МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский политехнический университет Петра Великого»

Институт компьютерных наук и кибербезопасности
Высшая школа технологии искусственного интеллекта
Направление 02.03.01 Математика и компьютерные науки

ОТЧЁТ

О ВЫПОЛНЕНИИ ЛАБОРАТОРНЫХ РАБОТ

по дисциплине «Методы проектирования баз данных»

Функциональные возможности баз данных

Студент,
группы 5130201/20102 _____ Михалец Мартин

Преподаватель,
к.т.н, доц. _____ Попов Сергей Геннадьевич

«_____»_____ 2024г.

Санкт-Петербург, 2024

Содержание

1 Введение

В данном отчете, описаны результаты выполнения лабораторных работ, расширяющих функциональные возможности базы данных на тему «система учета сдачи лабораторных работ». Которая была спроектирована в течении предыдущего семестрового курса «Теоретические основы баз данных». В ходе выполнения лабораторных работ, изучены и реализованы в СУБД: Представления, событийная модель (триггеры).

2 Постановка задач

В ходе прохождения данного курса, необходимо выполнить пять задач:

1. Создать представление, инкапсулирующее запрос. Написать запрос, использующий в себе представление. Представление отображает таблицу, в которой для каждого преподавателя посчитано число назначенному ему лабораторных работ и число групп, в которых их он уже принимал.

2. Написать триггеры, автоматизирующие сбор статистической информации о количестве лабораторных работ у каждого предмета.

3. Создать пользователей и предоставить им различные права доступа к представлению и таблицам, которые в нём участвуют.

4. Создать функцию и использовать её в запросе. Создать процедуру, которая будет создавать новые записи в таблицах по определённым условиям.

5. Выбрать определённый уровень изоляции транзакций и продемонстрировать наличие или отсутствие артефактов: «Грязное чтение», «Неповторяемое чтение» и «Фантомы».

3 Лабораторные работы

3.1 Работа 1: Создание представлений

Задача: Разработать представление для хранения запроса внутри СУБД.

Формулировка: Для каждого преподавателя посчитано число назначенному ему лабораторных работ и число групп, в которых их он уже принимал.

Сделано представление (view), `teacher_lab_assignments_view`, которое хранит запрос, считающий число назначенному каждому преподавателю лабораторных работ и число групп, в которых их он уже принимал.

Рис. 1 – Создание View teacher_lab_assignments_view.

```sql
SET search_path TO labs;

DROP VIEW IF EXISTS teacher_lab_assignments_view;
CREATE VIEW teacher_lab_assignments_view AS
SELECT teachers.id,
       teachers.last_name,
       teachers.first_name,
       teachers.middle_name,
       COALESCE((
              SELECT COUNT(*)

              FROM lab_assignments_teachers

              WHERE lab_assignments_teachers.teacher_id = teachers.id

              GROUP BY lab_assignments_teachers.teacher_id
       ), 0) AS assigned_labs_count,
       COALESCE((
              SELECT COUNT(DISTINCT lab_assignments.group_id)

              FROM lab_assignments_teachers

              INNER JOIN lab_assignments
              ON lab_assignments_teachers.lab_assignment_id = lab_assignments.id

              INNER JOIN students_lab_assignments
              ON lab_assignments.id = students_lab_assignments.lab_assignment_id

              WHERE lab_assignments_teachers.teacher_id = teachers.id

              GROUP BY lab_assignments_teachers.teacher_id
       ), 0) AS taken_group_labs_count

FROM teachers;
```

Представление создает таблицу с полями id преподавателя, фамилия, имя, отчество, количество назначенному преподавателю лабораторных работ и число групп, в которых их он уже принимал. Пример заполнения представлен в таблице 1.

Таблица 1 – Первых 5 кортежей `teacher_lab_assignments_view`.

id	last_name	first_name	middle_name	assigned_labs_count	taken_group_labs_count
1	Гиндуллина	Татьяна	Романовна	182	131
2	Крупницкая	Эльвира	Эдуардовна	185	130
3	Рогова	Ксения	Васильевна	178	134
4	Поникаровская	Карина	Григорьевна	182	141
5	Мазур	Дамир	Константинович	170	122

Представление используется в следующем запросе, в котором уберается id и добавляется почта преподавателя.

