

Факультет программной инженерии и компьютерной техники

Дисциплина «Информационный системы и базы данных»

**Курсовая работа**

Выполнили

Громилова М. Д.

Мозговая Л.А.

P33311

Преподаватель

Николаев В. В

Санкт-Петербург 2023

# Этап 1

**Описание предметной области**

Приложение для героев вселенной “Все везде и сразу”

Злодейка Джобу Тупаки хочет уничтожить мультивселенную. Чтобы это предотвратить герои ежедневно сражаются с ней и ее монстрами. Цель героев - уничтожить дрейдл, который Для противостояния они используют специальные навыки. Каждый человек в зависимости от его скилла может получить ненадолго навык своего персонажа из другой вселенной. Для этого ему нужно совершить одно из 5 абсурдных действий. Чем дальше вселенная, тем сильнее она отличается от вселенной героя.

**Описание и правила**

**Герои** - участники борьбы с Джобу Тупаки. Авторизованные пользователи, которые могут при помощи приложения отслеживать свои навыки в других вселенных, свой скилл, получать задания для извлечения навыка из другой вселенной.

**Скилл** - уровень героя. Скилл повышается в двух случаях: герой убил монстра, герой накопил определенный опыт по перемещению между вселенными. Чем выше скилл, тем дальше по вселенным герой может перемещаться.

**Навык -** в зависимости от вселенной герой может получить навыки, например боевых искусств. Чем дальше в мультивселенной перемещается герой, тем более мощный навык он получает. В случае поражения от монстра, герой теряет навык

**Задания** - это абсурдное действие, которое должен совершить герой. Дается на выбор 5 заданий. В зависимости от выбора, герой перемещается в нужную вселенную.

**Монстр -** приспешник Джобы Тупаки. Чтобы подобраться к главной злодейке нужно уничтожить попадающихся монстров, разной силы.

**Сражение -** в рандомный момент в приложении появляется информация о нападении монстра. Тогда герою может вступить с ним в поединок. Если герой выиграл, он получает дополнительный скилл за убийство. А координатор, который ему помогал получает деньги. Дальше у героя идет процесс регенерации, чем дальше он перемешался по вселенной, тем дольше.

**Бизнес-процессы**

**Герой -** человек, который убивает монстров, чтобы повысить скилл и убить главную злодейку

**Координатор** - человек, который помогает герою в сражении за деньги. Герой и координатор могут списаться и договориться о сотрудничестве. Координатор помогает с выбором навыка, перемещениями и ведением боя.

**Обычный пользователь** любой человек, может стать героем или может просто сообщать о монстрах

**Технический специалист** следит за корректной работой приложения

**Сотрудник колл-центра** принимает информацию от людей о монстрах

**Админ -** начальник, принимает решения о ставке за убийство, зарплат координаторов и других сотрудников

**1.Управление(улучшение) навыками и компетенциями**

**1. Отслеживание навыков**

Сейчас во вселенной есть приложение, оно анализирует жизнь человека, выявляет те сферы, которым он начинал заниматься, исходя из этого строит модель мультивселенной. По этой модели можно узнать в какую вселенную нужно переместиться, чтобы получить навык и какой навык.

Каждый человек может зарегистрироваться в приложении, пройти опрос. Анкета парсится и создается модель вселенных с возможными навыками. По правилам вселенной те навыки, которыми ты пытался безуспешно овладеть в своей вселенной, появляются у твоего персонажа в другой вселенной. Чем хуже этот навык в твоей вселенной, тем лучше он в параллельной. Получаем набор навыков, чем лучше навык, тем он дальше от твоей вселенной. Исходя из этого и строится модель.

1. Обычный пользователь регистрируется в приложении (выбирает кем он хочет быть герой/координатор/тех.спец), в зависимости от выбора проходит опрос.
2. Анкета парсится и создается модель вселенных с возможными навыками навыками.
3. Пользователь получает аккаунт, может запросить модель своих навыков.
4. Этим аккаунтом он в дальнейшем пользуется. он может вести борьбу, выбрав заказ на убийство монстра

**2. Выдача заданий**

Герою дается на выбор 5 рандомных заданий. В зависимости от выбора герой перемещается в нужную вселенную.

0. Технический специалист заносит в базу данных абсурдные задания и раз в месяц обновляет их.

1. Герой хочет переместиться, чтобы получить определенный навык, он делает запрос к приложению о заданиях
2. Приложение дает на выбор 5 абсурдных заданий. Оно генерирует их из готовой бд.
3. Герой выбирает задание
4. Если герой выполняет его, то с помощью магии перемещается в нужную вселенную.

