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 технологий искусственного интеллекта

Направление: 02.03.01 «Математика и компьютерные науки»

Отчет по лабораторным работам «Теоретические основы баз данных»

Студент,
группы 5130201/20102 _____ Тищенко А. А.

Преподаватель _____ Попов С. Г.

«_____»_____ 2024г.

Санкт-Петербург, 2024

Содержание

Введение

В данном отчёте описан результат выполнения лабораторных работ, расширяющих функциональные возможности базы данных для проведения соревнований по стрельбе из лука, которая была разработана в течение предыдущего семестрового курса «Теоретические основы баз данных». В ходе выполнения лабораторных работ изучены и реализованы в СУБД: представления, событийная модель и триггеры, разграничение прав доступа пользователей.

1 Постановка задачи

В ходе прохождения данного курса необходимо выполнить пять лабораторных работ.

1. Создать представление, инкапсулирующее запрос. Написать запрос, использующий в себе представление.
2. Написать триггеры, автоматизирующие сбор статистической информации о количестве соревнований, в которых участвовал каждый судья.
3. Создать двух пользователей. Первый должен иметь доступ только на просмотр представления из первого задания. Второй также должен уметь редактировать таблицы, участвующие в запросе представления.

2 Лабораторные работы

2.1 Работа 1: Создание представлений

Задача: Разработать представление для хранения запроса внутри СУБД.

Формулировка задачи: Посчитать для каждого спортсмена число заявок и число серий. Представление выдает только идентификатор спортсмена, число его заявок на соревнования и число серий выстрелов. Использовать представление в другом запросе.

Было создано представление sportsman_requests_series_count, которое хранит запрос, выполняющий поставленную задачу. В представлении участвуют таблицы `sportsman`, `participant_request` и `shot_series`. Код запроса для создания представления представлен на Рис. 2, а первые десять записей получившегося представления представлены на Рис. 1.

id_sportsman integer	pr_count bigint	ss_count bigint
201	97	2674
202	87	2275
203	99	2520
204	87	2074
205	94	2579
206	89	2340
207	97	2384
208	89	2393
209	97	2413
210	93	2355

Рис. 1. Первые десять записей в представлении `sportsman_requests_series_count`.

```
1   create view
2       sportsman_requests_series_count as
3   select
4       s.id_sportsman,
5       count(distinct pr.id_participant_request) as pr_count,
6       count(ss.id_shot_series) as ss_count
7   from
8       sportsman as s
9       left join participant_request as pr
10          on s.id_sportsman = pr.id_sportsman
11      left join shot_series as ss
12          on pr.id_participant_request = ss.id_participant_request
13  group by
14      s.id_sportsman;
```

Рис. 2. Запрос для создания представления `sportsman_requests_series_count`.

Также был составлен пример запроса, в котором используется представление `sportsman_requests_series_count`, его код представлен на Рис. 3. В этом запросе

выбираются 10 самых активных с точки зрения участия в соревнованиях спортсменов из клуба «ЛК Парадокс Лучника». К исходному представлению также добавляются дополнительные столбцы с данными спортсмена из таблицы `sportsman`. Результат выполнения этого запроса представлен на Рис. 4.

```
1   select
2       srsc.id_sportsman,
3       s.name,
4       s.category,
5       srsc.pr_count,
6       srsc.ss_count
7   from
8       sportsman_requests_series_count as srsc
9       join sportsman as s on srsc.id_sportsman = s.id_sportsman
10  where
11      s.gender = 'Мужчина'
12      and s.club = 'ЛК_Парадокс_Лучника'
13  order by
14      srsc.pr_count desc
15  limit 10;
```

Рис. 3. Код примера запроса, в котором используется представление `sportsman_requests_series_count`.

id_sportsman integer	name character varying (100)	category character varying (100)	pr_count bigint	ss_count bigint
252	Бочаров Алексей	КМС	100	2341
298	Бочаров Алексей	КМС	99	2597
262	Ушаков Алексей	б/р	97	2515
207	Конанков Алексей	б/р	97	2384
299	Конанков Алексей	б/р	97	2442
282	Огородников Михаил	б/р	95	2365
206	Бочаров Алексей	КМС	89	2340
215	Серебряков Александр	б/р	89	2219
236	Огородников Михаил	б/р	88	2132
261	Серебряков Александр	б/р	87	2304

Рис. 4. Результат выполнения запроса с участием представления `sportsman_requests_series_count`.