Рис. 2 – Запроса по View `teacher_lab_assignments_view`.

```
1  SET search_path TO labs;
2
3  SELECT teacher_lab_assignments_view.last_name,
4         teacher_lab_assignments_view.first_name,
5         teacher_lab_assignments_view.middle_name,
6         teachers.corporate_email,
7         CONCAT(teacher_lab_assignments_view.assigned_labs_count, '/',
8         teacher_lab_assignments_view.taken_group_labs_count) as lab_progress
9
10 FROM teacher_lab_assignments_view
11 INNER JOIN teachers
12 ON teacher_lab_assignments_view.id = teachers.id;
```

Результат выполнения запроса для первых 5 преподавателей, представлен в таблице 2.

Таблица 2 – Первых 5 кортежей запроса по представленью.

last_name	first_name	middle_name	corporate_email	lab_progress
Гиндуллина	Татьяна	Романовна	0@spbstu.ru	182/131
Крупницкая	Эльвира	Эдуардовна	1@spbstu.ru	185/130
Рогова	Ксения	Васильевна	2@spbstu.ru	178/134
Поникаровская	Карина	Григорьевна	3@spbstu.ru	182/141
Мазур	Дамир	Константинович	4@spbstu.ru	170/122

Проверка того, что данные через представления нельзя модифицировать:

```
SET search_path TO labs;

INSERT INTO teacher_lab_assignments_view VALUES (12345, 234, 123);
```

```
SET search_path TO labs;

UPDATE teacher_lab_assignments_view SET id = 123455 WHERE id = 1;
```

```
SET search_path TO labs;

DELETE FROM teacher_lab_assignments_view WHERE id = 1;
```

3.2 Работа 2: Событийная модель, триггеры

Задача: Разработать триггер, производящий манипуляцию над БД, при добавлении, удалении и обновлении данных.

Формулировка: Обновлять статистику о числе лабораторных работ по каждому предмету, при добавлении и удалении предмета или лабораторной работы.

Скрипт ниже, создает таблицу, которая содержит поля id предмета и количество лабораторных работ, после чего её заполняет для соответствия с настоящим состоянием.

```
SET search_path TO labs;

DROP TABLE IF EXISTS subject_lab_counts;
CREATE TABLE subject_lab_counts (
       subject_id int,
       lab_count  int,
       PRIMARY KEY (subject_id),
       FOREIGN KEY (subject_id) REFERENCES subjects(id) ON DELETE CASCADE
);

INSERT INTO subject_lab_counts
SELECT subjects.id, COUNT(*)

FROM subjects

INNER JOIN labs
ON subjects.id = labs.subject_id

GROUP BY subjects.id;
```

Триггеры будут вызваны при добавлении строк в таблицу subjects и добавлении или удалении строк в таблице labs. При добавлении нового предмета, в таблицу статистика добавится соответствующая запись с начальным количеством лабораторных работ равному нулю. При добавлении или удалении лабораторной работы, в таблице статистики, соответственно, инкрементируется или декрементируется количество лабораторных работ, по которому данная лабораторная работа была задана.

Ниже приведены скрипты для создания и привязания к таблицам всех 3 триггеров:

Рис. 3 – Добавление предмета.

```
1  SET search_path TO labs;
2
3  CREATE OR REPLACE FUNCTION insert_subject_to_subject_lab_counts()
4         RETURNS TRIGGER
5         LANGUAGE PLPGSQL
6  AS $$
7  BEGIN
8          INSERT INTO subject_lab_counts(subject_id, lab_count)
9          VALUES(NEW.id, 0);
10
11         RETURN NEW;
12 END;
13 $$
14
15 CREATE OR REPLACE TRIGGER subject_inserted
16        AFTER INSERT ON subjects
17        FOR EACH ROW EXECUTE FUNCTION insert_subject_to_subject_lab_counts();
```

Рис. 4 – Добавление или удаление лабораторной работы.

```
SET search_path TO labs;

CREATE OR REPLACE FUNCTION increment_subject_lab_count()
       RETURNS TRIGGER
       LANGUAGE PLPGSQL
AS $$
BEGIN
        UPDATE subject_lab_counts
        SET lab_count = subject_lab_counts.lab_count + 1
        WHERE subject_lab_counts.subject_id = NEW.subject_id;

        RETURN NEW;
END;
$$;

CREATE OR REPLACE FUNCTION decrement_subject_lab_count()
       RETURNS TRIGGER
       LANGUAGE PLPGSQL
AS $$
BEGIN
        UPDATE subject_lab_counts
        SET lab_count = subject_lab_counts.lab_count - 1
        WHERE subject_lab_counts.subject_id = OLD.subject_id;

        RETURN OLD;
END;
$$;

CREATE OR REPLACE TRIGGER lab_inserted
       AFTER INSERT ON labs
       FOR EACH ROW EXECUTE FUNCTION increment_subject_lab_count();

CREATE OR REPLACE TRIGGER lab_deleted
       AFTER DELETE ON labs
       FOR EACH ROW EXECUTE FUNCTION decrement_subject_lab_count();
```

Ниже приведены запросы для проверки работоспособности триггеров и в целом таблицы статистики.

Рис. 5 – Запрос статистики.

```
SET search_path TO labs;

SELECT name AS subject, lab_count
FROM subject_lab_counts
INNER JOIN subjects
ON subject_id = subjects.id;
```

Таблица 3 – Начальная статистика.

subject	lab_count
Дискретная математика	39
Безопасность жизнедеятельности	40
Объектно-ориентированное программирование	34
Теория графов	45
Основы архитектуры ЦВМ	60
Физика	37
Программирование микроконтроллеров	55
Архитектура суперкомпьютеров	30
Алгоритмизация и программирование	61

Рис. 6 – Запрос добавления предмета.

```
SET search_path TO labs;

INSERT INTO subjects (name) VALUES ('test-subject');
```

Таблица 4 – Статистика после добавления предмета.

subject	lab_count
Дискретная математика	39
Безопасность жизнедеятельности	40
Объектно-ориентированное программирование	34
Теория графов	45
Основы архитектуры ЦВМ	60
Физика	37
Программирование микроконтроллеров	55
Архитектура суперкомпьютеров	30
Алгоритмизация и программирование	61
test-subject	0

Рис. 7 – Запрос добавления лабораторной работы.

```
1  SET search_path TO labs;
2
3  INSERT INTO labs (name, subject_id)
4  VALUES ('test-lab', (SELECT id FROM subjects WHERE name = 'test-subject'));
```

Таблица 5 – Статистика после добавления лабораторной работы.

subject	lab_count
Дискретная математика	39
Безопасность жизнедеятельности	40
Объектно-ориентированное программирование	34
Теория графов	45
Основы архитектуры ЦВМ	60
Физика	37
Программирование микроконтроллеров	55
Архитектура суперкомпьютеров	30
Алгоритмизация и программирование	61
test-subject	1

Рис. 8 – Запрос удаления лабораторной работы.

```
1 SET search_path TO labs;
2
3 DELETE FROM labs WHERE name = 'test-lab';
```

Таблица 6 – Статистика после удаления лабораторной работы.

subject	lab_count
Дискретная математика	39
Безопасность жизнедеятельности	40
Объектно-ориентированное программирование	34
Теория графов	45
Основы архитектуры ЦВМ	60
Физика	37
Программирование микроконтроллеров	55
Архитектура суперкомпьютеров	30
Алгоритмизация и программирование	61
test-subject	0

Рис. 9 – Запрос удаления предмета.

```
1 SET search_path TO labs;
2
3 DELETE FROM subjects WHERE name = 'test-subject';
```

Таблица 7 – Статистика после удаления предмета.

subject	lab_count
Дискретная математика	39
Безопасность жизнедеятельности	40
Объектно-ориентированное программирование	34
Теория графов	45
Основы архитектуры ЦВМ	60
Физика	37
Программирование микроконтроллеров	55
Архитектура суперкомпьютеров	30
Алгоритмизация и программирование	61

3.3 Работа 3: Разграничение прав доступа

Задача: Создать пользователей и предоставить им различные права доступа к представлению и таблицам, которые в нём участвуют.

Формулировка: Создать пользователя `user_readonly` и `user_readwrite`. Выдать первому права только на чтение данных из представления, созданного ранее. Второму также выдать права на редактирование всех таблиц, участвующих в этом представлении. Сравнить результаты различных операций с этими таблицам от имени этих пользователей.

Код запросов для создания пользователей и выдачи им необходимых прав представлен на рис. 10. По мимо стандартных прав на операции `insert`, `update`, `delete`, пользователю `user_readonly` также необходимо предоставить права `usage` и `select` для объектов последовательностей, связанных с редактируемыми таблицами. Это особенность СУБД PostgreSQL, в которых последовательности выделены в отдельные объекты базы данных и, соответственно, для работы с ними необходимо явно предоставить пользователю права. Последователь-

ности используются в PostgreSQL для автоинкрементных полей, в первую очередь для первичных ключей таблиц. Также обоим пользователем необходимо предоставить права на использование схемы базы данных, без этого разрешения пользователь даже при наличии прав на отдельные объекты внутри схемы не сможет к ним получить доступ.

Примеры различных операций от имён этих пользователей вместе с реакцией СУБД демонстрируются в таблице 8.

Рис. 10 – Код запросов для создания пользователей и выдачи им определённых прав.

```
1  SET search_path TO labs;
2
3  CREATE USER user_readonly  WITH password '123';
4  CREATE USER user_readwrite WITH password '123';
5
6  GRANT USAGE ON SCHEMA labs TO user_readonly;
7  GRANT USAGE ON SCHEMA labs TO user_readwrite;
8
9  GRANT SELECT ON teacher_lab_assignments_view TO user_readonly;
10 GRANT SELECT ON teacher_lab_assignments_view TO user_readwrite;
11
12 GRANT SELECT, INSERT, UPDATE, DELETE ON teachers                TO user_readwrite;
13 GRANT SELECT, INSERT, UPDATE, DELETE ON lab_assignments         TO user_readwrite;
14 GRANT SELECT, INSERT, UPDATE, DELETE ON lab_assignments_teachers TO user_readwrite;
15 GRANT SELECT, INSERT, UPDATE, DELETE ON students_lab_assignments TO user_readwrite;
16
17 GRANT USAGE, SELECT ON SEQUENCE  teachers_id_seq                TO user_readwrite;
18 GRANT USAGE, SELECT ON SEQUENCE  lab_assignments_id_seq         TO user_readwrite;
19 GRANT USAGE, SELECT ON SEQUENCE  lab_assignments_teachers_id_seq TO user_readwrite;
20 GRANT USAGE, SELECT ON SEQUENCE  students_lab_assignments_id_seq TO user_readwrite;
```

3.4 Работа 4: Создание функций и процедур

Задача: Создать функцию и использовать её в запросе. Создать процедуру, которая будет создавать новые записи в таблицах по определённым условиям.

Формулировка: Создать функцию `to_initials`, которая будет получать на вход строки с ФИО, а возвращать строку с инициалами. Создать процедуру `add_student`, которая по заданной информации о студенте и его группе будет создавать нужные записи в таблицах

Таблица 8 – Сравнение результатов действий, выполненных от пользователей user_readonly и user_readwrite.

№	user_readwrite	user_readonly
1	Просмотр содержимого представления. SELECT * FROM teacher_lab_assignments_view	
	(100 rows) Time: 216.026 ms	(100 rows) Time: 208.032 ms
2	Изменение данных через представление. INSERT INTO teacher_lab_assignments_view (id, assigned_labs_count, taken_group_labs_count) VALUES (9999, 2, 3)	
	ERROR: 0A000: cannot insert into column ”assigned_labs_count”of view ”teacher_lab_assignments_view” DETAIL: View columns that are not columns of their base relation are not updatable. LOCATION: rewriteTargetView, rewriteHandler.c:3172 Time: 0.419 ms	ERROR: 0A000: cannot insert into column ”assigned_labs_count”of view ”teacher_lab_assignments_view” DETAIL: View columns that are not columns of their base relation are not updatable. LOCATION: rewriteTargetView, rewriteHandler.c:3172 Time: 0.419 ms
3	Чтение данных из таблицы, несвязанной с представлением. SELECT * FROM teachers	
	ERROR: 42501: permission denied for table students LOCATION: aclcheck_error, aclchk.c:3447 Time: 0.651 ms	ERROR: 42501: permission denied for table students LOCATION: aclcheck_error, aclchk.c:3447 Time: 0.651 ms
4	Чтение данных из таблицы teachers. SELECT * FROM teachers	
	(100 rows) Time: 0.650 ms	ERROR: 42501: permission denied for table teachers LOCATION: aclcheck_error, aclchk.c:3447 Time: 0.650 ms
5	Чтение данных из таблицы lab_assignments. SELECT * FROM lab_assignments	
	(5993 rows) Time: 11.647 ms	ERROR: 42501: permission denied for table lab_assignments LOCATION: aclcheck_error, aclchk.c:3447 Time: 0.633 ms
6	Чтение данных из таблицы lab_assignments_teachers. SELECT * FROM lab_assignments_teachers	
	(17926 rows) Time: 12.026 ms	ERROR: 42501: permission denied for table lab_assignments_teachers LOCATION: aclcheck_error, aclchk.c:3447 Time: 0.743 ms
7	Чтение данных из таблицы students_lab_assignments. SELECT * FROM students_lab_assignments	
	(205032 rows) Time: 199.778 ms	ERROR: 42501: permission denied for table students_lab_assignments LOCATION: aclcheck_error, aclchk.c:3447 Time: 0.432 ms
8	Добавление данных в таблицу teachers. INSERT INTO teachers (id, last_name, first_name) VALUES (9999, ’Michalec’, ’Martin’)	
	INSERT 0 1 Time: 8.221 ms	ERROR: 42501: permission denied for table teachers LOCATION: aclcheck_error, aclchk.c:3447 Time: 0.675 ms
9	Добавление данных в таблицу students. INSERT INTO students (id, last_name, first_name) VALUES (9999, ’Michalec’, ’Martin’)	
	ERROR: 42501: permission denied for table teachers LOCATION: aclcheck_error, aclchk.c:3447 Time: 0.305 ms	ERROR: 42501: permission denied for table teachers LOCATION: aclcheck_error, aclchk.c:3447 Time: 0.824 ms

students и groups.

Код определения функции to_initials представлен на Рис. 11. Функция принимает на вход три параметра типа varchar: last_name — фамилия, first_name — имя, middle_name — отчество. Возвращает также varchar — строку с инициалами. Если фамилия или имя не заданы, функция возвращает пустую строку. Функция также отдельно обрабатывает случай, когда не задано отчество, в таком случае инициалы будут состоять только из первой буквы имени и фамилии. Код запроса с использованием этой функции представлен на Рис. 12, а результат его выполнения на Рис. 9.

Рис. 11 – Код определения функции to_initials.

```
1  SET search_path TO labs;
2
3  CREATE OR REPLACE FUNCTION to_initials(
4         last_name varchar,
5         first_name varchar,
6         middle_name varchar)
7  RETURNS varchar
8  LANGUAGE PLPGSQL
9  AS $$
10 BEGIN
11        IF first_name IS NULL OR
12           first_name = '' OR
13           last_name IS NULL OR
14           last_name = '' THEN
15               RETURN '';
16        END IF;
17
18        IF middle_name IS NULL OR
19           middle_name = '' THEN
20               RETURN concat(last_name, ' ',
21                             SUBSTRING(first_name FROM 1 FOR 1), '.');
22        END if;
23
24        RETURN concat(last_name, ' ',
25                      SUBSTRING(first_name FROM 1 FOR 1), '.',
26                      SUBSTRING(middle_name FROM 1 FOR 1), '.');
27 END;
28 $$
```

Рис. 12 – Код JOIN запроса с использованием функции to_initials.

```
SET search_path TO labs;

SELECT last_name,
       first_name,
       middle_name,
       to_initials (last_name, first_name, middle_name)

FROM groups
INNER JOIN students
ON groups.id = group_id
WHERE groups.id < 2;
```

Таблица 9 – Последние десять записей результата выполнения запроса с применением функции to_initials.

last_name	first_name	middle_name	to_initials
Аверченко	Петр	Аркадьевич	Аверченко П.А.
Легошин	Григорий	Дамирович	Легошин Г.Д.
Маковецкая	Надежда	Федоровна	Маковецкая Н.Ф.
Азольская	Надежда	Валентиновна	Азольская Н.В.
Курляндцев	Роман	Леонидович	Курляндцев Р.Л.
Фролов	Евгений	Алексеевич	Фролов Е.А.
Наклеушева	Галина	Ефимовна	Наклеушева Г.Е.
Коркмазова	Альбина	Маратовна	Коркмазова А.М.
Щиглева	Екатерина	Ринатовна	Щиглева Е.Р.

