Стоколесов М. С.

М8О-207М-19

**Отчет к финальной работе**

**по дисциплине: «Проектирование БД»**

**Цель работы**

Закрепить навыки работы с СУБД PostgreSQL путем разработки финального проекта, представляющего собой наполненную базу данных, снабженную набором типичных запросов, триггерами и процедурами.

# Задачи

1. Выбрать предметную область;
2. Составить описание предметной области;
3. Указать пользу, которую даст база данных;
4. Составить требования, предъявляемые к БД;
5. Спроектировать концептуальную, логическую и физическую модели БД;
6. Определить, в каких нормальных формах находятся отношения;
7. Реализовать спроектированную базу данных.

# Ход работы

1. **Описание предметной области**

В качестве предметной области был выбран *форум*, на котором пользователи могут выкладывать публикации, а также обсуждать их и давать оценки.

Таким образом выбранная предметная область объединяет в себе следующие 4 основных сущности:

* Пользователей;
* Публикации;
* Комментарии пользователей;
* Оценки пользователей.

При этом каждая публикация может принадлежать набору тем, что позволит осуществлять пользователям более быстрый поиск с помощью фильтрации.

Кроме того данная предметная область требует наличия минимального набора средств для борьбы с непорядочными пользователями, в который может входить, например, черный список таких пользователей.

Таким образом, наличие базы данных для данной предметной области позволит хранить текущее состояние форума: зарегистрированных пользователей, их публикации, а также оценки и комментарии к ним.

1. **Требования к базе данных**
   1. **Требования к данным**
      1. **Пользователи**

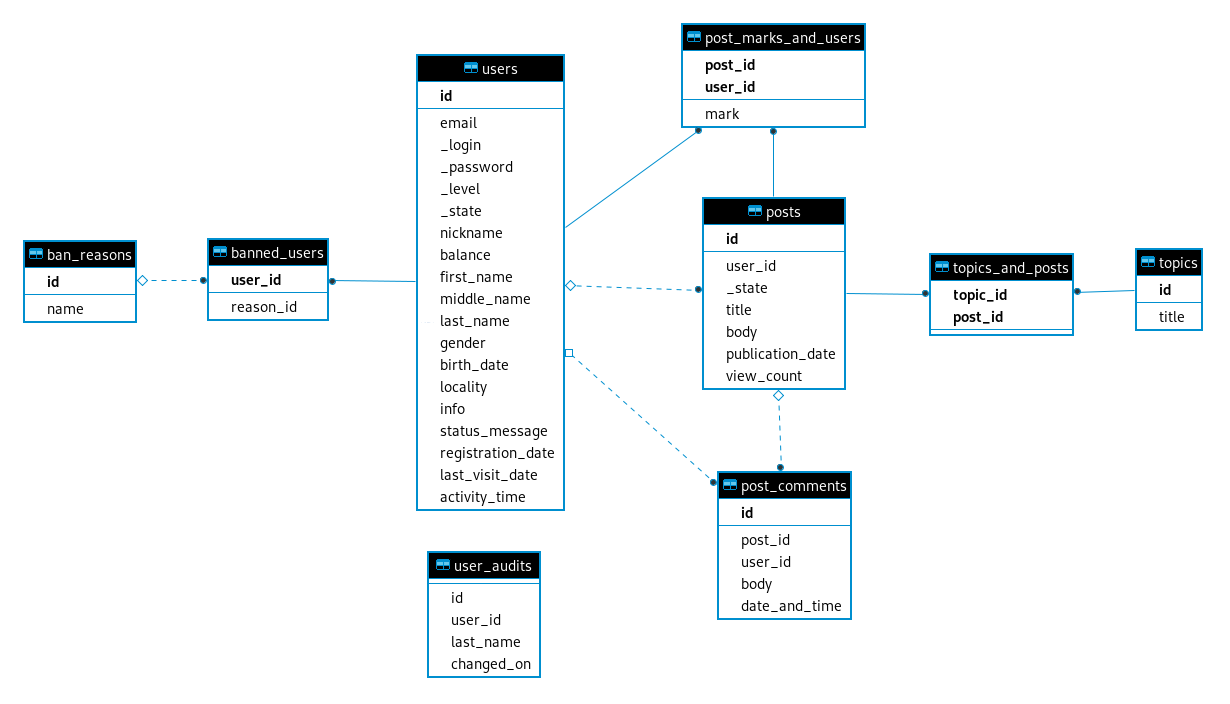
Каждый пользователь имеет электронную почту, имя, фамилию, а также псевдоним на форуме. Кроме этого у каждого пользователя так же имеется свой виртуальный счет, с помощью которого можно осуществлять внутренние покупки на форуме. Также для каждого пользователя необходимо хранить информацию о дате регистрации, состоянии аккаунта на текущий момент (заблокированный или нет).