Созданное представление нельзя обновлять напрямую, а попытки его обновить приведут к ошибке, которая демонстрируется на Рис. 5. Изменить данные представления можно только изменив исходные таблицы, которые в нём участвуют.

```
ERROR:  Представления с GROUP BY не обновляются автоматически.вставить
данные в представление "sportsman_requests_series_count" нельзя

ОШИБКА:  вставить данные в представление "sportsman_requests_series_count"
нельзя
SQL state: 55000
Detail: Представления с GROUP BY не обновляются автоматически.
Hint: Чтобы представление допускало добавление данных, установите триггер
INSTEAD OF INSERT или безусловное правило ON INSERT DO INSTEAD.
```

Рис. 5. Результат попытки изменения данных напрямую через представление `sportsman_requests_series_count`.

2.2 Работа 2: Событийная модель, триггеры

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

Формулировка задачи: Необходимо вести статистику о том, в скольки соревнованиях участвовал каждый судья. Событийная модель должна поддерживать актуальность данных в таблицах со статистикой при изменении данных.

Для выполнения поставленной задачи, была создана отдельная таблица для сохранения статистики – `judge_statistics`. Код запроса для её создания представлен на Рис. 6.

```
1   create table judge_statistics (
2       id_judge integer primary key,
3       competition_count integer not null default 0,
4       foreign key (id_judge) references judge(id_judge)
5   );
```

Рис. 6. Код запроса для создания таблицы `judge_statistics`.

Данные в таблице `judge_statistics` должны изменяться при изменении таблиц `judge` и `judge_request`. Список созданных для этого триггеров представлен ниже.

1. `after_judge_insert`
    - **Срабатывает при** добавлении новых судей в таблицу `judge`.
    - **Действие:** добавляет нового судью в `judge_statistics`.
    - **Код:** листинг 7.

```
1  create or replace function update_statistics_on_judge_insert()
2  returns trigger as
3  begin
4      insert into judge_statistics (id_judge, competition_count)
5      values (new.id_judge, 0);
6      return new;
7  end;
8   language plpgsql;
9
```

```
10  create trigger after_judge_insert
11  after insert on judge
12  for each row
13  execute function update_statistics_on_judge_insert();
```

Рис. 7. Код запроса для создания триггера `after_judge_insert` и функции `update_statistics_on_judge_insert`.

2. `after_judge_delete`

   - **Срабатывает при** удалении судей из таблицы `judge`.
   - **Действие:** удаляет судей из `judge_statistics`.
   - **Код:** листинг 8.

```
1   create or replace function update_statistics_on_judge_delete()
2   returns trigger as
3   begin
4       delete from judge_statistics where id_judge = old.id_judge;
5       return old;
6   end;
7    language plpgsql;
8
9   create trigger after_judge_delete
10  after delete on judge
11  for each row
12  execute function update_statistics_on_judge_delete();
```

Рис. 8. Код запроса для создания триггера `after_judge_delete` и функции `update_statistics_on_judge_delete`.

3. `after_judge_request_insert`

   - **Срабатывает при** добавлении заявки судьи на участие в соревновании в таблицу `judge_request`.
   - **Действие:** увеличивает на единицу счётчик с количеством соревнований
     в `judge_statistics` для судьи, от имени которого была добавлена заявка.
   - **Код:** листинг 9.

```
1   create or replace function update_statistics_on_judge_request_insert()
2   returns trigger as
3   begin
4       update judge_statistics
5       set competition_count = competition_count + 1
6       where id_judge = new.id_judge;
7       return new;
8   end;
9    language plpgsql;
10
11  create trigger after_judge_request_insert
12  after insert on judge_request
13  for each row
```

```
14  execute function update_statistics_on_judge_request_insert();
```

Рис. 9. Код запроса для создания триггера `after_judge_request_insert` и функции `update_statistics_on_judge_request_insert`.

