Отчет по 2 этапу курсовой работы

Платформа для организации мероприятий и встреч

Выполнили:
Боринский Игорь Дмитриевич
и Болорболд Аригуун
Преподаватель:
Харитонова Анастасия Евгеньевна

Санкт-Петербург
2024 год

Содержание

1 ER-диаграмма

На рис. 1 показана ER-диаграмма, которая отражает сущности, их атрибуты, ключи и связи.

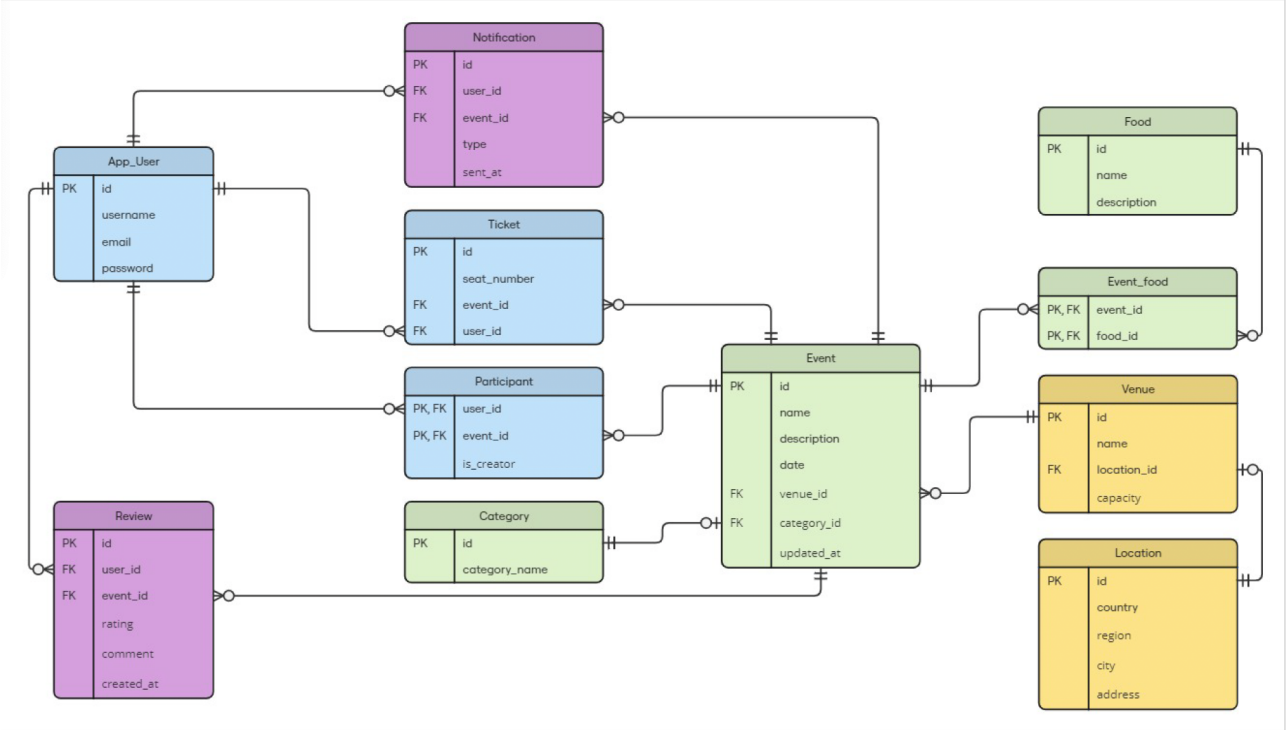

Рис. 1: ER-диаграмма системы

2 Даталогическая модель

На рис. 2 представлена даталогическая модель, которая отражает типы данных атрибутов и ограничения для каждой сущности.

