Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное
образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

ОТЧЕТ по лабораторной работе
«Этап 2. Начало работы»
по дисциплине «**Проектирование баз данных**»

Автор:	Константинова Елизавета Анатольевна
Факультет:	ФИТиП
Группа:	М3203
Преподаватели:	Шевчик Софья Владимировна

ИТМО

Санкт-Петербург 2024

Этап 2. Начало работы

Задачи: развернуть СУБД, создание таблиц и заполнение данными.

На втором этапе необходимо развернуть СУБД PostgreSQL на сервере или локальной машине внутри docker контейнера:

docker-compose.yml, с помощью которого развертывается бд

```
version: '3.8'
services:
  db:
    build: .
    restart: always
    env_file:
      - .env
    environment:
        POSTGRES_DB: ${POSTGRES_DB}
        POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
        DB_USERNAME: ${DB_USERNAME}
        DB_SUPERUSER: ${DB_SUPERUSER}
        DB_VERSION: ${DB_VERSION}
        FILLING_AMOUNT: ${FILLING_AMOUNT}
    ports:
      - "5432:5432"
    volumes:
      - ./migrations:/migrations
      - ./bashScripts:/bashScript
      - ./bashScripts/entrypoint.sh:/docker-entrypoint-initdb.d/entrypoint.sh
      - ./bashScripts/start_migrations.sh:/docker-entrypoint-
initdb.d/start_migrations.sh
```

.env:

```
POSTGRES_DB="discord-db"
POSTGRES_PASSWORD="mypassword"
DB_USERNAME="postgres"
DB_SUPERUSER="postgres"
FILLING_AMOUNT=100000
```