**3. Отслеживание скилла**

Скилл повышается в двух случаях: герой убил монстра, герой накопил определенный опыт по перемещению между вселенными. Чем выше скилл, тем дальше по вселенным герой может без вреда здоровью перемещаться. Чем дальше в мультивселенной перемещается герой, тем более мощный навык он получает. В случае поражения от монстра, герой теряет навык и возможность им завладеть в дальнейшем. Эту информацию приложение анализирует.

1. Герой делает запрос о скилле к приложению
2. Приложение обрабатывает запрос и выдает скилл героя
3. Если герой убил монстра, эти данные проверяются и его скилл повышается (у каждого монстра есть уровень который прибавляется к скиллу героя)
4. Если герой переместился в другую вселенную и получил навык, его скилл увеличивается.

**2. Система поиска и борьбы с монстрами**

\*Этой функции сейчас нет, герои сами ищут монстров.

Мы сделаем так, что герой сможет выбирать задание на убийство монстра, а так же он может объединиться с другим героем или координатором для помощи в бою. (Условная доска заданий)

1. Обычный пользователь приложения может видеть на сайте телефон горячей линии.
2. Если человек видит монстра он может сообщить о его местоположении работнику колл-центра.
3. Работник определяет тип монстра заполняет информацию о происшествии, его адресе в специальной форме в приложении, отправляет ее.
4. Информация добавляется на доску заказов.
5. Герой
6. имеет возможность в своем аккаунте перейти на доску заказов.
7. может посмотреть и выбрать подходящего для себя монстра
8. может взять заказ с доски заказов вместе к каким то человеком
9. после этого может посмотреть подробную информацию об адресе монстра
10. может взять заказ с доски заказов, тогда он скрывается от других героев.
11. Герой выбирает себе помощника (героя или координатора), из тех, что у него уже в “друзьях”. (подробнее в *коммуникациях сотрудников*)
12. Координатор
13. Принимает заявку от героя (или они могут быть уже в друзьях
14. Они списываются в приложении и обо всем договариваются
15. Вместе вступают в бой (приложение за это не отвечает)
16. Координатор помогает с выбором навыка, перемещениями и ведением боя

7. Отчет о выполнении задания:

1. После выполнения задания, координатор звонит в колл-центр и сообщает о выполненном задании
2. Сотрудник колл-центра сообщает сотрудникам контроля качества, о том что нужно проверить качество выполнения задания
3. Сотрудник прибывает на место боя, фиксирует результат и заполняет форму (результат, аккаунты координатора и героя).
4. Технический специалист заносит данные в приложение. Если задание выполнено герою начисляется скилл, а его помощник может позвонить в отдел и договорить о получении гонорара

**3. Коммуникация сотрудников**

\*Этой функции сейчас нет, люди сами пытаются объединиться в группы, созваниваются и помогают друг другу.