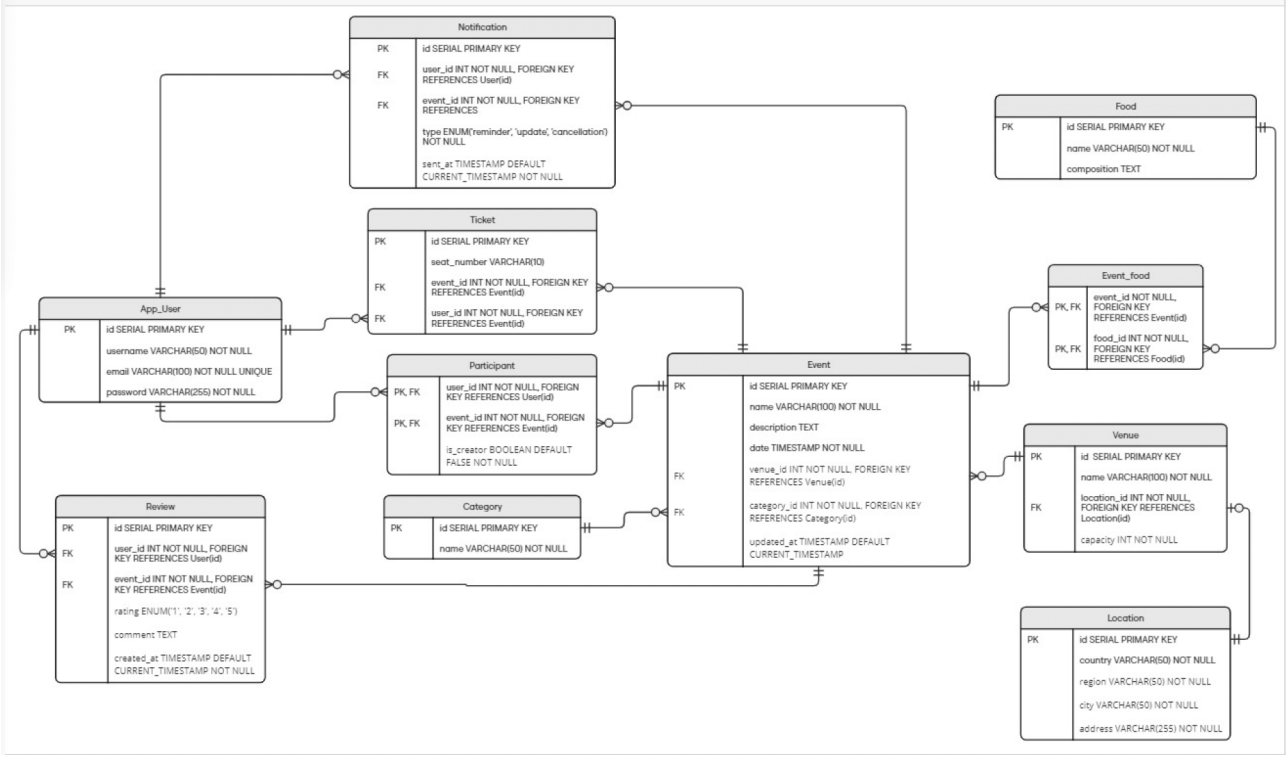

Рис. 2: Даталогическая модель системы

3 DDL для реализации даталогической модели

В данном разделе представлен SQL-код, используемый для создания структуры базы данных в системе управления базами данных PostgreSQL. Он включает в себя создание таблиц, установку первичных и внешних ключей, а также другие ограничения.

Основные сущности системы и их атрибуты отражены в коде, приведенном ниже. Структура базы данных реализована на основе даталогической модели, представленной в предыдущем разделе. Для обеспечения целостности данных были установлены внешние ключи, связывающие основные сущности системы.