* + 1. **Публикации, комментарии и оценки**

Каждая публикация должна иметь заголовок, а также, очевидно, само содержание. Кроме этого для каждой публикации необходимо хранить список тем, к которой она принадлежит, дату ее создания, а также количество просмотров. Также необходимо хранить информацию об оценках пользователей и их комментариях.

* 1. **Требования к транзакциям**
     1. **Ввод данных**
* Ввести данные о новом пользователе (имя, фамилия, псевдоним, почта, логин, пароль);
* Добавить публикацию (тема, содержание, автор);
* Поставить оценку публикации;
* Добавить комментарий к публикации;
* Присвоить публикации тему;
* Добавить пользователя в список заблокированных;
  + 1. **Изменение / удаление данных**
* Изменить данные о пользователе (идентификатор, изменение);
* Удаление пользователя (сначала нужно предварительно заморозить аккаунт)
* Изменение / Удаление публикации (тема, содержание);
* Изменение оценки публикации;
* Изменение набора тем, которым принадлежит публикация;
* Удаление пользователя из списка заблокированных.
  + 1. **Запросы к данным**
* Для каждого пользователя вывести среднее количество просмотров его публикаций;
* Вывести пользователей, у которых среднее количество просмотров публикаций выше определенного порога;
* Отсортировать пользователей по убыванию популярности их публикаций;
* Вывести список пар таких пользователей, в которых год рождения одного пользователя совпадает с годом регистрации другого;
* Вывести пользователей, имеющих максимальное количество просмотров какой-либо публикации, превышающее определенный порог;
* Вывести всех пользователей вместе с их публикациями.

1. **Модели базы данных**
   1. **Концептуальная модель**
      1. **Диаграмма**

****

* + 1. **Описание**
* **Пользователи (users)**

Как следует из требований, таблица с пользователями должна содержать в себе информацию об имени, фамилии, псевдониме, балансе, дате регистрации, а также логине, пароле и почте пользователя.

* **Публикации (posts)**

Таблица с публикациями содержит в себе заголовок, содержание, дату, количество просмотров, а также автора публикации.

* **Оценки публикаций (posts\_marks\_and\_users**

Таблица с оценками в себе хранит только соответствие между пользователями, публикациями и оценками.

* **Комментарии к публикациями (post\_comments)**

В таблице с комментариями необходимо хранить идентификатор комментируемой публикации, автора комментария, а также его содержание и дату отправления.

* **Темы и публикации (topics\_and\_posts)**

Таблица с темами и публикациями, как следует из ее названия, хранит только соответствие между указанными сущностями.

* **Список возможных тем (topics)**

В таблице с темами хранится набор всех возможных тем с идентификаторами.

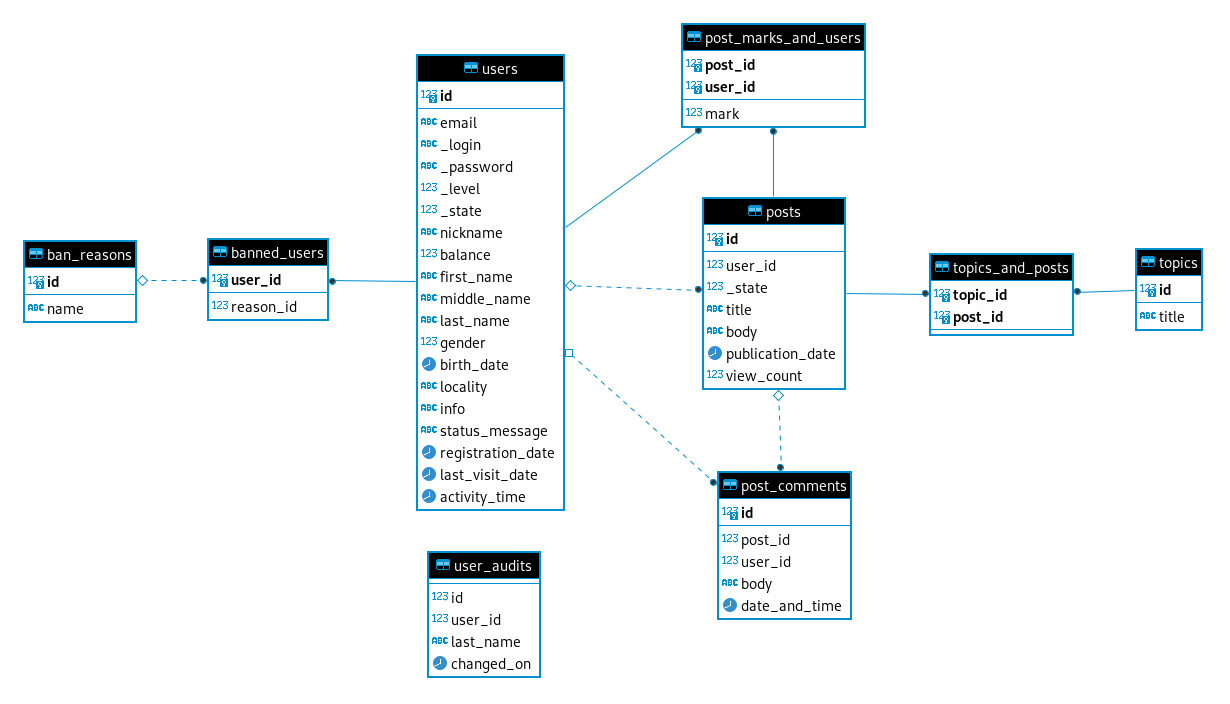
* **Заблокированные пользователи (banned\_users)**