Код определения процедуры add_student представлен ниже.

Процедура принимает на вход пять параметров типа varchar: last_surname – фамилия студента, first_name – имя студента, middle_name – отчетсво студента, p_corporate_email – электронная почта студента, p_group_designator – идентификатор группы студента. Внутри процедуры сначала проверяется существование студента в базе данных по переданным данным. Если студент не существует, он создаётся. Далее проверяется, существует ли группа с указанными идентификатором. Если группа не найдена, то создаётся новая запись о группе. В конце связывается студент со свей группой через id.

```
1  SET search_path TO labs;
2
3  CREATE OR REPLACE PROCEDURE add_student(
4         p_last_name varchar(25),
5         p_first_name varchar(25),
6         p_middle_name varchar(25),
7         p_corporate_email varchar(25),
8         p_group_designator varchar(25))
9  LANGUAGE PLPGSQL
10 AS $$
11 DECLARE
12        l_student_id integer;
13        l_group_id integer;
14        l_current_group_id integer;
15 BEGIN
16        SELECT groups.id INTO l_group_id
17        FROM groups
18        WHERE designator = p_group_designator
19        LIMIT 1;
20
21        IF NOT FOUND THEN
22               INSERT INTO groups (designator)
23               VALUES (p_group_designator)
24               RETURNING groups.id INTO l_group_id;
25               RAISE NOTICE 'Inserted group: %', l_group_id;
26        END IF;
27
28        SELECT students.id INTO l_student_id
29        FROM students
30        WHERE last_name       = p_last_name   AND
31              first_name      = p_first_name  AND
32              middle_name     = p_middle_name AND
33              corporate_email = p_corporate_email
34        LIMIT 1;
35
36        IF NOT FOUND THEN
37               INSERT INTO students (last_name,
38                                     first_name,
39                                     middle_name,
40                                     corporate_email,
41                                     group_id)
42               VALUES (p_last_name,
43                       p_first_name,
44                       p_middle_name,
```

```
45                      p_corporate_email,
46                      l_group_id)
47              RETURNING id INTO l_student_id;
48              RAISE NOTICE 'Inserted student: %', l_student_id;
49      END IF;
50
51      SELECT group_id INTO l_current_group_id
52      FROM students
53      WHERE students.id = l_student_id
54      LIMIT 1;
55
56      IF l_current_group_id IS DISTINCT FROM l_group_id THEN
57              UPDATE students
58              SET group_id = l_group_id
59              WHERE students.id = l_student_id;
60              RAISE NOTICE 'Updated group_id: % -> %', l_current_group_id, l_group_id;
61      END IF;
62  END;
63  $$;
```

Проверка работы процедуры add_student:

Рис. 13 – Вызов add_student при отсутсвии и студента и группы

```
1  SET search_path TO labs;
2
3  BEGIN;
4  CALL add_student('TEST1', 'TEST2', 'TEST3', 'TEST4', 'TEST5');
5  SELECT * FROM students WHERE last_name = 'TEST1';
6  SELECT * FROM groups WHERE designator = 'TEST5';
7  ROLLBACK;
```

```
NOTICE:  Inserted group: 325

NOTICE:  Inserted student: 6853
```

Рис. 14 – Вызов add_student при отсутсвии группы

```
SET search_path TO labs;

BEGIN;
CALL add_student('TEST1', 'TEST2', 'TEST3', 'TEST4', 'TEST5');
CALL add_student('TEST1', 'TEST2', 'TEST3', 'TEST4', 'OTHER');
SELECT * FROM students WHERE last_name = 'TEST1';
SELECT * FROM groups WHERE designator = 'OTHER';
ROLLBACK;
```

```
NOTICE:  Inserted group: 323

NOTICE:  Updated group_id: 322 -> 323
```

Рис. 15 – Вызов add_student при отсутсвии студента

```
SET search_path TO labs;

BEGIN;
CALL add_student('TEST1', 'TEST2', 'TEST3', 'TEST4', 'TEST5');
CALL add_student('OTHER', 'TEST2', 'TEST3', 'TEST4', 'TEST5');
SELECT * FROM students WHERE last_name = 'OTHER';
SELECT * FROM groups WHERE designator = 'TEST5';
ROLLBACK;
```

```
NOTICE:  Inserted group: 327

NOTICE:  Inserted student: 6856
```

Рис. 16 – Вызов add_student при присутсвии и студента и группы

```
SET search_path TO labs;

BEGIN;
CALL add_student('TEST1', 'TEST2', 'TEST3', 'TEST4', 'TEST5');
CALL add_student('TEST1', 'TEST2', 'TEST3', 'TEST4', 'TEST5');
SELECT * FROM students WHERE last_name = 'TEST1';
SELECT * FROM groups WHERE designator = 'TEST5';
ROLLBACK;
```

```
NOTICE:  Inserted group: 329

NOTICE:  Inserted student: 6859
```

3.5 Работа 5: Транзакционная модель

Задача: Выбрать определённый уровень изоляции транзакций и продемонстрировать наличие или отсутствие артефактов: «Грязное чтение», «Неповторяемое чтение» и «Фантомы».

Уровень изоляции `Read Commited` в PostgreSQL задётся при помощи команды: `SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`

На этом уровне изоляции исключается артефакт «Грязное чтение», однако сохраняются артефакты «Неповторяемое чтение» и «Фантомы».

В таблице 10 представлены транзакции, на примере которых демонстрируется отсутствие артефакта «Грязное чтение».

В таблице 11 представлены транзакции, на примере которых демонстрируется наличие артефакта «Неповторяемое чтение».

В таблице 12 представлены транзакции, на примере которых демонстрируется наличие артефакта «Фантомы».

Таблица 10 – Транзакции для демонстрации отсутствия артефакта «Грязное чтение».

t	**Транзакция 1**	**Транзакция 2**
	`SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`	
	Запуск транзакции 1 `BEGIN;`	Запуск транзакции 2 `BEGIN;`

t_1	Изменение фамилии преподавателя с `id = 10`
	
	`cmmm=*# SELECT id, last_name`
	`FROM teachers WHERE id = 10;`
	`id | last_name`
	`- -+- - - - - -`
	`10 | Дубенкина`
	`(1 row)`
	`cmmm=*# UPDATE teachers`
	`SET last_name = 'EDITED'`
	`WHERE id = 10;`
	`UPDATE 1`
	`cmmm=*# SELECT id, last_name`
	`FROM teachers WHERE id = 10;`
	`id | last_name`
	`- -+- - - - - -`
	`10 | EDITED`
	`(1 row)`	

t_2		Получение фамилии преподавателя с `id = 10`
		
		и завершение транзакции
		`cmmm=*# SELECT id, last_name`
		`FROM teachers WHERE id = 10;`
		`id | last_name`
		`- -+- - - - - -`
		`10 | Дубенкина`
		`(1 row)`
		`COMMIT;`

t_3	Отмена внесённых изменений транзакцией 1
	
	`ROLLBACK;`
	`cmmm=*# SELECT id, last_name`
	`FROM teachers WHERE id = 10;`
	`id | last_name`
	`- -+- - - - - -`
	`10 | Дубенкина`
	`(1 row)`	

	В первой транзакции у преподавателя с `id = 10` изменяется имя с «Дубенкина» на «EDITED». В момент после обновления вторая транзакция читает эту же запись из `teachers`, однако в связи с отсутствием артефакта «Грязное чтение» она видит имя «Дубенкина», то есть значение до обновлений первой транзакции.	

Таблица 11 – Транзакции для демонстрации наличия артефакта «Неповторяемое чтение».

t	**Транзакция 1**	**Транзакция 2**
	`SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`	
	Запуск транзакции 1 `BEGIN;`	Запуск транзакции 2 `BEGIN;`

t_1	Изменение фамилии преподавателя с `id = 10`
	`cmmm=*# SELECT id, last_name`
	`FROM teachers WHERE id = 10;`
	`id | last_name`
	`- -+- - - - - -`
	`10 | Дубенкина`
	`(1 row)`	

t_2		Изменение фамилии преподавателя с `id = 10`
		и фиксация транзакции
		`cmmm=*# SELECT id, last_name`
		`FROM teachers WHERE id = 10;`
		`id | last_name`
		`- -+- - - - - -`
		`10 | Дубенкина`
		`(1 row)`
		`cmmm=*# UPDATE teachers`
		`SET last_name = 'EDITED'`
		`WHERE id = 10;`
		`UPDATE 1`
		`cmmm=*# SELECT id, last_name`
		`FROM teachers WHERE id = 10;`
		`id | last_name`
		`- -+- - - - - -`
		`10 | EDITED`
		`(1 row)`
		`COMMIT;`

t_3	Получение фамилии преподавателя с `id = 10`
	и фиксация транзакции
	
	`cmmm=*# SELECT id, last_name`
	`FROM teachers WHERE id = 10;`
	`id | last_name`
	`- -+- - - - - -`
	`10 | EDITED`
	`(1 row)`
	`COMMIT;`	