4 DDL для реализации даталогической модели

В этом разделе представлен SQL-код, который был использован для создания структуры базы данных. Для создания таблиц использовались первичные ключи, внешние ключи и ограничения, чтобы обеспечить целостность данных.

```
CREATE TABLE IF NOT EXISTS App_User (
                        id SERIAL PRIMARY KEY,
                        username VARCHAR(50) NOT NULL,
                        email VARCHAR(100) NOT NULL UNIQUE,
                        password VARCHAR(255) NOT NULL
);
```

```
7
8  CREATE TABLE IF NOT EXISTS Location (
9                              id SERIAL PRIMARY KEY,
10                             country VARCHAR(100) NOT NULL,
11                             region VARCHAR(100) NOT NULL,
12                             city VARCHAR(100) NOT NULL,
13                             address VARCHAR(255) NOT NULL
14 );
15
16 CREATE TABLE IF NOT EXISTS Venue (
17                         id SERIAL PRIMARY KEY,
18                         name VARCHAR(100) NOT NULL,
19                         location_id INT NOT NULL UNIQUE REFERENCES Location(
       id),
20                         capacity INT NOT NULL
21 );
22
23 CREATE TABLE IF NOT EXISTS Category (
24                             id SERIAL PRIMARY KEY,
25                             name VARCHAR(50) NOT NULL
26 );
27
28 CREATE TABLE IF NOT EXISTS Event (
29                         id SERIAL PRIMARY KEY,
30                         name VARCHAR(100) NOT NULL,
31                         description TEXT,
32                         date TIMESTAMP NOT NULL,
33                         venue_id INT NOT NULL REFERENCES Venue(id),
34                         category_id INT NOT NULL REFERENCES Category(id),
35                         updated_at TIMESTAMP DEFAULT NOW()
36 );
37
38 CREATE TABLE IF NOT EXISTS Food (
39                        id SERIAL PRIMARY KEY,
40                        name VARCHAR(100) NOT NULL,
41                        composition TEXT
42 );
43
44 CREATE TABLE IF NOT EXISTS Review (
45                          id SERIAL PRIMARY KEY,
46                          user_id INT NOT NULL REFERENCES App_User(id),
47                          event_id INT NOT NULL REFERENCES Event(id),
48                          rating rating_enum NOT NULL,
49                          comment TEXT,
50                          created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT
       NULL
51 );
52
53
54 CREATE TABLE IF NOT EXISTS Ticket (
55                          id SERIAL PRIMARY KEY,
56                          seat_number VARCHAR(10),
57                          event_id INT NOT NULL REFERENCES Event(id),
58                          user_id INT NOT NULL REFERENCES App_User(id)
59 );
60
61 CREATE TABLE IF NOT EXISTS Notification (
62                                id SERIAL PRIMARY KEY,
63                                user_id INT NOT NULL REFERENCES App_User(id),
64                                event_id INT NOT NULL REFERENCES Event(id),
```

```
                                content TEXT NOT NULL,
                                sent_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    NOT NULL
);

CREATE TABLE IF NOT EXISTS Participant (
                                user_id INT NOT NULL REFERENCES App_User(id),
                                event_id INT NOT NULL REFERENCES Event(id),
                                is_creator BOOLEAN DEFAULT FALSE NOT NULL,
                                PRIMARY KEY (user_id, event_id)
);

CREATE TABLE IF NOT EXISTS Event_Food (
                               event_id INT NOT NULL REFERENCES Event(id),
                               food_id INT NOT NULL REFERENCES Food(id),
                               PRIMARY KEY (event_id, food_id)
);
```

Листинг 1: DDL для создания таблиц

5 Реализация триггеров и тестирование

В данной секции представлены триггеры, разработанные для обеспечения целостности данных и автоматизации некоторых процессов в базе данных. Мы реализовали три триггера для таблицы `Event` и таблицы `Participant`, а также функции для них.

5.1 Триггер для обновления времени последнего изменения

Триггер `set_timestamp` автоматически обновляет поле `updated_at` в таблице `Event` при каждом изменении данных в строке. Это позволяет сохранять информацию о последнем времени обновления события.

```
--                              Event
CREATE OR REPLACE FUNCTION update_timestamp()
RETURNS TRIGGER AS $$
BEGIN
    NEW.updated_at = NOW();
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER set_timestamp
    BEFORE UPDATE ON Event
    FOR EACH ROW
    EXECUTE FUNCTION update_timestamp();
```

Листинг 2: Функция и триггер для обновления времени изменения

5.1.1 Тестирование триггера set_timestamp

Для проверки работы триггера мы обновили название существующего события и проверили значение поля `updated_at`.

```
--
UPDATE Event
SET name = 'Updated Test-Event'
WHERE name = 'Test-Event';

--
SELECT name, updated_at FROM Event WHERE name = 'Updated Test-Event';
```

Листинг 3: Тестирование триггера set_timestamp

5.2 Триггер проверки даты события

Триггер `check_event_date_trigger` проверяет, что дата начала события не может быть в прошлом. Если пытаются вставить или обновить событие с датой в прошлом, триггер генерирует ошибку и предотвращает изменение.

```
--                                                  Event
CREATE OR REPLACE FUNCTION check_event_date()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.date < NOW() THEN
        RAISE EXCEPTION 'Event date cannot be in the past';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_event_date_trigger
    BEFORE INSERT OR UPDATE ON Event
    FOR EACH ROW
    EXECUTE FUNCTION check_event_date();
```

Листинг 4: Функция и триггер для проверки даты

5.2.1 Тестирование триггера check_event_date

Для тестирования этого триггера мы попытались создать событие с датой, которая уже прошла, и проверили, что система корректно предотвращает добавление такого события.

```
--

DO $$
BEGIN
INSERT INTO Event (name, description, date, venue_id, category_id)
VALUES ('Past-Event', 'This event is in the past', NOW() - INTERVAL '1 day',
        (SELECT id FROM Venue WHERE name = 'Test-Venue'),
        (SELECT id FROM Category WHERE name = 'Test-Category'));
EXCEPTION
    WHEN OTHERS THEN
        RAISE NOTICE 'Trigger successfully prevented event creation with a
   past date.';
END $$;

--
```

```
SELECT * FROM Event WHERE name = 'Past-Event';
```