В таблице с заблокированными пользователями хранятся идентификаторы пользователей, которые были заблокированы, причина указана для каждого пользователя.

* **Список возможных причин блокировок (ban\_reasons)**

В таблице с возможными причинами блокировок хранится набор всех возможных причин, из-за которых могут быть заблокированы аккаунты пользователей.

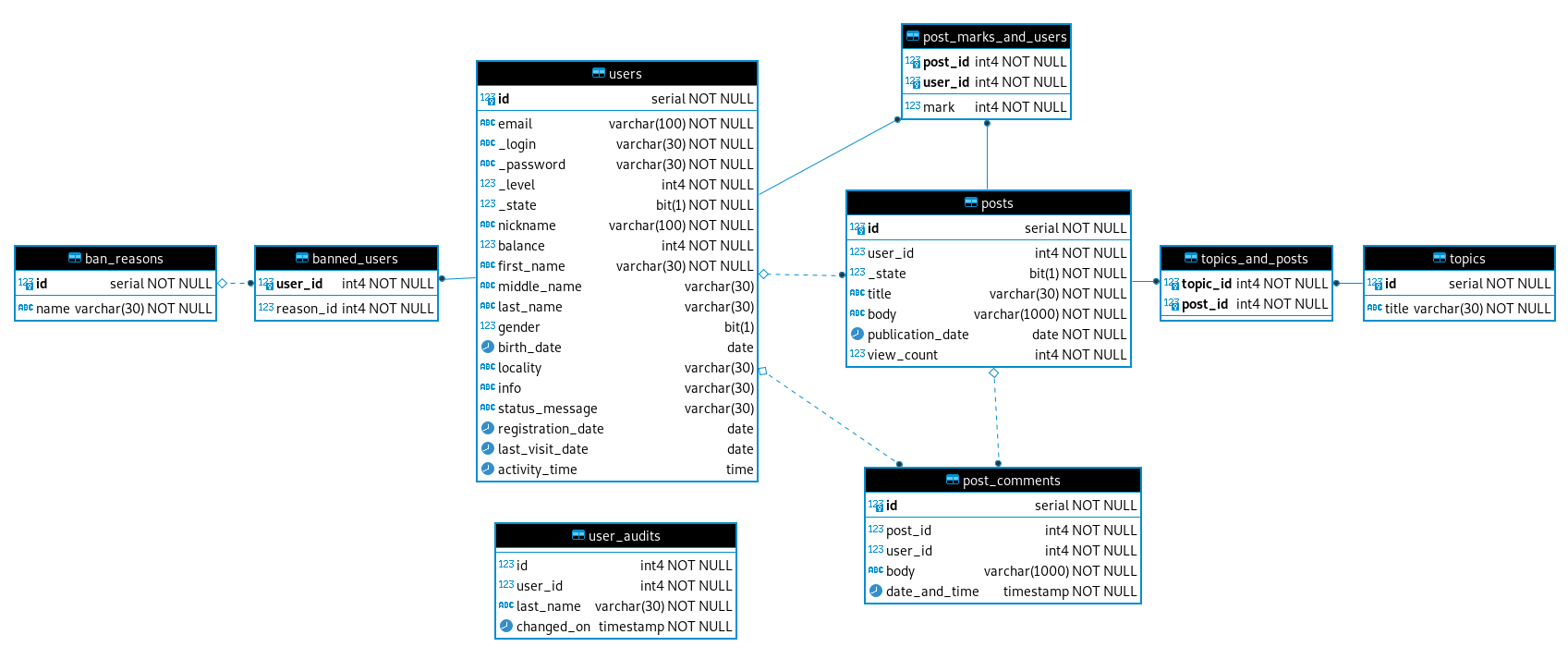
* 1. **Логическая модель**
     1. **Диаграмма**

****

* + 1. **Описание**

В настоящей базе данных представлено два способа задания первичного ключа:

