставку
* Удалим профиль с user\_id = 3
* Добавим новый профиль с user\_id = 3 и ДР в будущем и сохраним

|  |
| --- |
|  |
| * Вставилась запись с сегодняшней датой |

## Особенности в MySQL

* При удалении записи users (ON DELETE CASCADE) в зависимой profiles не сработает триггер на удаление – связано с архитектурой MySQL (движки отдельно от ядра)