Листинг 5: Тестирование триггера check_event_date

5.3 Триггер проверки вместимости события

Триггер `check_capacity_trigger` проверяет, что количество участников события не превышает максимальной вместимости, определенной для площадки `Venue`. Если достигается предел вместимости, добавление новых участников блокируется.

```
--
CREATE OR REPLACE FUNCTION check_event_capacity()
RETURNS TRIGGER AS $$
DECLARE
participant_count INT;
event_capacity INT;
BEGIN
    SELECT COUNT(*) INTO participant_count FROM Participant WHERE event_id =
    NEW.event_id;
    SELECT capacity INTO event_capacity FROM Venue WHERE id = (SELECT
   venue_id FROM Event WHERE id = NEW.event_id);

    IF participant_count >= event_capacity THEN
        RAISE EXCEPTION 'Event capacity exceeded. No more participants can
   be added.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_capacity_trigger
    BEFORE INSERT ON Participant
    FOR EACH ROW
    EXECUTE FUNCTION check_event_capacity();
```

Листинг 6: Функция и триггер для проверки вместимости

5.3.1 Тестирование триггера check_capacity

Для тестирования этого триггера мы добавили несколько участников до достижения предела вместимости и попытались добавить ещё одного участника, что должно было вызвать ошибку.

```
--                                                  (capacity = 3)
INSERT INTO App_User (username, email, password)
VALUES ('user3', 'user3@test.com', 'pass3');

INSERT INTO Participant (user_id, event_id, is_creator)
VALUES ((SELECT id FROM App_User WHERE username = 'user3'),
        (SELECT id FROM Event WHERE name = 'Updated Test-Event'), false);

--                                                                    ,

DO $$
BEGIN
INSERT INTO Participant (user_id, event_id, is_creator)
VALUES ((SELECT id FROM App_User WHERE username = 'user4'),
```

```
        (SELECT id FROM Event WHERE name = 'Updated Test-Event'), false);
EXCEPTION
    WHEN OTHERS THEN
        RAISE NOTICE 'Trigger successfully prevented participant addition
   beyond capacity.';
END $$;

--
SELECT * FROM Participant WHERE event_id = (SELECT id FROM Event WHERE name
   = 'Updated Test-Event')
AND user_id = (SELECT id FROM App_User WHERE username = 'user4');
```

Листинг 7: Тестирование триггера check_capacity

5.4 Выводы по тестированию триггеров

Реализованные триггеры успешно выполняют свои задачи, предотвращая некорректные изменения данных и автоматизируя обновление временных меток. Тестирование показало, что триггеры корректно реагируют на все условия, предусмотренные логикой системы.

6 Реализация критически важных запросов: PL/pgSQL функции и процедуры

Для обработки наиболее важных бизнес-процессов системы были разработаны функции и процедуры на языке PL/pgSQL. Эти запросы охватывают следующие процессы: создание мероприятия, регистрация участника на мероприятие, оставление отзыва, добавление информации о еде на мероприятие, изменение категории мероприятия и изменение вместимости мероприятия.

6.1 Функция создания мероприятия

Функция `create_event` реализует процесс создания нового мероприятия. Она проверяет обязательные поля, добавляет запись о мероприятии в таблицу `Event` и регистрирует организатора как участника мероприятия в таблице `Participant`. Если отсутствуют обязательные поля, функция генерирует исключение.

```
CREATE OR REPLACE FUNCTION create_event(
    _name VARCHAR,
    _description TEXT,
    _date TIMESTAMP,
    _venue_id INT,
    _category_id INT,
    _organizer_id INT
) RETURNS INT AS $$
DECLARE
_event_id INT;
BEGIN
    IF _name IS NULL OR _date IS NULL THEN
        RAISE EXCEPTION 'The name of the event and the date are required.';
    END IF;

    INSERT INTO Event (name, description, date, venue_id, category_id,
   updated_at)
    VALUES (_name, _description, _date, _venue_id, _category_id, NOW())
```

```
    RETURNING id INTO _event_id;

    INSERT INTO Participant (user_id, event_id, is_creator)
    VALUES (_organizer_id, _event_id, TRUE);

    RETURN _event_id;
END;
$$ LANGUAGE plpgsql;
```