* Простой первичный ключ:
  + Пользователи
  + Заблокированные пользователи
  + Причины блокировок
  + Публикации
  + Комментарии
  + Темы
* Составной первичный ключ (для обесп. уникальности):
  + Публикации и оценки
  + Темы и публикации
  1. **Физическая модель**
     1. **Диаграмма**

****

* + 1. **Описание**

В базе данных были применены следующие типы данных:

* varchar -- для строковых полей (имена, заголовки, темы, ...)
* int -- для целочисленных (идентификаторы, баланс, ...)
* date -- для дат (регистрации, публикации, ...)
* time -- для времени (активности)
* bit -- для тех случаев, когда можно ограничиться логическим значением истина / ложь (например, заморожен аккаунт или нет)

1. **Нормальные формы**

Ограничимся определениями трех самых главных нормальных форм:

* **Первая НФ:**

Таблица находится в первой нормальной форме (1НФ) тогда и только тогда, когда в любом допустимом значении таблицы каждый ее кортеж содержит только одно значение для каждого из атрибутов.

* **Вторая НФ:**

Таблица находится во второй нормальной форме тогда и только тогда, когда она находится в первой нормальной форме и каждый неключевой атрибут неприводимо (функционально полно) зависит от её потенциального ключа. Функционально полная зависимость означает, что если потенциальный ключ является составным, то атрибут зависит от всего ключа и не зависит от его частей.

* **Третья НФ:**

Переменная отношения находится в третьей нормальной форме тогда и только тогда, когда она находится во второй нормальной форме, и отсутствуют транзитивные функциональные зависимости неключевых атрибутов от ключевых.

Таким образом, исходя из определений нормальных форм, заключаем, что каждая таблица настоящей базы данных находится в третьей нормальной форме.

1. **Типичные запросы к базе данных**

* Для каждого пользователя вывести среднее количество просмотров его публикаций:

WITH ids\_with\_stat AS (

SELECT user\_id, AVG(view\_count) AS avg\_count

FROM posts

GROUP BY user\_id

)

SELECT users.nickname,

users.first\_name,

ids\_with\_stat.avg\_count

FROM

ids\_with\_stat JOIN users

on ids\_with\_stat.user\_id = users.id

nickname | first\_name | avg\_count

--------------+------------+------------------

Blossom88 | Марья | 1534.00

GuitarHero82 | Роман | 1800.67

LightBringer | Святослав | 3459.00

Crowbar97 | Gordon | 5348.00

(4 rows)

* Вывести пользователей, у которых среднее количество просмотров публикаций выше определенного порога;

SELECT user\_id, AVG(view\_count) AS avg\_count, SUM(view\_count) AS sum

FROM posts

GROUP BY user\_id

HAVING SUM(view\_count) > 5000;

user\_id | avg\_count | sum

---------+-----------------------+------

5 | 1800.67 | 5402

1 | 5348.00 | 5348

(2 rows)

* Отсортировать пользователей по убыванию популярности их публикаций;

WITH top\_ids AS (

SELECT user\_id, view\_count, rank()

OVER ( PARTITION BY user\_id ORDER BY view\_count DESC )

FROM posts

)

SELECT users.nickname,

users.first\_name,

top\_ids.view\_count,

rank

FROM

top\_ids JOIN users

ON top\_ids.user\_id = users.id;

nickname | first\_name | view\_count | rank

--------------+------------+------------+------

Crowbar97 | Gordon | 5348 | 1

Blossom88 | Марья | 2486 | 1

Blossom88 | Марья | 582 | 2

GuitarHero82 | Роман | 2548 | 1

GuitarHero82 | Роман | 1526 | 2

GuitarHero82 | Роман | 1328 | 3

LightBringer | Святослав | 3459 | 1

(7 rows)

* Вывести список пар таких пользователей, в которых год рождения одного пользователя совпадает с годом регистрации другого;

WITH users\_bds AS (

SELECT nickname, EXTRACT(YEAR FROM birth\_date) as year

FROM users

),

users\_regs AS (

SELECT nickname, EXTRACT(YEAR FROM registration\_date) as year

FROM users

)

SELECT users\_regs.nickname,

users\_bds.nickname,

users\_bds.year

FROM users\_bds, users\_regs

WHERE users\_regs.year = users\_bds.year;

nickname | nickname | year

--------------+-------------+------

IndiaFinder | BloodVlad | 1997

LightBringer | DreamStream | 2010

IndiaFinder | WhiteSheep | 1997

(3 rows)

* Вывести пользователей, имеющих максимальное количество просмотров какой-либо публикации, превышающее определенный порог;

SELECT nickname

FROM users

WHERE EXISTS (

SELECT \* FROM posts

WHERE user\_id = users.id AND view\_count > 1000

);

nickname

--------------

Crowbar97

Blossom88

GuitarHero82

LightBringer

(4 rows)

* Вывести всех пользователей вместе с их публикациями.