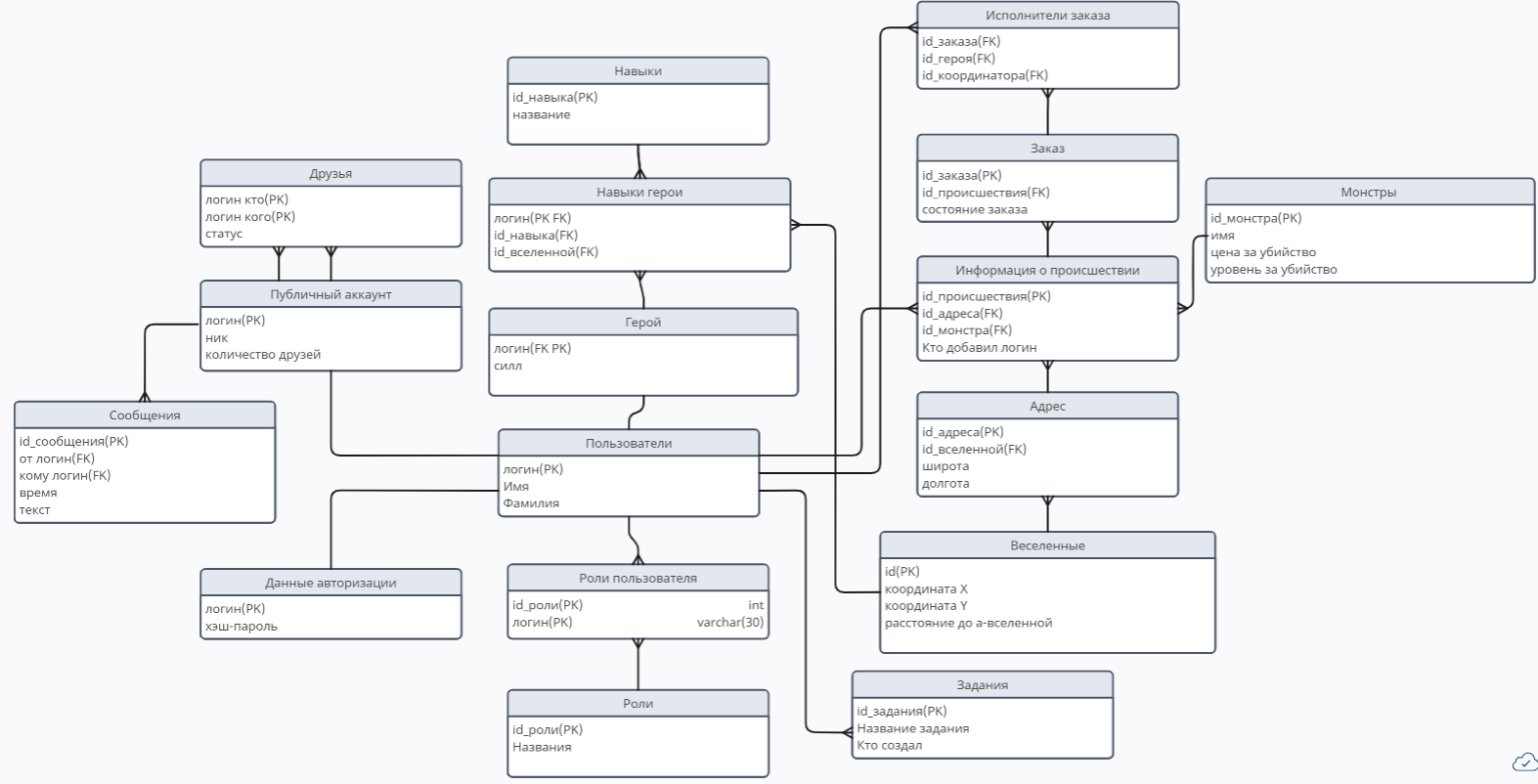
Мы поможем им найти единомышленников и эффективно бороться с монстрами, зарабатывая деньги.

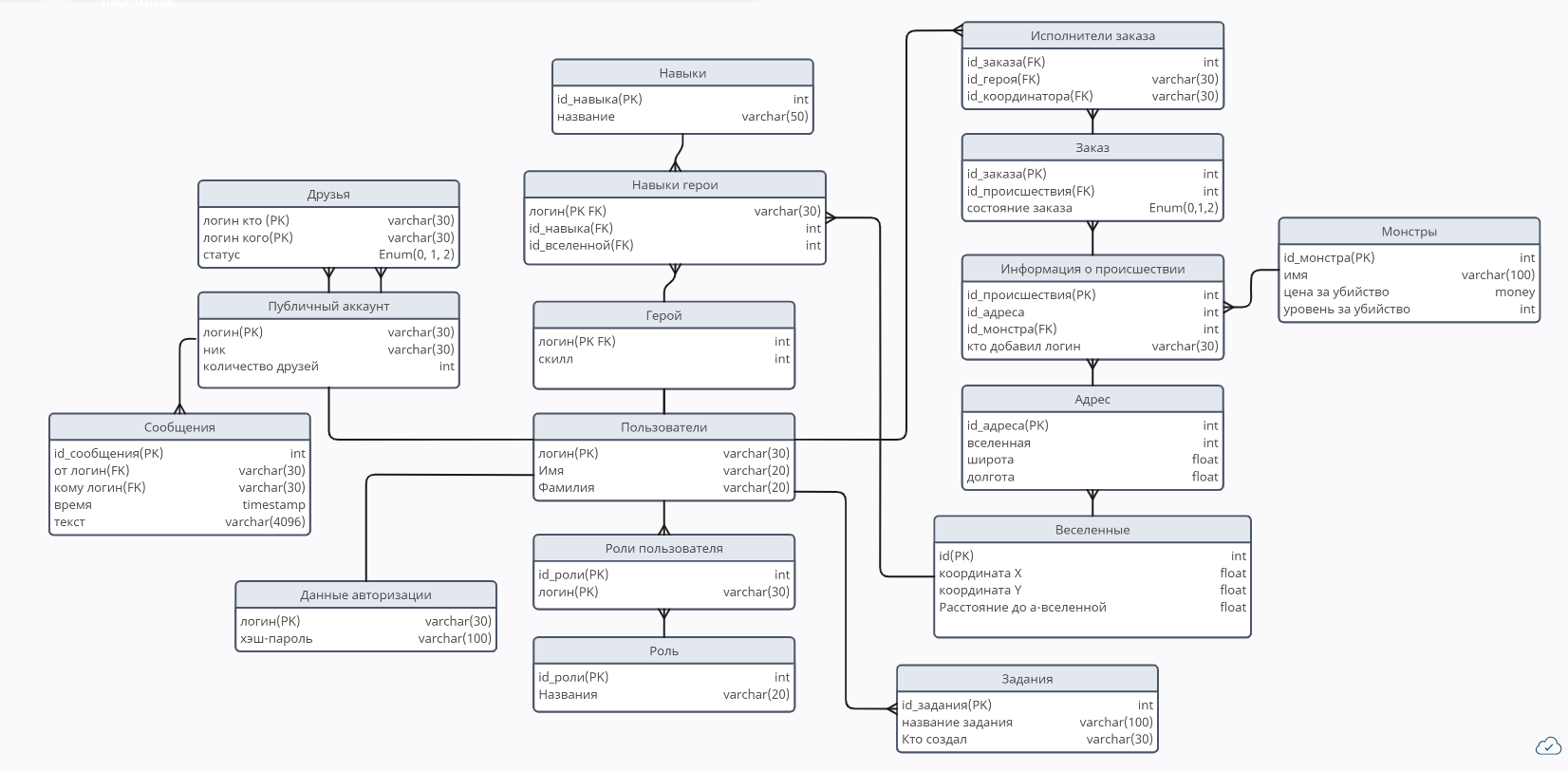
1. У каждого героя и координатора есть публичные аккаунты
2. В аккаунт можно добавить личную информацию (ник, скилл)
3. Раздел со всеми аккаунтами:
4. Можно в поиске найти и добавить пользователей в друзья.
5. Пользователи могут переписываться
6. Раздел с диалогами:
7. Можно создать новый диалог со своим другом или зайти в старый.
8. В диалоге можно написать сообщение или обменяться номерами телефонов
9. В диалоге хранятся старые сообщения