Листинг 8: Функция для создания мероприятия

6.2 Функция регистрации участника на мероприятие

Функция `register_participant` проверяет, не превышено ли количество участников, регистрирует нового участника в мероприятии. В случае достижения лимита участников выбрасывается исключение.

```
CREATE OR REPLACE FUNCTION register_participant(
    _user_id INT,
    _event_id INT
) RETURNS VOID AS $$
DECLARE
_current_participants INT;
_max_participants INT;
BEGIN
    SELECT COUNT(*) INTO _current_participants FROM Participant WHERE
   event_id = _event_id;
    SELECT capacity INTO _max_participants FROM Venue WHERE id = (SELECT
   venue_id FROM Event WHERE id = _event_id);

    IF _current_participants >= _max_participants THEN
        RAISE EXCEPTION 'The maximum number of participants has been reached
   .';
    END IF;

    INSERT INTO Participant (user_id, event_id, is_creator)
    VALUES (_user_id, _event_id, FALSE);
END;
$$ LANGUAGE plpgsql;
```

Листинг 9: Функция для регистрации участника на мероприятие

6.3 Функция оставления отзыва

Функция `leave_review` проверяет, зарегистрирован ли пользователь на мероприятие, после чего позволяет оставить отзыв.

```
CREATE OR REPLACE FUNCTION leave_review(
    _user_id INT,
    _event_id INT,
    _rating rating_enum,
    _comment TEXT
) RETURNS VOID AS $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Participant WHERE user_id = _user_id AND
   event_id = _event_id) THEN
        RAISE EXCEPTION 'The user is not registered for this event.';
    END IF;
```

```
11
12      INSERT INTO Review (user_id, event_id, rating, comment, created_at)
13      VALUES (_user_id, _event_id, _rating, _comment, NOW());
14  END;
15  $$ LANGUAGE plpgsql;
```

Листинг 10: Функция для оставления отзыва о мероприятии

6.4 Функция добавления информации о еде на мероприятии

Функция `add_food_to_event` позволяет организатору добавить информацию о предоставляемой еде на мероприятии. Проверяется, что добавляющий информацию участник является организатором мероприятия.

```
1  CREATE OR REPLACE FUNCTION add_food_to_event(
2      _organizer_id INT,
3      _event_id INT,
4      _food_name VARCHAR,
5      _description TEXT
6  ) RETURNS VOID AS $$
7  DECLARE
8  _food_id INT;
9  BEGIN
10     IF NOT EXISTS (SELECT 1 FROM Participant WHERE user_id = _organizer_id
       AND event_id = _event_id AND is_creator = TRUE) THEN
11         RAISE EXCEPTION 'The user is not the organizer of the event.';
12     END IF;
13
14     INSERT INTO Food (name, composition)
15     VALUES (_food_name, _description)
16     RETURNING id INTO _food_id;
17
18     INSERT INTO Event_Food (event_id, food_id)
19     VALUES (_event_id, _food_id);
20 END;
21 $$ LANGUAGE plpgsql;
```

Листинг 11: Функция для добавления еды на мероприятие

6.5 Процедура изменения категории мероприятия

Процедура `update_event_category` позволяет изменить категорию существующего мероприятия. Если указанная категория или мероприятие не существуют, процедура генерирует исключение.

```
1  CREATE OR REPLACE PROCEDURE update_event_category(
2      event_id INT,
3      new_category_id INT
4  )
5  LANGUAGE plpgsql
6  AS $$
7  BEGIN
8      IF NOT EXISTS (SELECT 1 FROM Event WHERE id = event_id) THEN
9          RAISE EXCEPTION 'An event with this ID does not exist.';
10     END IF;
11
12     IF NOT EXISTS (SELECT 1 FROM Category WHERE id = new_category_id) THEN
13         RAISE EXCEPTION 'A category with this ID does not exist.';
```

```
    END IF;

    UPDATE Event
    SET category_id = new_category_id
    WHERE id = event_id;

    RAISE NOTICE 'The event category has been successfully updated for event ID %', event_id;
END;
$$;
```

Листинг 12: Процедура для изменения категории мероприятия

6.6 Процедура изменения вместимости мероприятия

Процедура `update_event_capacity` изменяет вместимость мероприятия. Если новое значение вместимости меньше текущего количества участников, процедура генерирует исключение.

```
CREATE OR REPLACE PROCEDURE update_event_capacity(
    _event_id INT,
    _new_capacity INT
)
LANGUAGE plpgsql
AS $$
DECLARE
_current_participants INT;
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Event WHERE id = _event_id) THEN
        RAISE EXCEPTION 'An event with this ID does not exist.';
    END IF;

    SELECT COUNT(*) INTO _current_participants FROM Participant WHERE event_id = _event_id;

    IF _new_capacity < _current_participants THEN
        RAISE EXCEPTION 'The new capacity cannot be less than the current number of participants.';
    END IF;

    UPDATE Venue
    SET capacity = _new_capacity
    WHERE id = (SELECT venue_id FROM Event WHERE id = _event_id);

    RAISE NOTICE 'The event capacity has been successfully updated for the event ID %', _event_id;
END;
$$;
```