Идемпотентный файл миграции:

```
CREATE TABLE IF NOT EXISTS users (
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    email VARCHAR(100) NOT NULL UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    online_status BOOLEAN,
    registration_date TIMESTAMP
    );

INSERT INTO  users (username, email, password_hash, online_status,
registration_date)
SELECT
    'user' || generate_series,
    'user' || generate_series || '@example.com',
```

```
    md5(random()::text),
    random() < 0.5,
    NOW() - (random() * INTERVAL '365 days')
FROM
    generate_series(1, ${FILLING_AMOUNT});

CREATE TABLE IF NOT EXISTS  servers (
    server_id SERIAL PRIMARY KEY,
    server_name VARCHAR NOT NULL,
    owner_id INTEGER NOT NULL REFERENCES users(user_id),
    creation_date TIMESTAMP NOT NULL
    );

INSERT INTO servers (server_name, owner_id, creation_date)
SELECT
        'server' || generate_series,
        FLOOR(RANDOM() * ${FILLING_AMOUNT}) + 1,
        NOW() - (random() * INTERVAL '365 days')
FROM
    generate_series(1, ${FILLING_AMOUNT});

CREATE TABLE IF NOT EXISTS  channels (
    channel_id SERIAL PRIMARY KEY,
    channel_name VARCHAR NOT NULL,
    server_id INTEGER NOT NULL REFERENCES servers(server_id),
    channel_type VARCHAR NOT NULL
    );

INSERT INTO channels (channel_name, server_id, channel_type)
SELECT
        'channel' || generate_series,
        FLOOR(RANDOM() * ${FILLING_AMOUNT}) + 1,
        CASE WHEN random() < 0.5 THEN 'text_channel' ELSE 'voice_channel' END
FROM
    generate_series(1, ${FILLING_AMOUNT});

CREATE TABLE IF NOT EXISTS  roles (
     role_id SERIAL PRIMARY KEY,
     role_name VARCHAR NOT NULL,
     server_id INTEGER NOT NULL REFERENCES servers(server_id)
    );

INSERT INTO roles (role_name, server_id)
SELECT
        'role' || generate_series,
        FLOOR(RANDOM() * ${FILLING_AMOUNT}) + 1
FROM
    generate_series(1, ${FILLING_AMOUNT});

CREATE TABLE IF NOT EXISTS  permissions (
    permission_id SERIAL PRIMARY KEY,
    role_id INTEGER NOT NULL REFERENCES roles(role_id),
    channel_id INTEGER NOT NULL REFERENCES channels(channel_id),
    can_send_messages BOOLEAN,
    can_delete_messages BOOLEAN,
    can_edit_messages BOOLEAN,
    can_create_roles BOOLEAN,
```

```
    can_ban_users BOOLEAN
    );

INSERT INTO permissions (role_id, channel_id, can_send_messages,
can_delete_messages, can_edit_messages, can_create_roles, can_ban_users)
SELECT
    FLOOR(RANDOM() * ${FILLING_AMOUNT}) + 1,
    FLOOR(RANDOM() * ${FILLING_AMOUNT}) + 1,
    random() < 0.5,
    random() < 0.5,
    random() < 0.5,
    random() < 0.5,
    random() < 0.5
FROM
    generate_series(1, ${FILLING_AMOUNT});

CREATE TABLE IF NOT EXISTS  invitations (
    invitation_id SERIAL PRIMARY KEY,
    server_id INTEGER NOT NULL REFERENCES servers(server_id),
    inviter_id INTEGER REFERENCES users(user_id),
    invited_user_id INTEGER NOT NULL REFERENCES users(user_id),
    invitation_date TIMESTAMP
    );

INSERT INTO invitations (server_id, inviter_id, invited_user_id,
invitation_date)
SELECT
    FLOOR(RANDOM() * ${FILLING_AMOUNT}) + 1,
    FLOOR(RANDOM() * ${FILLING_AMOUNT}) + 1,
    FLOOR(RANDOM() * ${FILLING_AMOUNT}) + 1,
    NOW() - (random() * INTERVAL '365 days')
FROM
    generate_series(1, ${FILLING_AMOUNT});

CREATE TABLE IF NOT EXISTS  moderation_logs (
    log_id SERIAL PRIMARY KEY,
    moderator_id INTEGER NOT NULL REFERENCES users(user_id),
    user_id INTEGER REFERENCES users(user_id),
    action VARCHAR NOT NULL,
    reason VARCHAR,
    timestamp TIMESTAMP
    );

INSERT INTO moderation_logs (moderator_id, user_id, action, reason,
timestamp)
SELECT
    FLOOR(RANDOM() * ${FILLING_AMOUNT}) + 1,
    FLOOR(RANDOM() * ${FILLING_AMOUNT}) + 1,
    'action' || generate_series,
    'reason' || generate_series,
    NOW() - (random() * INTERVAL '365 days')
FROM
    generate_series(1, ${FILLING_AMOUNT});

CREATE TABLE IF NOT EXISTS  emojis (
    emoji_id SERIAL PRIMARY KEY,
    emoji_name VARCHAR NOT NULL,
```

```
    emoji_image VARCHAR NOT NULL,
    server_id INTEGER REFERENCES servers(server_id),
    creator_id INTEGER NOT NULL REFERENCES users(user_id),
    creation_date TIMESTAMP
    );

INSERT INTO emojis (emoji_name, emoji_image, server_id, creator_id,
creation_date)
SELECT
        'emoji' || generate_series,
        'https://example.com/emoji/' || generate_series || '.png',
        FLOOR(RANDOM() * ${FILLING_AMOUNT}) + 1,
        FLOOR(RANDOM() * ${FILLING_AMOUNT}) + 1,
        NOW() - (random() * INTERVAL '365 days')
FROM
    generate_series(1, ${FILLING_AMOUNT});

CREATE TABLE IF NOT EXISTS  user_roles (
    user_role_id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES users(user_id),
    role_id INTEGER NOT NULL REFERENCES roles(role_id),
    server_id INTEGER NOT NULL REFERENCES servers(server_id)
    );

INSERT INTO user_roles (user_id, role_id, server_id)
SELECT
    FLOOR(RANDOM() * ${FILLING_AMOUNT}) + 1,
    FLOOR(RANDOM() * ${FILLING_AMOUNT}) + 1,
    FLOOR(RANDOM() * ${FILLING_AMOUNT}) + 1
FROM
    generate_series(1, ${FILLING_AMOUNT});

CREATE TABLE IF NOT EXISTS  user_settings (
    settings_id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES users(user_id),
    theme_preference VARCHAR,
    notification_settings VARCHAR,
    privacy_settings VARCHAR,
    other_preferences VARCHAR,
    language VARCHAR
    );

INSERT INTO user_settings (user_id, theme_preference, notification_settings,
privacy_settings, other_preferences, language)
SELECT
    FLOOR(RANDOM() * ${FILLING_AMOUNT}) + 1,
    CASE WHEN random() < 0.5 THEN 'Light' ELSE 'Dark' END,
    'notification_setting' || generate_series,
    'privacy_setting' || generate_series,
    'other_preference' || generate_series,
    'English'
FROM
    generate_series(1, ${FILLING_AMOUNT});

CREATE TABLE IF NOT EXISTS  bans (
    ban_id SERIAL PRIMARY KEY,
    banned_user_id INTEGER NOT NULL REFERENCES users(user_id),
```

```
    banned_by_user_id INTEGER NOT NULL REFERENCES users(user_id),
    server_id INTEGER NOT NULL REFERENCES servers(server_id),
    reason VARCHAR,
    timestamp TIMESTAMP NOT NULL
    );

INSERT INTO bans (banned_user_id, banned_by_user_id, server_id, reason,
timestamp)
SELECT
    FLOOR(RANDOM() * ${FILLING_AMOUNT}) + 1,
    FLOOR(RANDOM() * ${FILLING_AMOUNT}) + 1,
    FLOOR(RANDOM() * ${FILLING_AMOUNT}) + 1,
    'reason' || generate_series,
    NOW() - (random() * INTERVAL '365 days')
FROM
    generate_series(1, ${FILLING_AMOUNT});
```