SELECT nickname, title

FROM users LEFT OUTER JOIN posts

ON user\_id = users.id;

nickname | title

----------------+---------------------

Crowbar97 | Theoretical physics

Blossom88 | Flower Power

Blossom88 | Autumn Garden

GuitarHero82 | Guitar Lesson № 1

GuitarHero82 | Guitar Lesson № 2

GuitarHero82 | Guitar Lesson № 3

LightBringer | What is kindness?

NoSpamFromHere |

Admin |

WhiteSheep |

Johny19 |

Moderor |

DreamStream |

BloodVlad |

MyBlogTeam |

IndiaFinder |

Julian |

(17 rows)

1. **Реализация базы данных**

Все исходные коды, снабженные комментариями и включающие в себя создание всех приведенных в настоящем отчете таблиц, различных запросов, триггеров и процедур, а также результаты их применения, сохраненные в текстовых файлах, можно найти в прикрепленном к данному отчету архиве или в моем GitHub-репозитории: <https://github.com/Crowbar97/mai-sql-course>.

1. **Выводы**

В ходе выполнения работы были закреплены на практике навыки использования СУБД PostgreSQL. Были спроектированы концептуальная, логическая и физическая модели базы данных. Была реализована и заполнена сама база данных, а также снабжена набором типичных запросов, триггеров и процедур.

**Приложение**

1. **Исходный код для создания и заполнения всех таблиц:**

***-- Опасная зона ------------------------------***

**DROP TABLE IF EXISTS post\_comments;**

**DROP TABLE IF EXISTS post\_marks\_and\_users;**

**DROP TABLE IF EXISTS topics\_and\_posts;**

**DROP TABLE IF EXISTS topics;**

**DROP TABLE IF EXISTS posts;**

**DROP TABLE IF EXISTS banned\_users;**

**DROP TABLE IF EXISTS ban\_reasons;**

**DROP TABLE IF EXISTS users;**

***----------------------------------------------***

***-- Пользователи***

**CREATE TABLE users (**

**id SERIAL NOT NULL,**

**email varchar(100) NOT NULL,**

**\_login varchar(30) NOT NULL,**

**\_password varchar(30) NOT NULL,**

**\_level int NOT NULL,**

**\_state bit NOT NULL,**

**nickname varchar(100) NOT NULL,**

**balance int NOT NULL,**

**first\_name varchar(30) NOT NULL,**

**middle\_name varchar(30),**

**last\_name varchar(30),**

**gender bit,**

**birth\_date date,**

**locality varchar(30),**

**info varchar(30),**

**status\_message varchar(30),**

**registration\_date date,**

**last\_visit\_date date,**

**activity\_time time,**

**CONSTRAINT pk\_users PRIMARY KEY(id)**

**);**

**INSERT INTO users**

**( email, \_login, \_password, \_level, \_state, nickname, balance, first\_name, last\_name, gender, birth\_date, registration\_date) VALUES**

**( 'RealGordonFreeman@XPhysics.com', 'GF789', 'Black159Mesa', 0, B'1', 'Crowbar97', 20, 'Gordon', 'Freeman', B'1', '01/01/1950', '05/30/2004' ),**

**( 'John1940@gmail.com', 'John40', 'YoYo40', 0, B'1', 'Johny19', 80, 'Иван', 'Дубравин', B'1', '05/07/1940', '06/08/2008' ),**

**( 'MaryFlower@yandex.ru', 'Mary357', 'Blossom8And8Die', 0, B'1', 'Blossom88', 90, 'Марья', '-', B'0', '07/18/1988', '03/05/2006' ),**

**( 'WhatIsLove@yandex.ru', 'BloodVlad159', 'BabyDontHurtMe1', 0, B'1', 'BloodVlad', 150, 'Владислав', 'Лихачев', B'1', '04/16/1997', '12/15/2012' ),**

**( 'HellFever999@yandex.ru', 'RomanG', 'GuitarForever!888', 0, B'1', 'GuitarHero82', 1800, 'Роман', 'Конограй', B'1', '12/10/1982', '07/18/2014' ),**

**( 'LightBringer33@mail.ru', 'LB33', 'LastHope0', 0, B'1', 'LightBringer', 1260, 'Святослав', 'Кондратов', B'1', '05/03/1933', '03/24/2010' ),**

**( 'Caesar100@yandex.ru', 'TheOne', 'The1Only1One', 0, B'1', 'Julian', 1250, 'Юлиан', 'Белозеров', B'1', '01/02/0003', '04/08/2007' ),**

**( 'SuperAlex2010@gmail.com', 'Streamer7', 'RoadTo100k', 0, B'1', 'DreamStream', 8800, 'Александр', 'Харитонов', B'1', '06/19/2010', '07/22/2016' ),**