**Вывод**

Основная цель нашей базы данных - помогать героям предотвратить уничтожение мультивселенной. Получать информацию о других вселенных, возможных навыках, нужных заданиях, а также отслеживать свой уровень. Получать предупреждения о нападении монстра и сражаться с ним.

# Этап 2





Триггеры:

1. Нельзя добавить дубликат пары друзей в таблицу. BEFORE INSERT ON friends FOR EACH ROW
2. В таблице исполнители заказа герой и координатор должны быть разными людьми, также герой и координатор это люди, имеющие эти роли. BEFORE INSERT OR UPDATE ON employers FOR EACH ROW
3. При добавлении роли “Герой” добавляется новое значение в таблицу героев. При добавлении роли “Герой” или “Координатор” добавляется новое значение в таблицу публичный аккаунт. AFTER INSERT ON user\_role FOR EACH ROW
4. Информацию об инциденте может менять только технический специалист. AFTER INSERT ON incident\_information FOR EACH ROW
5. Задания может добавлять только Технический специалист. AFTER INSERT ON quest FOR EACH ROW
6. Если статус заказа меняется и становится успешно выполненным, герою прибавляется скилл по тому монстру, которого он убил. AFTER UPDATE ON orders FOR EACH ROW
7. После добавления нового героя добавляется информация в таблицу навыки героя

# Этап 3

## Создание таблиц

CREATE TABLE monsters

(

id SERIAL PRIMARY KEY,

name VARCHAR(100) NOT NULL,

price MONEY NOT NULL,

level INTEGER NOT NULL,

CHECK (price::numeric BETWEEN 1.00 AND 999.99 AND level BETWEEN 1 AND 9)

);

CREATE TABLE universals

(

id SERIAL PRIMARY KEY,

coordinate\_x float not null,

coordinate\_y float not null,

distance float not null

CHECK (coordinate\_x >= -100 and coordinate\_x <= 100 and coordinate\_y >= -100 and coordinate\_y <= 100)

);

CREATE TABLE address

(

id SERIAL PRIMARY KEY,

universals\_id INTEGER REFERENCES universals (id) ON DELETE CASCADE,

longitude float not null,

latitude float not null

);

CREATE TABLE users

(

login VARCHAR(30) PRIMARY KEY,

name VARCHAR(20) NOT NULL,

surname VARCHAR(20) NOT NULL

);

CREATE TABLE authorization\_data

(

user\_login VARCHAR(30) REFERENCES users (login) ON DELETE CASCADE,

password VARCHAR(100) NOT NULL,

salt VARCHAR(10) NOT NULL,

PRIMARY KEY (user\_login)

);

CREATE TYPE role\_name AS ENUM ('Герой', 'Координатор', 'Технический специалист', 'Админ');