4. `after_judge_request_delete`

   - **Срабатывает при** удалении заявки судьи на участие в соревновании из таблицы `judge_request`.
   - **Действие:** уменьшает на единицу счётчик с количеством соревнований в `judge_statistics` для судьи, от имени которого была добавлена заявка.
   - **Код:** листинг 10.

```
1   create or replace function update_statistics_on_judge_request_delete()
2   returns trigger as
3   begin
4       update judge_statistics
5       set competition_count = competition_count - 1
6       where id_judge = old.id_judge;
7       return old;
8   end;
9    language plpgsql;
10
11  create trigger after_judge_request_delete
12  after delete on judge_request
13  for each row
14  execute function update_statistics_on_judge_request_delete();
```

Рис. 10. Код запроса для создания триггера `after_judge_request_delete` и функции `update_statistics_on_judge_request_delete`.

5. `after_judge_request_update`

   - **Срабатывает при** изменении заявки судьи на участие в соревновании в таблице `judge_request`.
   - **Действие:** уменьшает на единицу счётчик с количеством соревнований для
     в `judge_statistics` для судьи, от имени которого была добавлена заявка.
   - **Код:** листинг 11.

```
1   create or replace function update_statistics_on_judge_request_update()
2   returns trigger as
3   begin
4       -- уменьшение счётчика у старого судьи
5       update judge_statistics
6       set competition_count = competition_count - 1
7       where id_judge = old.id_judge;
8
9       -- увеличение счётчика у нового судьи
10      update judge_statistics
```

```
11      set competition_count = competition_count + 1
12      where id_judge = new.id_judge;
13
14      return new;
15  end;
16   language plpgsql;
17
18  create trigger after_judge_request_update
19  after update on judge_request
20  for each row
21  execute function update_statistics_on_judge_request_update();
```

Рис. 11. Код запроса для создания триггера `after_judge_request_update` и функции `update_statistics_on_judge_request_update`.

После создания таблицы `judge_statistics` и всех триггеров, база данных была заполнена заново с помощью скрипта, разработанного ещё в прошлом семестре, для того, чтобы таблица содержала актуальные данные. Первые десять строк таблицы `judge_statistics` демонстрируются на Рис. 12.

id_judge [PK] integer	competition_count integer
126	23
131	15
140	26
120	21
117	24
141	24
107	23
127	21
111	19
148	29

Рис. 12. Первые десять записей в таблице `judge_statistics`.

2.3 Работа 3: Разграничение прав доступа

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

Формулировка задачи: Создать пользователя `readonly_user` и `edit_user`. 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. 13. По мимо стандартных прав на операции `insert`, `update`, `delete`,

пользователю `readonly_user` 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. Последовательности 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 1.

```
1  create user readonly_user with password '123';
2  grant usage on schema public to readonly_user;
3  grant select on sportsman_requests_series_count to readonly_user;
4
5  create user edit_user with password '123';
6  grant usage on schema public to edit_user;
7  grant select on sportsman_requests_series_count to edit_user;
8  grant select, insert, update, delete on sportsman to edit_user;
9  grant select, insert, update, delete on participant_request to edit_user;
10 grant select, insert, update, delete on shot_series to edit_user;
11 grant usage, select on sequence sportsman_id_sportsman_seq to edit_user;
12 grant usage, select on sequence participant_request_id_participant_request_seq to edit_user;
13 grant usage, select on sequence shot_series_id_shot_series_seq to edit_user;
```

Рис. 13. Код запросов для создания пользователей и выдачи им определённых прав.

Таблица 1. Сравнение результатов действий, выполненных от пользователей `readonly_user` и `edit_user`.

№	**edit_user**	**readonly_user**
1	Просмотр содержимого представления. `select * from sportsman_requests_series_count`	
	`Successfully run. Total query runtime: 272 msec.` `100 rows affected.`	`Successfully run. Total query runtime: 272 msec.` `100 rows affected.`

2	Изменение данных через представление.
	`insert into sportsman_requests_series_count (id_sportsman, pr_count, ss_count)`
	`values (100, 50, 25)`	

	`ERROR: Представления с GROUP BY не обновляются автоматически.вставить данные в представление "sportsman_requests_series_count" нельзя`
	