Листинг 13: Процедура для изменения вместимости мероприятия

6.7 Тестирование функций и процедур

Для каждой функции и процедуры были разработаны тесты, которые позволяют убедиться в правильности их работы. Ниже приведены тесты с примерами использования функций и ожидаемыми результатами.

```
--          1:
SELECT create_event(
    'Test Event', 'This is a test event', NOW() + INTERVAL '1 day',
    (SELECT id FROM Venue WHERE name = 'Test Venue'),
    (SELECT id FROM Category WHERE name = 'Test Category'),
    (SELECT id FROM App_User WHERE username = 'user4')
) AS event_id;

--                                      :
                    Event
SELECT * FROM Event WHERE id = 3;

--          2:
SELECT register_participant((SELECT id FROM App_User WHERE username = '
    user12'), 3);

--                                      :
                     Participant
SELECT * FROM Participant WHERE user_id = (SELECT id FROM App_User WHERE
    username = 'user12') AND event_id = 3;

--          3:
SELECT leave_review(5, 3, '5', 'Great event!');

--                                      :
                   Review
SELECT * FROM Review WHERE user_id = 5 AND event_id = 3;

--          4:
SELECT add_food_to_event(4, 3, 'Sandwich', 'A vegetarian sandwich');

--                                      :
                   Food
SELECT * FROM Event_Food WHERE event_id = 3;
SELECT * FROM Food WHERE name = 'Sandwich';

--          5:
CALL update_event_category(3, 3);

--                                      :

SELECT * FROM Event WHERE id = 3;

--          6:

CALL update_event_capacity(3, 150);

--                                      :

SELECT capacity FROM Venue WHERE id = 1;
```

Листинг 14: Тесты для функций и процедур

6.8 Выводы по тестированию функций и процедур

Реализованные функции и процедуры выполняют свои задачи корректно, обеспечивая обработку ключевых операций, связанных с мероприятиями. Все тесты успешно пройдены, и система готова к дальнейшему использованию.

7 Реализация Liquibase Changelog-ов

Для управления версионностью схемы базы данных и выполнения миграций использовался инструмент Liquibase. Было создано несколько changelog-ов, каждая из которых отвечает за определённую функциональность, такие как создание таблиц, триггеров, функций и процедур, тестирование и вставка моковых данных.

7.1 Changelog 1.0: Типы данных и DDL

Первоначальный changelog включает в себя создание пользовательского типа данных `rating_enum`, а также выполнение DDL для создания всех необходимых таблиц. Этот changelog также содержит инструкции по откату изменений.

```yaml
databaseChangeLog:
  - changeSet:
      id: 1
      author: Raisondetre
      changes:
        - sql: DROP TYPE IF EXISTS rating_enum
        - sql: CREATE TYPE rating_enum AS ENUM ('1', '2', '3', '4', '5');
      rollback:
        - sql: DROP TYPE IF EXISTS rating_enum CASCADE

  - changeSet:
      id: 2
      author: Raisondetre
      changes:
        - sqlFile:
            path: /migrations/ddl.sql
      rollback:
        - sqlFile:
            path: /migrations/rollback-ddl.sql
```

Листинг 15: db-changelog-1.0.yaml

7.2 Changelog 2.0: Триггеры

В changelog 2.0 добавляются триггеры и их соответствующие функции. Все триггеры и функции помещены в отдельный SQL-файл.

```yaml
databaseChangeLog:
  - changeSet:
      id: 3
      author: Raisondetre
      changes:
        - sqlFile:
            path: /migrations/triggers.sql
            splitStatements: false
      rollback:
```

```
        - sqlFile:
            path: /migrations/rollback-triggers.sql
```

Листинг 16: db-changelog-2.0.yaml

7.3 Changelog 3.0: Тесты для триггеров

Этот changelog включает в себя тесты для триггеров, которые проверяют корректность их работы.

```
databaseChangeLog:
  - changeSet:
      id: 4
      author: Raisondetre
      changes:
        - sqlFile:
            path: /migrations/trigger-tests.sql
            splitStatements: false
      rollback:
        - sqlFile:
            path: /migrations/rollback-tests.sql
```