CREATE TABLE role

(

id SERIAL PRIMARY KEY,

name role\_name

);

CREATE TABLE user\_role

(

role\_id INTEGER REFERENCES role (id) ON DELETE CASCADE,

login varchar(30) REFERENCES users (login) ON DELETE CASCADE,

PRIMARY KEY (role\_id, login)

);

CREATE TABLE account

(

login varchar(30) REFERENCES users (login) ON DELETE CASCADE,

nickname varchar(30),

friends integer DEFAULT (0),

PRIMARY KEY (login)

);

CREATE TABLE messages

(

id SERIAL PRIMARY KEY,

login1 varchar(30) REFERENCES users (login) ON DELETE SET NULL,

login2 varchar(30) REFERENCES users (login) ON DELETE SET NULL,

time TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP*,

content varchar(4096)

);

CREATE TYPE status\_enum AS ENUM ('0', '1', '2');

CREATE TABLE friends

(

login1 varchar(30) REFERENCES users (login) ON DELETE SET NULL,

login2 varchar(30) REFERENCES users (login) ON DELETE SET NULL,

status status\_enum,

PRIMARY KEY (login1, login2)

);

CREATE TABLE hero

(

login varchar(30) REFERENCES users (login) ON DELETE CASCADE,

skill integer default (0),

PRIMARY KEY (login)

);

CREATE TABlE ability

(

id SERIAL PRIMARY KEY,

title varchar(50) not null

);

CREATE TABlE hero\_ability

(

login varchar(30) REFERENCES users (login) ON DELETE CASCADE,

ability\_id INTEGER REFERENCES ability (id) ON DELETE CASCADE,

universal\_id INTEGER REFERENCES universals (id) ON DELETE CASCADE,

PRIMARY KEY (login, ability\_id)

);

CREATE TABlE quest

(

id SERIAL PRIMARY KEY,

name varchar(100),

login varchar(30) REFERENCES users (login) ON DELETE SET NULL

);

CREATE TABLE incident\_information

(

id SERIAL PRIMARY KEY,

address\_id INTEGER REFERENCES address (id) ON DELETE SET NULL,

monster\_id INTEGER REFERENCES monsters (id) ON DELETE SET NULL,

login varchar(30) REFERENCES users (login) ON DELETE SET NULL

);

CREATE TYPE order\_status AS ENUM ('0', '1', '2');

CREATE TABLE orders

(

id SERIAL PRIMARY KEY,

incident\_id INTEGER REFERENCES incident\_information (id) ON DELETE CASCADE,

status order\_status

);

CREATE TABLE employers

(

order\_id INTEGER REFERENCES orders (id) ON DELETE SET NULL,

hero varchar(30) REFERENCES users (login) ON DELETE SET NULL,

coordinator varchar(30) REFERENCES users (login) ON DELETE SET NULL,

PRIMARY KEY (order\_id, hero, coordinator)

);

## Добавление триггеров

1. Нельзя добавить дубликат пары друзей в таблицу

CREATE OR REPLACE FUNCTION *check\_friends\_unique*()

RETURNS TRIGGER AS

$$

BEGIN

IF *EXISTS*(

SELECT 1

FROM friends

WHERE friends.login1 = NEW.login1

AND friends.login2 = NEW.login2

OR friends.login1 = NEW.login2

AND friends.login2 = NEW.login1

) THEN

RAISE EXCEPTION 'Combination of friends must be unique';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_friends\_unique

BEFORE INSERT

ON friends

FOR EACH ROW

EXECUTE FUNCTION *check\_friends\_unique*();

1. В таблице исполнители заказа герой и координатор разные люди, также герой и координатор это люди, имеющие эту роль

CREATE OR REPLACE FUNCTION *check\_hero\_coordinator*()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.hero = NEW.coordinator THEN

RAISE EXCEPTION 'hero and coordinator cannot have the same value';

END IF;

IF *EXISTS*(

SELECT 1

FROM users inner join user\_role on users.login = user\_role.username

inner join role on role.id = user\_role.role\_id

WHERE (NEW.hero = users.login AND role.name !='Герой') OR

(NEW.coordinator = users.login AND (role.name!= 'Герой' or role.name !='Координатор'))

)THEN

RAISE EXCEPTION 'It is not hero or coordinator';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_hero\_coordinator\_trigger

BEFORE INSERT OR UPDATE

ON employer

FOR EACH ROW

EXECUTE FUNCTION *check\_hero\_coordinator*();

1. При добавлении роли “Герой” добавляется новое значение в таблицу героев. При добавлении роли “Герой” или “Координатор” добавляется новое значение в таблицу публичный аккаунт.