	`ОШИБКА: вставить данные в представление "sportsman_requests_series_count" нельзя`
	`SQL state: 55000`
	`Detail: Представления с GROUP BY не обновляются автоматически.`
	`Hint: Чтобы представление допускало добавление данных, установите триггер INSTEAD OF INSERT или безусловное правило ON INSERT DO INSTEAD.`
		`ERROR: Представления с GROUP BY не обновляются автоматически.вставить данные в представление "sportsman_requests_series_count" нельзя`
		
		`ОШИБКА: вставить данные в представление "sportsman_requests_series_count" нельзя`
		`SQL state: 55000`
		`Detail: Представления с GROUP BY не обновляются автоматически.`
		`Hint: Чтобы представление допускало добавление данных, установите триггер INSTEAD OF INSERT или безусловное правило ON INSERT DO INSTEAD.`

Таблица 1. Сравнение результатов действий, выполненных от пользователей `readonly_user` и `edit_user` (продолжение).

№	**edit_user**	**readonly_user**
3	Чтение данных из таблицы, несвязанной с представлением. `select * from judge`	
	ERROR: нет доступа к таблице judge ОШИБКА: нет доступа к таблице judge SQL state: 42501	ERROR: нет доступа к таблице judge ОШИБКА: нет доступа к таблице judge SQL state: 42501
4	Чтение данных из таблицы `sportsman`. `select * from sportsman`	
	Successfully run. Total query runtime: 69 msec. 100 rows affected.	ERROR: нет доступа к таблице sportsman ОШИБКА: нет доступа к таблице sportsman SQL state: 42501
5	Чтение данных из таблицы `participant_request`. `select * from participant_request`	
	Successfully run. Total query runtime: 125 msec. 9257 rows affected.	ERROR: нет доступа к таблице participant_request ОШИБКА: нет доступа к таблице participant_request SQL state: 42501
6	Чтение данных из таблицы `shot_series`. `select * from shot_series`	
	Successfully run. Total query runtime: 286 msec. 241357 rows affected.	ERROR: нет доступа к таблице shot_series ОШИБКА: нет доступа к таблице shot_series SQL state: 42501

7	Добавление данных в таблицу `participant_request`.
	`insert into shot_series (score, photo, id_participant_request,`
	`id_result_in_stage, id_judge_request)`
	`values (10, 'path/to/photo.jpg', 16784, 67368, 3403)`	

	INSERT 0 1 Query returned successfully in 36 msec.	ERROR: нет доступа к таблице shot_series ОШИБКА: нет доступа к таблице shot_series SQL state: 42501

8	Добавление данных в таблицу, несвязанную с представлением.
	`insert into judge (name, category, region, federation)`
	`values ('Иван Иванов', '1 категория', 'Санкт-Петербург', 'Федерация')`	

	ERROR: нет доступа к таблице judge ОШИБКА: нет доступа к таблице judge SQL state: 42501	ERROR: нет доступа к таблице judge ОШИБКА: нет доступа к таблице judge SQL state: 42501

2.4 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 задачи: Создать функцию `convert_to_initials`, которая будет получать на вход строки с ФИО, а возвращать строку с инициалами. Создать процедуру `create_participant_request`, которая по заданной информации о соревновании, спортсмене и дивизионе будет создавать заявку на участие в соревновании.

Код определения функции `convert_to_initials` представлен на Рис. 14. Функция принимает на вход три параметра типа `varchar`: `p_name` – имя, `p_surname` – фамилия, `p_patronymic` – отчество. Возвращает также `varchar` – строку с инициалами. Если имя или фамилия 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. 15, а результат его выполнения на Рис. 16.

```
1  create or replace function convert_to_initials(
2      p_name varchar,
3      p_surname varchar,
4      p_patronymic varchar
5  ) returns varchar as
6  begin
7      if p_name is null or p_name = '' or p_surname is null or p_surname = '' then
8          return '';
9      end if;
10
11     if p_patronymic is null or p_patronymic = '' then
12         return concat(substring(p_name from 1 for 1), '. ', p_surname);
13     end if;
14
15     return concat(substring(p_patronymic from 1 for 1), '. ', substring(p_name from 1 for 1), '. ',
         p_surname);
16 end;
17  language plpgsql;
```

Рис. 14. Код определения функции `convert_to_initials`.

```
1  select
2      id_judge,
3      convert_to_initials (name, surname, patronymic),
4      name,
5      surname,
6      patronymic
7  from
8      judge
```

Рис. 15. Код запроса с использованием функции `convert_to_initials`.