	В первой транзакции происходит получение фамилии преподавателя с `id = 10` – выводится имя «Дубенкина». После чего эта же запись обновляется во второй транзакции, имя изменяется на «EDITED», изменения фиксируются и вторая транзакция успешно завершается. Затем первая транзакция повторно считывает эту запись и получает обновлённые данные – имя «EDITED», из-за наличия артефакта «Неповторяемое чтение».	

Таблица 12 – Транзакции для демонстрации наличия артефакта «Фантомы».

t	**Транзакция 1**	**Транзакция 2**
	SET TRANSACTION ISOLATION LEVEL READ COMMITTED;	
	Запуск транзакции 1 BEGIN;	Запуск транзакции 2 BEGIN;
t_1	Получение фамилий преподавателей с id >= 100 cmmm=*# SELECT id, last_name FROM teachers WHERE id >= 100; id | last_name - - -+- - - - - - 100 | Уилсон (1 row)	
t_2		Добавление нового преподавателя и фиксация транзакции cmmm=*# SELECT id, last_name FROM teachers WHERE id >= 100; id | last_name - - -+- - - - - - 100 | Уилсон (1 row) cmmm=*# INSERT INTO teachers (first_name, last_name, middle_name) VALUES ('Фантом', 'Фантомович', 'Фантомов'); INSERT 1 cmmm=*# SELECT id, last_name FROM teachers WHERE id = 10; id | last_name - - -+- - - - - - 100 | Уилсон 101 | Фантомов (2 rows) COMMIT;
t_3	Получение фамилий преподавателей с id >= 100 cmmm=*# SELECT id, last_name FROM teachers WHERE id = 10; id | last_name - - -+- - - - - - 100 | Уилсон 101 | Фантомов (2 rows) COMMIT;	
	В первой транзакции происходит получение фамилий преподавателей из таблицы teachers с id >= 100, в ответ выводится один представлены с фамилией «Уилсон». После чего в эту таблицу добовляется новый преподаватель во второй транзакции с фамилией «Фантомов», изменения фиксируются и вторая транзакция успешно завершается. Затем первая транзакция повторно получает записи с id >= 100. В этот раз, по мимо преподавателя с фамилией «Уилсон», она также получает данные о записи, добавленной второй транзакцией – преподавателе с фамилией «Фантомов», эта запись и является артефактом «Фантом».	

Заключение

В процессе изучения данного курса было выполнено пять лабораторных работ:

1. Создано представление, которое инкапсулирует запрос. Показана невозможность модификации этого представления. Написан запрос, использующий созданное представление.

2. Создана таблица для подсчёта количества лабораторных работ по каждому предмету. Также разработано 5 триггеров, автоматизирующих обновление статистики в таблице.

3. Созданы два пользователя с различными уровнями доступа. Первый пользователь имеет права только на просмотр представления, а второй — на просмотр, вставку, удаление и обновление данных во всех таблицах, участвующих в представлении. На 8 примерах продемонстрировано поведение СУБД при различных операциях для каждого пользователя, включая недопустимые.

4. Разработаны процедура и функция. Функция принимает фамилию, имя и отчество человека и возвращает его фамилию и инициалы.

5. Управление транзакциями. Установлен уровень изоляции транзакций `Read Committed`, продемонстрировано отсутствие артефакта «Грязное чтение» и наличие артефактов «Неповторяемое чтение» и «Фантомы».

На работу было потрачено около 2-х месяцев, за которые было написано более 300 строк кода.

Работа была выполнена в системе управления базами данных PostgreSQL 14.13.

Полученные знания могут быть и будут использованы в работе над последующими проектами и заданиями.

Список литературы

1. PostgreSQL documentation URL: https://www.postgresql.org/docs/, Дата обращения: 18.11.2024