CREATE OR REPLACE FUNCTION *add\_hero\_role*()

RETURNS TRIGGER AS $$

BEGIN

IF *EXISTS*(

SELECT 1

FROM role

WHERE NEW.role\_id = role.id

AND role.name = 'Герой'

) THEN

INSERT INTO hero(login) VALUES (NEW.username);

END IF;

IF *EXISTS*(

SELECT 1

FROM role

WHERE NEW.role\_id = role.id

AND (role.name = 'Герой' OR role.name = 'Координатор')

) THEN

INSERT INTO account(login) VALUES (NEW.username);

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER add\_hero\_role\_trigger

AFTER INSERT

ON user\_role

FOR EACH ROW

EXECUTE FUNCTION *add\_hero\_role*();

1. Информацию об инциденте может менять только технический специалист

CREATE OR REPLACE FUNCTION *check\_role\_technical\_specialist*()

RETURNS TRIGGER AS $$

BEGIN

IF *EXISTS*(

SELECT 1

FROM user\_role inner join role ON role.id = user\_role.role\_id

WHERE NEW.login = user\_role.username

AND role.name != 'Технический\_специалист'

) THEN

RAISE EXCEPTION 'employer must be technical specialist';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_role\_incident

AFTER INSERT

ON incident\_information

FOR EACH ROW

EXECUTE FUNCTION *check\_role\_technical\_specialist*();

1. Задания может добавлять только Технический специалист

CREATE TRIGGER check\_role\_quest

AFTER INSERT

ON quest

FOR EACH ROW

EXECUTE FUNCTION *check\_role\_technical\_specialist*();

1. Если статус заказа меняется и становится успешно выполненным, герою прибавляется скилл по тому монстру, которого он убил

CREATE OR REPLACE FUNCTION *add\_points*()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.status = '2' THEN

UPDATE hero

SET skill = hero.skill + monsters.level

FROM users

INNER JOIN hero AS hero\_user ON users.login = hero\_user.login

INNER JOIN employer ON users.login = employer.hero

INNER JOIN orders ON employer.order\_id = orders.id

INNER JOIN incident\_information ON orders.incident\_id = incident\_information.id

INNER JOIN monsters ON incident\_information.monster\_id = monsters.id

WHERE incident\_id = NEW.incident\_id and hero.login = employer.hero and employer.order\_id = orders.id;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER add\_points

AFTER UPDATE

ON orders

FOR EACH ROW

EXECUTE FUNCTION *add\_points*();

1. После добавления нового героя добавляется информация в таблицу навыки героя

CREATE OR REPLACE FUNCTION *add\_abilities\_hero*()

RETURNS TRIGGER AS $$

BEGIN

INSERT INTO hero\_ability (ability\_id, login, mastery\_percentage)

SELECT DISTINCT a.id, NEW.login, 0

FROM ability a;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER add\_abilities\_hero

AFTER INSERT ON hero

FOR EACH ROW

EXECUTE FUNCTION *add\_abilities\_hero*();

## Добавление данных

INSERT INTO users(login, name, surname) VALUES

('user1', 'Иван', 'Иванов'),

('user2', 'Мария', 'Петрова'),

('user3', 'Алексей', 'Сидоров'),

('user4', 'Екатерина', 'Ковалева'),

('user5', 'Павел', 'Николаев'),

('user6', 'Анна', 'Смирнова'),

('user7', 'Дмитрий', 'Попов'),

('user8', 'Ольга', 'Морозова'),

('user9', 'Сергей', 'Кузнецов'),

('user10', 'Наталья', 'Васнецова');

INSERT INTO role(name) VALUES

('Координатор'),

('Герой'),

('Технический специалист'),

('Админ');

INSERT INTO user\_role(role\_id, login) VALUES

(8,'user1'),

(9,'user2'),

(10,'user3'),

(11, 'user4'),

(8, 'user5'),

(9, 'user6'),

(10,'user7'),

(8, 'user8'),

(9, 'user9'),

(9, 'user10');

INSERT INTO ability(title) VALUES

('карате'),

('танцы'),

('плавание'),

('футбол'),

('гитара'),

('живопись'),

('бег'),

('пение'),

('фотография'),

('шахматы'),

('волейбол'),

('кулинария'),

('фриланс'),

('программирование'),

('языки'),

('гольф'),

('стрельба из лука'),

('акробатика'),

('керамика'),

('баскетбол');

INSERT INTO monsters (name, price, level) VALUES

('Гоблин', 10.50, 3),

('Вампир', 150.75, 7),

('Орк', 25.00, 4),

('Медуза', 80.20, 6),

('Кентавр', 45.90, 5),

('Грифон', 200.00, 8),

('Банши', 75.30, 6),

('Харпия', 30.60, 4),

('Циклоп', 90.40, 7),

('Бес', 15.25, 2);