**( 'Columbus51@yahoo.com', 'Columbus', 'Disc1492overy', 0, B'1', 'IndiaFinder', 1200, 'Христофор', 'Колумбов', B'1', '12/10/1951', '12/03/1997' ),**

**( 'Sheep@gmail.com', 'WhiteSheep', 'H@CK\_THE\_W0RLD', 0, B'0', 'WhiteSheep', 2100, 'Иван', 'Овечкин', B'1', '12/18/1997', '07/01/2017' ),**

**( 'SpamOverlord@gmail.com', 'SomeSpam', 'ALOTOFSPAMAHAHAH999', 0, B'1', 'NoSpamFromHere', 500, 'Денис', NULL, B'1', '03/28/1992', '10/09/2016' ),**

**( 'MyBlogAdmin@gmail.com', 'Admin', '#ARENqAL220vFRfrst', 3, B'1', 'Admin', 10000, 'Иван', 'Иванов', B'1', '01/15/1960', '03/01/1978' ),**

**( 'MyBlogModeror@gmail.com', 'Moderor', '#MdrDsadppge78e', 2, B'1', 'Moderor', 4500, 'Денис', 'Денисов', B'1', '02/20/1962', '03/01/1978' ),**

**( 'MyBlogSystem@gmail.com', 'System', '#Ssxwzqr789S', 1, B'1', 'MyBlogTeam', 3200, 'Команда MyBlog', NULL, NULL, NULL, '03/01/1978' );**

**SELECT \* FROM users;**

***-- Возможные причины бана***

**CREATE TABLE ban\_reasons (**

**id SERIAL NOT NULL,**

**name varchar(30) NOT NULL,**

**CONSTRAINT pk\_ban\_reasons PRIMARY KEY(id)**

**);**

**INSERT INTO ban\_reasons**

**( name )**

**VALUES**

**( 'Рассылка спама' ),**

**( 'Попытка взлома сервиса' ),**

**( 'Неадекватное поведение' );**

**SELECT \* FROM ban\_reasons;**

***-- Забаненные пользователи***

**CREATE TABLE banned\_users (**

**user\_id int NOT NULL,**

**reason\_id int NOT NULL,**

**CONSTRAINT pk\_banned\_users PRIMARY KEY(user\_id),**

**CONSTRAINT fk\_pk\_banned\_users\_user\_id FOREIGN KEY(user\_id) REFERENCES users(id),**

**CONSTRAINT fk\_pk\_banned\_users\_reason FOREIGN KEY(reason\_id) REFERENCES ban\_reasons(id)**

**);**

**INSERT INTO banned\_users**

**( user\_id, reason\_id )**

**VALUES**

**( 10, 2 ),**

**( 8, 3 );**

**SELECT \* FROM banned\_users;**

***-- Публикации пользователей***

**CREATE TABLE posts (**

**id SERIAL NOT NULL,**

**user\_id int NOT NULL,**

**\_state bit NOT NULL,**

**title varchar(30) NOT NULL,**

**body varchar(1000) NOT NULL,**

**publication\_date date NOT NULL,**

**view\_count int NOT NULL,**

**CONSTRAINT pk\_posts PRIMARY KEY(id),**

**CONSTRAINT fk\_posts\_user\_id FOREIGN KEY(user\_id) REFERENCES users(id)**

**);**

**INSERT INTO posts**

**(user\_id, \_state, title, body, publication\_date, view\_count) VALUES**

**(1, B'0', 'Theoretical physics', 'Theoretical physics...', '06/10/2005', 5348 ),**

**(3, B'1', 'Flower Power', 'Garden roses...', '05/10/2007', 582 ),**

**(3, B'1', 'Autumn Garden', 'Autumn smell...', '05/09/2008', 2486 ),**

**(5, B'1', 'Guitar Lesson № 1', 'Today I will start...', '07/20/2014', 1328 ),**

**(5, B'1', 'Guitar Lesson № 2', 'Today we will continue...', '08/01/2014', 1526 ),**

**(5, B'1', 'Guitar Lesson № 3', 'Today we will play our first composition...', '08/15/2014', 2548 ),**

**(6, B'1', 'What is kindness?', 'What is kindness? It is a very good question...', '03/06/2011', 3459 );**

**SELECT \* FROM posts;**

***-- Темы постов***

**CREATE TABLE topics (**

**id SERIAL NOT NULL,**

**title varchar(30) NOT NULL,**

**CONSTRAINT pk\_topics PRIMARY KEY(id)**

**);**

**INSERT INTO topics**

**( title )**

**VALUES**

**( 'Science' ),**

**( 'Tutorials' ),**

**( 'Common' ),**

**( 'Music' );**

**SELECT \* FROM topics;**

***-- Соответствия между темами и постами***

**CREATE TABLE topics\_and\_posts (**

**topic\_id int NOT NULL,**

**post\_id int NOT NULL,**

**CONSTRAINT pk\_topics\_and\_posts PRIMARY KEY(topic\_id, post\_id),**

**CONSTRAINT fk\_topics\_and\_posts\_topic\_id FOREIGN KEY(topic\_id) REFERENCES topics(id),**

**CONSTRAINT fk\_topics\_and\_posts\_post\_id FOREIGN KEY(post\_id) REFERENCES posts(id)**

**);**

**INSERT INTO topics\_and\_posts**

**( post\_id, topic\_id )**

**VALUES**

**( 1, 1 ),**

**( 2, 2 ),**

**( 3, 2 ),**

**( 3, 3 ),**

**( 4, 2 ),**

**( 4, 4 ),**

**( 5, 2 ),**

**( 5, 4 ),**

**( 6, 2 ),**

**( 6, 4 ),**

**( 7, 3 );**

**SELECT \* FROM topics\_and\_posts;**

***-- Оценки постов пользователями***

**CREATE TABLE post\_marks\_and\_users (**

**post\_id int NOT NULL,**

**user\_id int NOT NULL,**

**mark int NOT NULL,**

**CONSTRAINT pk\_post\_marks\_and\_users PRIMARY KEY(post\_id, user\_id),**

**CONSTRAINT fk\_post\_marks\_and\_users\_post\_id FOREIGN KEY(post\_id) REFERENCES posts(id),**

**CONSTRAINT fk\_post\_marks\_and\_users\_user\_id FOREIGN KEY(user\_id) REFERENCES users(id)**

**);**

**INSERT INTO post\_marks\_and\_users**

**( post\_id, user\_id, mark )**

**VALUES**

**( 1, 2, 5 ),**

**( 1, 4, 3 ),**

**( 1, 7, 2 ),**

**( 1, 8, 4 ),**

**( 2, 1, 5 ),**

**( 2, 2, 1 ),**

**( 2, 4, 2 ),**

**( 2, 5, 4 ),**

**( 3, 8, 3 ),**

**( 3, 9, 2 ),**

**( 3, 10, 1 ),**

**( 3, 5, 4 ),**

**( 3, 4, 4 ),**

**( 4, 9, 3 ),**

**( 4, 2, 2 ),**

**( 4, 4, 2 ),**

**( 4, 3, 5 ),**

**( 4, 11, 5 ),**

**( 4, 12, 5 ),**

**( 4, 13, 3 ),**

**( 5, 4, 4 ),**

**( 5, 3, 5 ),**

**( 5, 11, 4 ),**

**( 5, 12, 3 ),**

**( 5, 13, 2 ),**

**( 6, 2, 4 ),**

**( 6, 5, 4 ),**

**( 6, 3, 4 ),**

**( 6, 11, 2 ),**

**( 6, 12, 3 ),**

**( 6, 13, 3 ),**

**( 7, 2, 2 ),**

**( 7, 3, 4 ),**

**( 7, 8, 3 ),**

**( 7, 7, 3 ),**

**( 7, 1, 3 );**

**SELECT \* FROM post\_marks\_and\_users;**

***-- Комментарии пользователей к постам***

**CREATE TABLE post\_comments (**

**id SERIAL NOT NULL,**

**post\_id int NOT NULL,**

**user\_id int NOT NULL,**

**body varchar(1000) NOT NULL,**

**date\_and\_time timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP,**

**CONSTRAINT pk\_post\_comments PRIMARY KEY(id),**

**CONSTRAINT fk\_post\_comments\_post\_id FOREIGN KEY(post\_id) REFERENCES posts(id),**

**CONSTRAINT fk\_post\_comments\_user\_id FOREIGN KEY(user\_id) REFERENCES users(id)**

**);**

**INSERT INTO post\_comments**

**( post\_id, user\_id, body ) VALUES**

**( 1, 2, 'Это потрясающе! Никогда не был равнодушен к физике!' ),**

**( 1, 5, 'Эта статья вдохновила меня на запись нового альбома!' ),**

**( 6, 4, 'Скоро сам буду учить друзей играть на гитаре!' ),**

**( 2, 10, 'Бесплатно рассылаю семена растений, для того, чтобы их получить, ввойдите в свой личный кабинет на сайте http://accStealing.com' );**

**SELECT \* FROM post\_comments;**

1. **Исходный код для создания триггеров:**

***-- Цель триггера -- записывать в журнал***

***-- изменения фамилий пользователей***

***-- Создадим журнал изменений фамилий***

**DROP TABLE IF EXISTS user\_audits;**

**CREATE TABLE user\_audits (**

**id int GENERATED ALWAYS AS IDENTITY,**

**user\_id int NOT NULL,**

**last\_name varchar(30) NOT NULL,**

**changed\_on timestamp(6) NOT NULL**

**);**

***-- Создадим функцию для триггера,***

***-- которая будет делать запись в журнал,***

***-- если у пользователя меняется фамилия***

**CREATE OR REPLACE FUNCTION log\_last\_name\_changes()**

**RETURNS TRIGGER**

**LANGUAGE PLPGSQL**

**AS**

**$$**

**BEGIN**

**IF NEW.last\_name <> OLD.last\_name THEN**

**INSERT INTO user\_audits(user\_id, last\_name, changed\_on)**

**VALUES(OLD.id, OLD.last\_name, now());**

**END IF;**

**RETURN NEW;**

**END;**

**$$;**

***-- Создадим триггер, который вызывает функцию***

***-- перед обновлением***

**DROP TRIGGER IF EXISTS last\_name\_changes on users;**

**CREATE TRIGGER last\_name\_changes**

**BEFORE UPDATE**

**ON users**

**FOR EACH ROW**

**EXECUTE PROCEDURE log\_last\_name\_changes();**

***-- Проверим результат***

**UPDATE users SET last\_name = 'Овчинников'**

**WHERE nickname = 'Johny19';**

**SELECT \* FROM user\_audits;**

***-- Цель триггера -- проверить корректность удаления аккаунта пользователя,***

***-- по правилам активные аккаунты удалять нельзя***

***-- (аккаунт должен быть предварительно заморожен)***

***-- Создадим функцию, проверяющую статус аккаунта,***

***-- и вызывающую исключение в случае ошибки***

**CREATE OR REPLACE FUNCTION check\_user\_delete()**

**RETURNS TRIGGER**

**LANGUAGE PLPGSQL**

**AS**

**$$**

**BEGIN**

**IF OLD.\_state = B'1' THEN**

**RAISE EXCEPTION 'Нельзя удалять активный аккаунт!';**

**END IF;**

**END;**

**$$;**

***-- Создадим триггер, который вызывает функцию***

***-- перед удалением в таблице с пользователями***

**DROP TRIGGER IF EXISTS user\_delete\_checker on users;**

**CREATE TRIGGER user\_delete\_checker**

**BEFORE DELETE**

**ON users**

**FOR EACH ROW**

**EXECUTE PROCEDURE check\_user\_delete();**

***-- Проверим результат***

**SELECT first\_name, nickname, \_state FROM users**

**WHERE nickname = 'Johny19';**

**DELETE FROM users**

**WHERE nickname = 'Johny19';**

**SELECT first\_name, nickname, \_state FROM users**

**WHERE nickname = 'Johny19';**

1. **Исходный код для создания процедур:**

***-- Процедура, позволяющая пересылать средства***

***-- от одного аккаунта в другой***

***-- Повысить уровень привилегий аккаунта***

**CREATE OR REPLACE PROCEDURE increase\_level (**

**user\_id int**

**)**

**LANGUAGE PLPGSQL**

**AS**

**$$**

**DECLARE**

**user\_level int;**

**BEGIN**

***-- Сначала проверим текущий уровень привилегий***

**user\_level := (SELECT \_level FROM users WHERE id = user\_id);**

**IF user\_level = 3 THEN**

**RAISE EXCEPTION 'Пользователь имеет максимальный уровень привилегий!';**

**END IF;**

***-- Если все ОК, то повышаем уровень***

**UPDATE users SET \_level = \_level + 1**

**WHERE id = user\_id;**

**COMMIT;**

**END;**

**$$;**

***-- Проверим результат***

**SELECT id, nickname, \_level FROM users**

**WHERE id = 13;**

**CALL increase\_level(13);**

**SELECT id, nickname, \_level FROM users**

**WHERE id = 13;**

**CALL increase\_level(13);**

**SELECT id, nickname, \_level FROM users**

**WHERE id = 13;**

***-- Процедура, позволяющая пересылать средства***

***-- от одного аккаунта в другой***

**CREATE OR REPLACE PROCEDURE send\_credits (**

**receiver\_id int,**

**sender\_id int,**

**amount int**

**)**

**LANGUAGE PLPGSQL**

**AS**

**$$**

**BEGIN**

***-- забираем количество пересылаемых средств***

***-- у аккаунта отправителя***

**UPDATE users**

**SET balance = balance - amount**

**WHERE id = sender\_id;**

***-- начисляем средства на аккаунт получателя***

**UPDATE users**

**SET balance = balance + amount**

**WHERE id = receiver\_id;**

**COMMIT;**

**END;**

**$$;**

***-- Проверим результат***

**SELECT id, nickname, balance FROM users**

**WHERE id = 2 OR id = 5;**

**CALL send\_credits(2, 5, 100);**

**SELECT id, nickname, balance FROM users**

**WHERE id = 2 OR id = 5;**