Листинг 17: db-changelog-3.0.yaml

7.4 Changelog 4.0: Функции и процедуры

В этом changelog находятся все PL/pgSQL функции и процедуры, которые были реализованы для выполнения критически важных операций с системой мероприятий.

```
databaseChangeLog:
  - changeSet:
      id: 5
      author: Raisondetre
      changes:
        - sqlFile:
            path: /migrations/functions-and-procedures.sql
            splitStatements: false
      rollback:
        - sqlFile:
            path: /migrations/rollback-functions-and-procedures.sql
```

Листинг 18: db-changelog-4.0.yaml

7.5 Changelog 5.0: Тесты для функций и процедур

Этот changelog добавляет тесты для проверок реализованных функций и процедур.

```
databaseChangeLog:
  - changeSet:
      id: 6
      author: Raisondetre
      changes:
        - sqlFile:
            path: /migrations/function-and-procedure-tests.sql
            splitStatements: false
      rollback:
        - sqlFile:
```

```
            path: /migrations/rollback-function-and-procedure-tests.sql
```

Листинг 19: db-changelog-5.0.yaml

7.6 Changelog 6.0: Вставка моковых данных

Для тестирования производительности и проверки корректности работы системы, была выполнена вставка моковых данных для всех сущностей. Changelog 6.0 содержит инструкции для загрузки данных из CSV файлов в соответствующие таблицы.

```
databaseChangeLog:
  - changeSet:
      id: 7
      author: Raisondetre
      changes:
        - loadData:
            file: /migrations/mock_data/app_user.csv
            tableName: app_user
        - loadData:
            file: /migrations/mock_data/location.csv
            tableName: location
        - loadData:
            file: /migrations/mock_data/venue.csv
            tableName: venue
        - loadData:
            file: /migrations/mock_data/category.csv
            tableName: category
        - loadData:
            file: /migrations/mock_data/event.csv
            tableName: event
            columns:
              - column:
                  name: name
                  type: STRING
              - column:
                  name: description
                  type: STRING
        - loadData:
            file: /migrations/mock_data/participant.csv
            tableName: participant
            columns:
              - column:
                  name: is_creator
                  type: STRING
        - loadData:
            file: /migrations/mock_data/review.csv
            tableName: review
            columns:
              - column:
                  name: rating
                  type: STRING
              - column:
                  name: comment
                  type: STRING
        - loadData:
            file: /migrations/mock_data/food.csv
            tableName: food
            columns:
              - column:
```

```
                    name: name
                    type: STRING
                - column:
                    name: composition
                    type: STRING
        - loadData:
            file: /migrations/mock_data/event_food.csv
            tableName: event_food
        - loadData:
            file: /migrations/mock_data/notification.csv
            tableName: notification
            columns:
              - column:
                  name: content
                  type: STRING
        - loadData:
            file: /migrations/mock_data/ticket.csv
            tableName: ticket
            columns:
              - column:
                  name: seat_number
                  type: STRING

    rollback:
      - sql: TRUNCATE TABLE app_user CASCADE
      - sql: TRUNCATE TABLE location CASCADE
      - sql: TRUNCATE TABLE venue CASCADE
      - sql: TRUNCATE TABLE category CASCADE
      - sql: TRUNCATE TABLE event CASCADE
      - sql: TRUNCATE TABLE participant CASCADE
      - sql: TRUNCATE TABLE review CASCADE
      - sql: TRUNCATE TABLE food CASCADE
      - sql: TRUNCATE TABLE event_food CASCADE
      - sql: TRUNCATE TABLE notification CASCADE
      - sql: TRUNCATE TABLE ticket CASCADE
```

Листинг 20: db-changelog-6.0.yaml

7.7 Выводы

Использование Liquibase для управления миграциями базы данных позволило автоматизировать процесс версионирования схемы, обеспечения целостности данных и откатов при необходимости.

8 Реализация индексов и анализ производительности

Для ускорения запросов, выполняемых в контексте описанных прецедентов, были разработаны индексы. Эти индексы позволяют значительно уменьшить время выполнения запросов за счёт оптимизации операций выборки данных. В этом разделе приведены индексы, реализованные для каждого прецедента, а также результаты анализа производительности до и после создания индексов при помощи `EXPLAIN ANALYZE`.