INSERT INTO universals (coordinate\_x, coordinate\_y, distance) VALUES

(10.5, 20.3, 30.7),

(-5.2, -8.9, 15.4),

(0.0, 0.0, 5.6),

(50.1, 75.6, 90.2),

(-30.8, 40.2, 60.0),

(15.0, -25.7, 35.8),

(70.3, -80.5, 110.1),

(-90.0, 90.0, 120.3),

(100.0, -100.0, 150.0),

(5.5, -10.0, 12.5);

INSERT INTO address (universals\_id, longitude, latitude) VALUES

(1, 30.5, 40.3),

(2, -15.2, -18.9),

(3, 0.0, 0.0),

(4, 60.1, 75.6),

(5, -40.8, 50.2),

(6, 25.0, -35.7),

(7, 80.3, -90.5),

(8, -90.0, 80.0),

(9, 100.0, -90.0),

(10, 15.5, -20.0);

INSERT INTO messages (login1, login2, content) VALUES

('user1', 'user2', 'Привет, как дела?'),

('user2', 'user1', 'Привет! Всё хорошо, спасибо. Как у тебя?'),

('user3', 'user1', 'Привет! Ты сегодня видел новость о...'),

('user4', 'user2', 'Этот фильм просто потрясающий!'),

('user5', 'user3', 'Когда у нас следующая встреча по проекту?'),

('user1', 'user3', 'Я сегодня заметил интересную статью по теме, о которой мы говорили.'),

('user4', 'user1', 'Спасибо за рекомендацию, обязательно посмотрю!'),

('user2', 'user3', 'Да, я видел эту новость. Шокирующие подробности!'),

('user3', 'user5', 'Давайте встретимся завтра и обсудим все детали.'),

('user5', 'user4', 'Твоя рекомендация насчет книги была отличной!');

INSERT INTO friends (login1, login2, status) VALUES

('user1', 'user2','0' ),

('user2', 'user3', '1'),

('user3', 'user1', '2'),

('user4', 'user2', '2'),

('user5', 'user1', '1');

INSERT INTO hero\_ability (login, ability\_id, universal\_id) VALUES

('user1', 1, 1),

('user1', 2, 2),

('user2', 3, 3),

('user3', 4, 4),

('user3', 1, 5),

('user4', 2, 6),

('user5', 3, 7),

('user5', 4, 8),

('user5', 1, 9),

('user6', 2, 10);

INSERT INTO incident\_information (address\_id, monster\_id, login) VALUES

(1, 3, 'user3'),

(2, 5, 'user3'),

(3, 2, 'user3'),

(4, 7, 'user3'),

(5, 1, 'user3'),

(6, 8, 'user3'),

(7, 4, 'user7'),

(8, 6, 'user3'),

(9, 10, 'user7'),

(10, 9, 'user7');

INSERT INTO orders (incident\_id, status) VALUES

(11, '0'),

(12, '1'),

(13, '2'),

(14, '0'),

(15, '1'),

(16, '2'),

(17, '0'),

(18, '1'),

(19, '2'),

(20, '0');

INSERT INTO employers (order\_id, hero, coordinator) VALUES

(11, 'user2', 'user1'),

(12, 'user2', 'user5'),

(13, 'user6', 'user1'),

(14, 'user6', 'user5'),

(15, 'user9', 'user1'),

(16, 'user9', 'user8'),

(17, 'user2', 'user1'),

(18, 'user2', 'user5'),

(19, 'user9', 'user1'),

(20, 'user6', 'user5');

## Наиболее частые запросы к базе и индексы

EXPLAIN ANALYZE select \* from messages where login1='user1' and login2 = 'user3';

create index login\_who on messages using hash(login1);

create index login\_whom on messages using hash(login2);

EXPLAIN ANALYZE select \* from friends where login1='user1';

create index login\_friend1 on friends using hash(login1);

create index login\_friend2 on friends using hash(login2);

select \* from account where login='user1';

## Функции и процедуры

Функция получения сообщений для диалога

CREATE OR REPLACE FUNCTION *get\_messages*(login\_who varchar(30), login\_whom varchar(30))

RETURNS TABLE (

name varchar(30),

content varchar(4096),

time1 timestamp

) AS $$

BEGIN

RETURN QUERY

SELECT

CASE

WHEN message.login1 = login\_who THEN message.login1::varchar(30)

WHEN message.login2 = login\_who THEN message.login1::varchar(30)