Код определения процедуры `create_participant_request` представлен на Рис. 17. Процедура принимает на вход пять параметров типа `varchar`: `p_name` – имя спортсмена, `p_surname` – фамилия спортсмена, `p_gender` – пол спортсмена, `p_competition_title` – название соревнования, `p_division_title` – название дивизиона. Внутри процедуры сначала проверяется существование дивизиона в базе

id_judge [PK] integer	convert_to_initials character varying	name character varying (100)	surname character varying (100)	patronymic character varying (100)
201	Л. С. Егорова	София	Егорова	Львовна
202	Г. А. Чистяков	Артём	Чистяков	Глебович
203	И. Шестаков	Илья	Шестаков	
204	Я. А. Александрова	Александра	Александрова	Ярославовна
205			Сафонов	Данилович
206	З. В. Григорьев	Владимир	Григорьев	Захарович
207				Алексеевна
208	В. В. Соколов	Василий	Соколов	Владимирович
209				
210	Я. М. Федоров	Макар	Федоров	Ярославович

Рис. 16. Первые десять записей результата выполнения запроса с применением функции `convert_to_initials`.

данных по переданному названию. Если дивизион не найден, выбрасывается ошибка. Затем проверяется существование соревнования. Если соревнование не существует, оно создаётся. Далее проверяется, существует ли спортсмен с указанными именем, фамилией и полом. Если спортсмен не найден, то создаётся новая запись о спортсмене. В конце создаётся заявка участника с указанием данных спортсмена, соревнования и дивизиона.

```
1  create or replace procedure create_participant_request(
2      p_name varchar(100),
3      p_surname varchar(100),
4      p_gender varchar(10),
5      p_competition_title varchar(100),
6      p_division_title varchar(100)
7  )
8  language plpgsql
9  as
10 declare
11     v_id_sportsman integer;
12     v_id_competition integer;
13     v_id_division integer;
14 begin
15     -- проверка существования дивизиона
16     select id_division into v_id_division
17     from division
18     where title = p_division_title
19     limit 1;
20
21     if not found then
22         raise exception 'Дивизиона %_не_существует', p_division_title;
23     end if;
24
25     -- проверка существования соревнования
26     select id_competition into v_id_competition
27     from competition
28     where title = p_competition_title
29     limit 1;
```

```
30
31      -- если соревнование не существует, создать его
32      if not found then
33          insert into competition (title)
34          values (p_competition_title)
35          returning id_competition into v_id_competition;
36      end if;
37
38      -- проверка существования спортсмена
39      select id_sportsman into v_id_sportsman
40      from sportsman
41      where name = p_name and surname = p_surname and gender = p_gender
42      limit 1;
43
44      -- если спортсмен не существует, создать его
45      if not found then
46          insert into sportsman (name, surname, gender)
47          values (p_name, p_surname, p_gender)
48          returning id_sportsman into v_id_sportsman;
49      end if;
50
51      -- создание заявки участника
52      insert into participant_request (name, surname, gender, is_registered, id_competition,
          id_sportsman, id_division)
53      values (p_name, p_surname, p_gender, false, v_id_competition, v_id_sportsman, v_id_division)
          ;
54  end;
55  ;
```

Рис. 17. Код определения процедуры `create_participant_request`.

2.5 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:

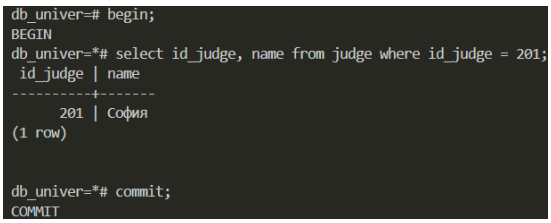
```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

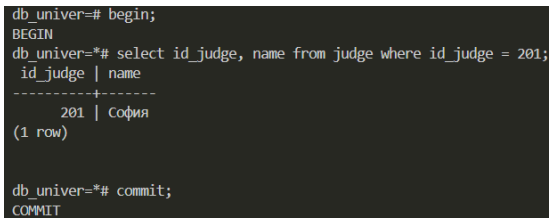
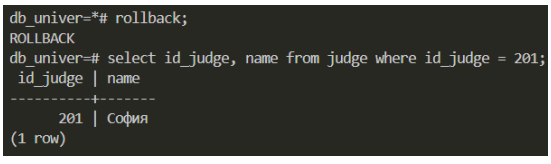
На этом уровне изоляции исключается артефакт «Грязное чтение», однако сохраняются артефакты «Неповторяемое чтение» и «Фантомы».

В таблице 2 представлены транзакции, на примере которых демонстрируется отсутствие артефакта «Грязное чтение».

Таблица 2. Транзакции для демонстрации отсутствия артефакта «Грязное чтение».

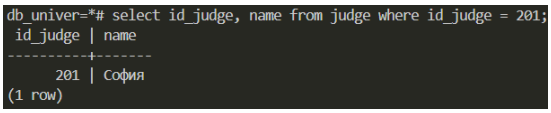
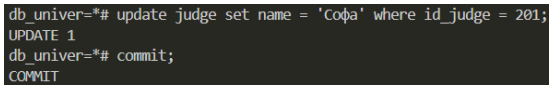
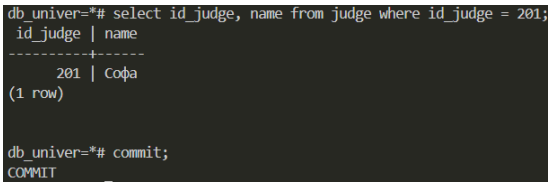
<table>
<tr><th>t</th><th>Транзакция 1</th><th>Транзакция 2</th></tr>
<tr><td rowspan="2"></td><td colspan="2">В первой транзакции обновляется запись в таблице <code>judge</code>. В момент после обновления вторая транзакция читает эту же запись из <code>judge</code>, однако в связи с отсутствием артефакта «Грязное чтение» она видит исходную таблицу до обновлений первой транзакции.</td></tr>
<tr><td colspan="2"><code>SET TRANSACTION ISOLATION LEVEL READ COMMITTED;</code></td></tr>
<tr><td></td><td>Запуск транзакции 1
<code>begin;</code></td><td>Запуск транзакции 2
<code>begin;</code></td></tr>
<tr><td>t_1</td><td>Изменение имени судьи с <code>id_judge = 201</code>

<code>update judge set name = 'Софа'</code>
<code>where id_judge = 201;</code>
 
</td><td></td></tr>
<tr><td>t_2</td><td></td><td>Получение имени судьи с <code>id_judge = 201</code>
и завершение транзакции

<code>select id_judge, name from judge</code>
<code>where id_judge = 201;</code>
<code>commit;</code>
 
</td></tr>
<tr><td>t_3</td><td>Отмена внесённых изменений транзакцией 1
<code>rollback;</code>
 
</td><td></td></tr>
</table>

В таблице 3 представлены транзакции, на примере которых демонстрируется наличие артефакта «Неповторяемое чтение».

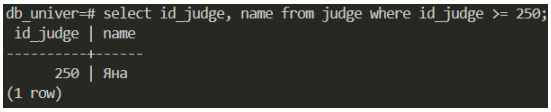
Таблица 3. Транзакции для демонстрации наличия артефакта «Неповторяемое чтение».

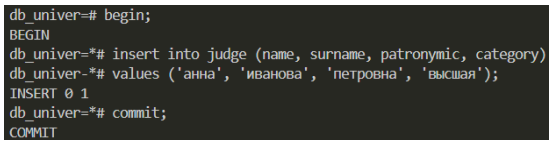
<table>
<tr><th>t</th><th>Транзакция 1</th><th>Транзакция 2</th></tr>
<tr><td></td><td colspan="2">В первой транзакции происходит чтение записи из таблицы judge. После чего эта же запись обновляется во второй транзакции, изменения фиксируются и вторая транзакция успешно завершается. Затем первая транзакция повторно считывает эту запись и получает обновлённые данные, а не исходные, из-за наличия артефакта «Неповторяемое чтение».</td></tr>
<tr><td></td><td colspan="2">SET TRANSACTION ISOLATION LEVEL READ COMMITTED;</td></tr>
<tr><td></td><td>Запуск транзакции 1
begin;</td><td>Запуск транзакции 2
begin;</td></tr>
<tr><td>t_1</td><td>Получение имени судьи с id_judge = 201
select id_judge, name from judge
where id_judge = 201;
  </td><td></td></tr>
<tr><td>t_2</td><td></td><td>Изменение имени судьи с id_judge = 201
и фиксация транзакции
update judge set name = ’Софа’
where id_judge = 201;
commit;
  </td></tr>
<tr><td>t_3</td><td>Получение имени судьи с id_judge = 201
и фиксация транзакции
select id_judge, name from judge
where id_judge = 201;
commit;
  </td><td></td></tr>
</table>

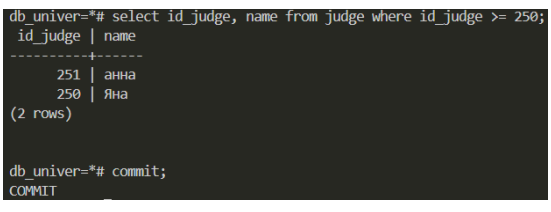
В таблице 4 представлены транзакции, на примере которых демонстрируется наличие артефакта «Фантомы».

Таблица 4. Транзакции для демонстрации наличия артефакта «Фантомы».

<table>
<tr><td>t</td><td>Транзакция 1</td><td>Транзакция 2</td></tr>
<tr><td rowspan="3"></td><td colspan="2">В первой транзакции происходит чтение записей из таблицы <code>judge</code> с <code>id_judge >= 250</code>. После чего в эту таблицу добовляется запись во второй транзакции, изменения фиксируются и вторая транзакция успешно завершается. Затем первая транзакция повторно получает записи с <code>id_judge >= 250</code>. В этот раз она также получает данные о записи, добавленной второй транзакцией, из-за наличия артефакта «Фантомы».</td></tr>
<tr><td colspan="2"><code>SET TRANSACTION ISOLATION LEVEL READ COMMITTED;</code></td></tr>
<tr><td>Запуск транзакции 1
<code>begin;</code></td><td>Запуск транзакции 2
<code>begin;</code></td></tr>
<tr><td>t_1</td><td>Получение имён судей с <code>id_judge >= 250</code>

<code>select id_judge, name from judge</code>
<code>where id_judge >= 250;</code>
  </td><td></td></tr>
<tr><td>t_2</td><td></td><td>Добавление нового судьи и фиксация транзакции

<code>insert into judge (name, surname,</code>
<code>patronymic, category)</code>
<code>values ('анна', 'иванова', 'петровна',</code>
<code>'высшая');</code>
<code>commit;</code>
  </td></tr>
<tr><td>t_3</td><td>Получение имён судей с <code>id_judge >= 250</code>

<code>select id_judge, name from judge</code>
<code>where id_judge >= 250;</code>
<code>commit;</code>
  </td><td></td></tr>
</table>

Заключение

В ходе освоения данного курса было выполнено пять лабораторных работ:

1. Создано представление, инкапсулирующее запрос. Продемонстрирована невозможность модификации представления; написан запрос, использующий в себе созданное представление.
2. Создана таблица подсчёта количества соревнований для каждого судьи. Также созданы триггеры, автоматизирующие сбор статистики в таблице.
3. Созданы два пользователя с различными правами доступа. Первый пользователь наделён правами только на просмотр представления, а второй наделён правами просмотра, вставки, удаления и обновления данных во всех таблицах, участвующих в представлении. Продемонстрировано поведение СУБД при различных операциях для каждого пользователя, в том числе при недопустимых.
4. Созданы процедура и функция. Функция принимает на вход фамилию, имя и отчество человека и возвращает фамилию и его инициалы.
5. Управление транзакциями. Задан уровень изоляции транзакций как Read Committed и 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 пятисот строк кода.

Работа была выполнена в системе управления базами данных PostgreSQL 16.2.

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

Список литературы

[1] MySQL Documentation URL: https://dev.mysql.com/doc/, Дата обращения: 01.11.2024

[2] PostgreSQL documentation URL: https://www.postgresql.org/docs/, Дата обращения: 01.11.2024