8.1 Индексы для прецедентов

8.1.1 Индекс для прецедента 1: Создание мероприятия

Индекс был создан для ускорения операций выборки данных по колонкам `venue_id` и `category_id`, которые часто используются при выборке мероприятий по месту проведения и категории.

```
CREATE INDEX idx_event_venue_category ON Event(venue_id, category_id);
```

8.1.2 Индексы для прецедента 2: Регистрация на мероприятие

Были добавлены индексы для оптимизации операций поиска по участникам мероприятия и для ускорения выборки данных по пользователю при регистрации.

```
CREATE INDEX idx_participant_event ON Participant(event_id);
CREATE INDEX idx_app_user_username ON App_User(username);
```

8.1.3 Индекс для прецедента 3: Оставление отзыва

Для ускорения проверок и выборки отзывов по событию был создан следующий индекс:

```
CREATE INDEX idx_review_event_user ON Review(event_id);
```

8.1.4 Индекс для прецедента 4: Добавление информации о еде

Индекс был создан для ускорения операций выборки еды, связанной с мероприятием, в таблице `Event_Food`.

```
CREATE INDEX idx_event_food_event ON Event_Food(event_id);
```

8.1.5 Индекс для прецедента 5: Изменение категории мероприятия

Чтобы ускорить выборку мероприятий по категории, был добавлен индекс по колонке `category_id`.

```
CREATE INDEX idx_event_category ON Event(category_id);
```

8.1.6 Индекс для прецедента 6: Изменение вместимости мероприятия

Индекс был добавлен для ускорения выборки количества участников на мероприятии.

```
CREATE INDEX idx_participant_count_event ON Participant(event_id);
```

8.2 Анализ производительности с использованием EXPLAIN ANALYZE

Для каждого прецедента был проведён анализ производительности запросов до и после создания индексов. Ниже приведены результаты этого анализа.

8.2.1 Пример 1: Выборка количества участников мероприятия

До создания индекса:

```
EXPLAIN ANALYZE
SELECT COUNT(*) FROM Participant WHERE event_id = 77;
Aggregate  (cost=405.30..405.31 rows=1 width=8) (actual time=2.176..2.176 rows=1 loops=1
   ->  Seq Scan on participant  (cost=0.00..405.00 rows=119 width=0) (actual time=0.042.
         Filter: (event_id = 77)
         Rows Removed by Filter: 19979
Planning Time: 0.266 ms
Execution Time: 2.248 ms
```

После создания индекса:

```
EXPLAIN ANALYZE
SELECT COUNT(*) FROM Participant WHERE event_id = 77;
Aggregate  (cost=4.31..4.32 rows=1 width=8) (actual time=0.052..0.052 rows=1 loops=1)
   ->  Index Only Scan using idx_participant_count_event on participant  (cost=0.29..4.3
         Index Cond: (event_id = 77)
         Heap Fetches: 0
Planning Time: 0.348 ms
Execution Time: 0.080 ms
```

Как видно из результата, индекс позволил значительно ускорить выполнение запроса — с 2.248 мс до 0.080 мс.

8.2.2 Пример 2: Выборка мероприятий по категории

До создания индекса:

```
EXPLAIN ANALYZE
SELECT * FROM Event WHERE category_id = 56;
Seq Scan on event  (cost=0.00..634.75 rows=1 width=123) (actual time=0.035..2.832 rows=1
   Filter: (category_id = 56)
   Rows Removed by Filter: 19979
Planning Time: 0.395 ms
Execution Time: 2.848 ms
```

После создания индекса:

```
EXPLAIN ANALYZE
SELECT * FROM Event WHERE category_id = 56;
Index Scan using idx_event_category on event  (cost=0.29..8.30 rows=1 width=123) (actual
   Index Cond: (category_id = 56)
Planning Time: 0.273 ms
Execution Time: 0.052 ms
```

Запрос был значительно ускорен с 2.848 мс до 0.052 мс.

8.2.3 Пример 3: Выборка отзывов по событию

До создания индекса:

```
EXPLAIN ANALYZE
SELECT * FROM Review WHERE event_id = 873;
Seq Scan on review  (cost=0.00..416.75 rows=1 width=35) (actual time=0.106..1.859 rows=1
   Filter: (event_id = 873)
   Rows Removed by Filter: 19979
Planning Time: 0.222 ms
Execution Time: 1.900 ms
```

После создания индекса:

```
EXPLAIN ANALYZE
SELECT * FROM Review WHERE event_id = 873;
Index Scan using idx_review_event_user on review  (cost=0.29..8.30 rows=1 width=35) (act
   Index Cond: (event_id = 873)
Planning Time: 0.279 ms
Execution Time: 0.042 ms
```

После создания индекса время выполнения запроса сократилось с 1.900 мс до 0.042 мс.

8.3 Выводы

Добавление индексов значительно улучшило производительность запросов, связанных с выборкой данных. Это подтверждается результатами анализа с помощью `EXPLAIN ANALYZE`.