END,

message.content, message.time

FROM

message

WHERE

(message.login1 = login\_who AND message.login2 = login\_whom)

OR (message.login2 = login\_who AND message.login1 = login\_whom)

ORDER BY

message.time;

END;

$$ LANGUAGE plpgsql;

Вывод всех друзей

CREATE OR REPLACE FUNCTION *get\_friends*(person varchar(30),state varchar(30))

RETURNS TABLE (

friend varchar(30)

) AS $$

BEGIN

RETURN QUERY

SELECT

CASE

WHEN friends.login1 = person THEN friends.login2::varchar(30)

WHEN friends.login2 = person THEN friends.login1::varchar(30)

END

FROM

friends

WHERE

(friends.login1 = person or friends.login2 = person)

and friends.status = state;

END

$$ LANGUAGE plpgsql;

SELECT \* FROM *get\_friends*('lara03','REQUEST\_ACCEPTED');

Вывод всех друзей или координаторов, который нет в друзьях у юзера

CREATE OR REPLACE FUNCTION *get\_people*(person varchar(30))

RETURNS TABLE (

friend varchar(30)

) AS $$

BEGIN

RETURN QUERY

SELECT

u.login

FROM users u

INNER JOIN user\_role ur on u.login = ur.username

inner join role r on ur.role\_id = r.id

WHERE u.login!= person and (r.name = 'Герой' or r.name = 'Координатор')

and u.login not IN (SELECT \* FROM *get\_friends*(person,'REQUEST\_ACCEPTED'));

END

$$ LANGUAGE plpgsql;

**Вывести данные на доску заказов**

**CREATE OR REPLACE FUNCTION *get\_order*()**

**RETURNS TABLE (**

**id integer,**

**name varchar(100),**

**level integer,**

**price double precision**

**) AS $$**

**BEGIN**

**RETURN QUERY**

**SELECT**

**orders.id,monsters.name, monsters.level, monsters.price**

**FROM**

**orders**

**INNER JOIN incident\_information on orders.incident\_id = incident\_information.id**

**INNER JOIN monsters on incident\_information.monster\_id = monsters.id**

**WHERE orders.status = 'WAITING';**

**END;**

**$$ LANGUAGE plpgsql;**

**Вывести все данные о заказе**

**CREATE OR REPLACE FUNCTION *get\_info\_about\_order*(idOrder integer)**

**RETURNS TABLE (**

**name varchar(100),**

**coordinatex double precision,**

**coordinatey double precision,**

**longitude double precision,**

**latitude double precision**

**) AS $$**

**BEGIN**

**RETURN QUERY**

**SELECT**

**universal.name, *CAST*(universal.coordinatex AS double precision),**

***CAST*(universal.coordinatey AS double precision),**

***CAST*(address.longitude AS double precision),**

***CAST*(address.latitude AS double precision)**

**FROM**

**orders**

**INNER JOIN incident\_information on orders.incident\_id = incident\_information.id**

**INNER JOIN address on incident\_information.address\_id = address.id**

**INNER JOIN universal on address.universal\_id = universal.universal\_id**

**WHERE orders.id = idOrder;**

**END;**

**$$ LANGUAGE plpgsql;**

**Получить модель навыков героя**

**CREATE OR REPLACE FUNCTION *get\_user\_skill\_model*(loginUser varchar(100))**

**RETURNS TABLE (**

**title varchar(100),**

**mastery\_percentage integer,**

**universal varchar(100)**

**) AS $$**

**BEGIN**

**RETURN QUERY**

**SELECT**

**ability.title, hero\_ability.mastery\_percentage, universal.name**

**FROM**

**hero\_ability**

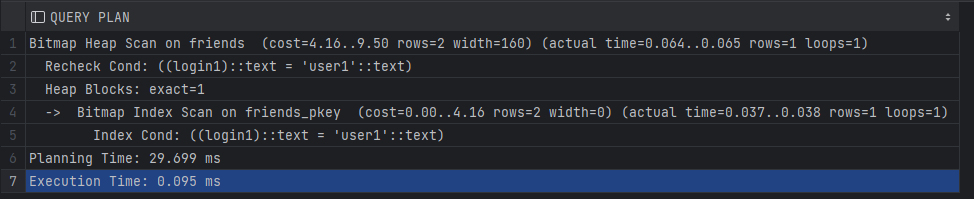
**INNER JOIN ability on hero\_ability.ability\_id = ability.id**

**INNER JOIN universal on hero\_ability.universal\_id = universal.universal\_id**

**WHERE hero\_ability.login = loginUser and hero\_ability.universal\_id is not null;**

**END;**

**$$ LANGUAGE plpgsql;**

****