Dockerfile:

```
FROM postgres:latest


EXPOSE 5432
```

Точка входа (entrypoint):

```
#!/bin/bash

execute_sql() {
    psql -U "$DB_SUPERUSER" -d "$POSTGRES_DB" -c "$1"
}

database_exists() {
    psql -U "$DB_SUPERUSER" -lqt | cut -d \| -f 1 | grep -qw "$1"
}

if ! database_exists "$POSTGRES_DB"; then
    execute_sql "CREATE DATABASE $POSTGRES_DB;"
fi

exec "$@"
```

Bash-скрипт, который запускает миграции, заполнение данными и создает роли и пользователей:

```
#!/bin/bash

execute_sql() {
    psql -U "$DB_SUPERUSER" -d "$POSTGRES_DB" -c "$1"
}

if [[ -z "$DB_USERNAME" ]]; then
    echo "Username for connection doesn't specified"
    exit 1
fi
```

```
version=$DB_VERSION

if [ -z "$version" ]; then
  versions_to_run=$(find /migrations -mindepth 1 -maxdepth 1 -type d | sort |
awk -F '/' '{ print $3 }')
else
  versions_to_run=$(find /migrations -mindepth 1 -maxdepth 1 -type d | sort |
awk -F '/' -v v="$version" '{ print $3; if ($3 == v) exit }')
fi

for v in $versions_to_run; do
  for script in $(find migrations/$v/ -name "*.sql" -type f); do
    TEMP_SQL_FILE=$(mktemp)
    sed "s/\${FILLING_AMOUNT}/$FILLING_AMOUNT/g" $script > "$TEMP_SQL_FILE"
    psql -U "$POSTGRES_USER" -d "$POSTGRES_DB" -f "$TEMP_SQL_FILE"
    rm "$TEMP_SQL_FILE"
  done
done

if ! execute_sql "\\du" | grep -qw "reader"; then
    execute_sql "CREATE USER reader WITH ENCRYPTED PASSWORD
'readerpassword';"
fi

if ! execute_sql "\\du" | grep -qw "writer"; then
    execute_sql "CREATE USER writer WITH ENCRYPTED PASSWORD
'writerpassword';"
fi

if ! execute_sql "\\du" | grep -qw "analytic"; then
    execute_sql "CREATE ROLE analytic;"
fi

execute_sql "CREATE ROLE group_role;"
execute_sql "GRANT ALL PRIVILEGES ON DATABASE $POSTGRES_DB TO group_role;"

for ((i=1; i<=4; i++)); do
    username="user$i"
    if ! execute_sql "\\du" | grep -qw "$username"; then
        execute_sql "CREATE USER $username WITH ENCRYPTED PASSWORD
'password$i';"
    fi
    execute_sql "GRANT group_role to $username;"
done

execute_sql "GRANT SELECT ON TABLE roles TO analytic;"

execute_sql "ALTER USER reader WITH NOCREATEDB;"
execute_sql "ALTER USER writer WITH CREATEDB;"
```

Итоговая структура проекта: