

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ПЕТРА ВЕЛИКОГО»

Институт Компьютерных наук и кибербезопасности
Высшая школа искусственного интеллекта
Направление 02.03.01 Математика и Компьютерные науки

Отчёт по лабораторным работам
По дисциплине
«Методы проектирования баз данных»
4 курс, группа: 5130201/00101

Студент: _____

Боева Анастасия Владимировна

Преподаватель: _____

Попов Сергей Геннадьевич

Содержание

Введение	3
1 Постановка задачи	4
2 База данных	5
3 Задание 1: View	7
4 Задание 2: Триггер	8
5 Задание 3: Права пользователей	12
6 Задание 4: Создание процедур и функций	14
6.1 Процедура	14
6.2 Функция	15
7 Задание 5: Транзакции	17
Заключение	19
Список литературы	20

Введение

При разработке баз данных может возникнуть потребность в большем функционале, чем запросы к таблицам. К примеру, точно настроить права доступа к таблицам базы данных, предоставив доступ исключительно к `view`, а не к исходным таблицам. При разграничении прав доступа и многопользовательском подходе к базе данных невозможно обойти транзакции, а значит уровни изоляции с возникающими феноменами. Либо же написание пользовательских функций, привязанных к конкретной базе данных и конкретному пользователю. Возможно есть необходимость в автономном обновлении полей некоторых таблиц или ведение статистики, для этого нужно будет применить триггеры.

Данная работа содержит лабораторные работы, направленные на изучение `view`, разграничения прав доступа, транзакций, функций и триггеров.

В данном отчёте описан результат выполнения комплекса лабораторных работ, расширяющих функциональные возможности базы данных мирового рейтинга футболистов в ценовой характеристике, которая была разработана в течение предыдущего семестрового курса «Теоретические основы баз данных».

В рамках комплекса работ № 1 - 4 необходимо изучить и реализовать в СУБД представления (`VIEW`), пользовательские функции и хранимые процедуры, процедуры на основе триггеров, а также исследовать возможности и методы администрирования прав доступа пользователей базы данных.

1 Постановка задачи

В рамках выполнения лабораторных работ было необходимо:

1. Создать `view` и продемонстрировать выборку из неё;
2. Создать `trigger` и продемонстрировать его работу;
3. Создать двух пользователей с разным набором прав, продемонстрировать их возможности.

Первый пользователь должен иметь только права на чтение `view` из пункта 1.

Второй пользователь должен иметь права на просмотр `view` из пункта 1 и на модификацию таблиц, связанных с этой `view`;

4. Создать пользовательскую `function` и `procedure` и продемонстрировать работу каждой;
5. Настроить уровень изоляции и проверить работу феномена при работе с таблицами.

2 База данных

Для выполнения данных лабораторных работ использовалась база данных формирования мирового рейтинга футболистов в ценовой характеристике, созданная в рамках курсовой работы по курсу «Теоретические основы баз данных». База данных состоит из 12 таблиц, 7 из которых словари. Связи между таблицами реализованы с помощью внешних ключей. Используется система управления базами данных MySQL 8. Сервер развернут локально. На Рис. 1 представлена схема базы данных на английском языке.

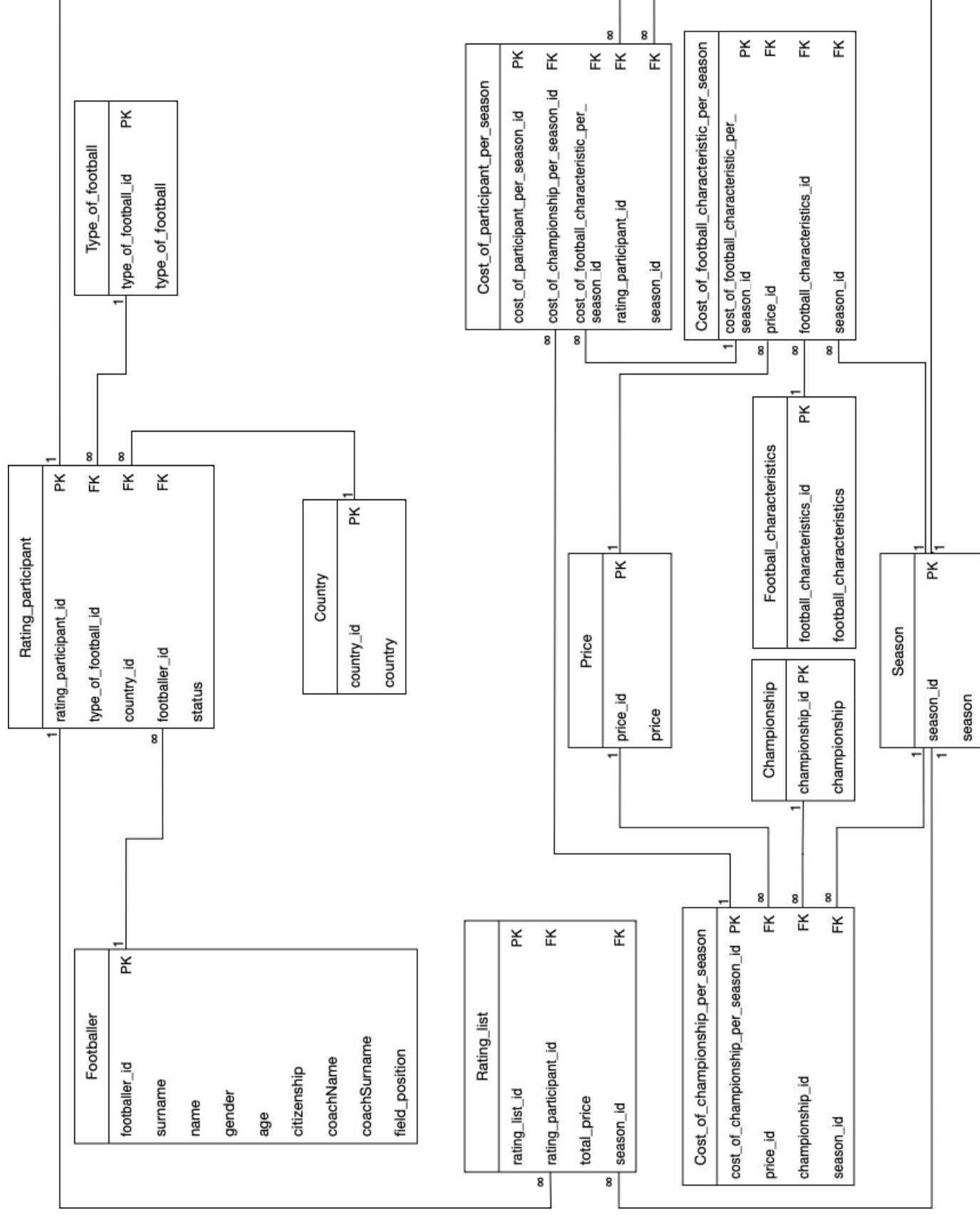


Рис. 1: Схема базы данных

3 Задание 1: View

Задача:

Разработать VIEW.

Формулировка задачи:

Для каждого футболиста определить позицию в рейтинге и цену.

Сформировать таблицу, в которой содержит ФИ футболиста, его позицию в рейтинге и сумму.

SQL-код создания View:

```
CREATE OR REPLACE VIEW rating_info AS
SELECT Footballer.footballer_id,
       Rating_list.position,
       Rating_list.total_price
FROM Footballer
JOIN Rating_participant ON Rating_participant.footballer_id =
    ↪ Footballer.footballer_id
JOIN Rating_list ON Rating_list.rating_participant_id =
    ↪ Rating_participant.rating_participant_id;
```

Объяснение View:

Для получения позиции в рейтинге и соответствующей цены для каждого футболиста необходимо связать таблицу `Footballer` с таблицей `Rating_list` через таблицу `Rating_participant`.

Код SQL-запроса с использованием View:

```
SELECT Footballer.surname,
       Footballer.name,
       rating_info.position,
       rating_info.total_price
FROM Footballer
JOIN rating_info ON Footballer.footballer_id = rating_info.footballer_id
WHERE rating_info.footballer_id = '5';
```

Объяснение запроса:

Для отображения ФИ футболиста, позиции в рейтинге и соответствующей цены для каждого необходимо связать таблицу `Footballer` с представлением `rating_info` по полю `footballer_id`. Из таблицы `Footballer` выбираем ФИ, из представления `rating_info` выбираем позицию в рейтинге для фигуриста с данным ФИО.

Результат выполнения запроса:

На Рис.2 представлен результат выполнения запроса с использованием View для футболиста с `id = 5`.

surname	name	position	total_price
Gardner	Albert	120	1850
1 row in set (0,00 sec)			

Рис. 2: Результат выполнения запроса с использованием View

4 Задание 2: Триггер

Задача:

Разработать TRIGGER.

Формулировка задачи:

Подсчитать для каждого участника рейтинга количество обновлений его записи.

1. Необходимо создать таблицу, в которой будет храниться результат работы триггера:

- Таблица, в которой хранятся все участники рейтинга для каждого из которых подсчитывается количество обновлений записи - `upd_count`.

SQL-код создания таблицы

```
CREATE TABLE Number_of_updates (  
  number_of_updates_id INT NOT NULL AUTO_INCREMENT,  
  upd_count INT,  
  PRIMARY KEY(number_of_updates_id));
```

2. Создадим триггер для таблицы `Number_of_updates`, который срабатывает после обновления записи в таблице `Rating_list`.

SQL-код создания триггера `update_upd_count`:

```
DELIMITER //  
CREATE TRIGGER update_upd_count AFTER  
UPDATE ON Rating_list  
FOR EACH ROW BEGIN  
  UPDATE Number_of_updates  
  SET upd_count = upd_count + 1  
  WHERE number_of_updates_id = NEW.number_of_updates_id;  
END; //  
DELIMITER ;
```

Объяснение:

Когда происходит UPDATE в таблице `Rating_list`, триггер выполняет оператор UPDATE для обновления таблицы `Number_of_updates`. `DELIMITER //` - указывает на то, что меняется разделитель команд с точки с запятой (;) на два слепа (//). Это необходимо для правильного определения начала и конца триггера. Далее начинается блок кода, который будет выполняться для каждой строки, на которую было выполнено обновление. После этого обновляется таблица `Number_of_updates`, с увеличенным на 1 значением поля `upd_count`, где значение поля `number_of_updates_id` соответствует новому значению, полученному в результате обновления в таблице `Rating_list`.

3. Создадим триггер для таблицы `Number_of_updates`, который срабатывает после вставки записи в таблице `Rating_list`. Для каждой вставленной строки триггер будет выполнен отдельно.

SQL-код создания триггера `insert_upd_count`:

```
DELIMITER //  
CREATE TRIGGER insert_upd_count AFTER  
INSERT ON Rating_list  
FOR EACH ROW BEGIN  
  INSERT INTO Number_of_updates(number_of_updates_id, upd_count)  
  VALUES (NEW.number_of_updates_id, 0);  
END; //  
DELIMITER ;
```


Объяснение:

Когда происходит INSERT в таблице Rating_list, триггер выполняет оператор UPDATE для обновления таблицы Number_of_updates. Значение number_of_updates_id берется из вставленной строки в таблицу Rating_list, а значение 0 устанавливается в столбец upd_count. Таким образом, этот триггер создает новую запись с идентификатором number_of_updates_id и начальным значением 0 для каждой вставленной строки в таблицу Rating_list.

4. Создадим триггер для таблицы Number_of_updates, который срабатывает после удаления записи в таблице Rating_list. Для каждой удаленной строки триггер будет выполнен отдельно.

SQL-код создания триггера delete_upd_count:

```
DELIMITER //
CREATE TRIGGER delete_upd_count AFTER
DELETE ON Rating_list
FOR EACH ROW BEGIN
UPDATE Number_of_updates
SET upd_count = -1
WHERE number_of_updates_id = OLD.number_of_updates_id;
END; //
DELIMITER ;
```

Объяснение:

Триггер будет выполняться после каждого оператора DELETE в таблице Rating_list. Когда происходит DELETE в таблице Rating_list, триггер выполняет оператор UPDATE для обновления таблицы Number_of_updates. В этом операторе указывается условие WHERE, которое проверяет, что значение столбца number_of_updates_id равно значению столбца number_of_updates_id в удаленной строке таблицы Rating_list. Затем происходит обновление столбца upd_count в таблице Number_of_updates путем установки значения -1. Таким образом, этот триггер обновляет значение столбца upd_count на -1 в таблице Number_of_updates для каждой удаленной строки в таблице Rating_list.

Проверим работу триггеров.

На Рис. 3 частично представлено исходное содержимое таблицы Number_of_updates.

number_of_updates_id	upd_count
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0

Рис. 3: Таблица Number_of_updates

Изменим в таблице Rating_list запись, у которой поле rating_list_id будет равно 2, а значение поля season_id изменяется на 5.

```
UPDATE Rating_list
SET season_id = 5
WHERE rating_list_id = 2;
```

На Рис. 4 представлено часть содержимого таблицы `Number_of_updates` после обновления записи в таблице `Rating_list`.

В таблице `Number_of_updates` значение `upd_count` для поля `number_of_updates_id = 2` увеличилось на 1 и стало равно 1.

```
mysql> select * from Number_of_updates;
```

number_of_updates_id	upd_count
1	0
2	1
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0

Рис. 4: Таблица `Number_of_updates`

Вставим в таблицу `Rating_list` запись, у которой поле `number_of_updates_id` будет равно 1501.

```
INSERT INTO Rating_list VALUES(1502, 1, 1995, 5, 20, 1501);
```

На Рис. 5 представлено часть содержимого таблицы `Number_of_updates` после добавления записи в таблицу `Rating_list`.

В таблице `Number_of_updates` значение `upd_count` для поля `number_of_updates_id = 1501` появилось и стало равно 0.

1491	0
1492	0
1493	0
1494	0
1495	0
1496	0
1497	0
1498	0
1499	0
1500	0
1501	1

1501 rows in set (0,00 sec)

Рис. 5: Таблица `Number_of_updates`

Удалим из таблицы `Rating_list` запись, у которой поле `rating_list_id` будет равно 2.

```
DELETE FROM Rating_list WHERE rating_list_id = 2;
```

На Рис. 6 представлено часть содержимого таблицы `Number_of_updates` после удаления записи в таблице `Rating_list`.

В таблице `Number_of_updates` значение `upd_count` для поля `number_of_updates_id = 2` стало равно -1.

```
mysql> select * from Number_of_updates;
```

number_of_updates_id	upd_count
1	2
2	-1
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0

Рис. 6: Таблица `Number_of_updates`

5 Задание 3: Права пользователей

Задача:

Разработать схему прав двух пользователей с разным набором прав.

Формулировка задачи:

Создаваемые пользователи должны иметь следующие права:

1. Имеет права только на чтение View `bd.rating_info`.
2. Имеет права на чтение View `bd.rating_info` и изменение всех таблиц, связанных с этой View: `bd.Footballer`, `bd.Rating_participant`, `bd.Rating_list`.

SQL-код создания пользователя 1 и назначения ему прав:

```
CREATE USER 'weak'@'localhost' IDENTIFIED BY 'aaa';  
GRANT SELECT ON bd.rating_info TO 'weak'@'localhost';
```

SQL-код создания пользователя 2 и назначения ему прав:

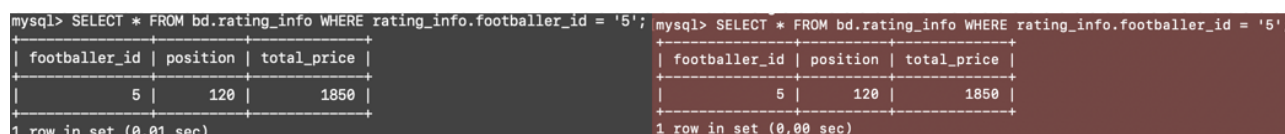
```
CREATE USER 'strong'@'localhost' IDENTIFIED BY 'aaaa';  
GRANT SELECT ON bd.rating_info TO 'strong'@'localhost';  
GRANT ALL PRIVILEGES ON bd.Footballer TO 'strong'@'localhost';  
GRANT ALL PRIVILEGES ON bd.Rating_participant TO 'strong'@'localhost';  
GRANT ALL PRIVILEGES ON bd.Rating_list TO 'strong'@'localhost';
```

Результаты выполнения запросов

Результат выполнения запроса на чтение View `bd.rating_info` для двух пользователей оказался одинаковым (Рис.7).

SQL-код запроса:

```
SELECT * FROM bd.rating_info WHERE rating_info.footballer_id = '5';
```



footballer_id	position	total_price
5	120	1850

1 row in set (0,01 sec)

Рис. 7: Чтение View. Пользователь weak - слева, пользователь strong - справа

Результат выполнения запроса на добавление записи в таблицу `bd.Footballer`, которая связана с View, оказался успешным для пользователя `strong` и не успешным для пользователя `weak` - выводится ошибка «INSERT command denied to user 'weak'@'localhost' for table 'footballer'» (Рис. 8).

SQL-код вставки новой записи:

```
INSERT INTO bd.Footballer VALUES (1501, 'Holmes ', 'Alex', 'Man', 1999, 'Abkhazia', '  
↪ Adderiy', 'Charles', 'Defender');
```

Проверим, что запись добавилась, и эти изменения отображаются у обоих пользователей. Значение поля `footballer_id` стало 1501, `position` - 5, `total_price` - 1995.

SQL-код запроса:

```
SELECT * FROM bd.rating_info WHERE rating_info.footballer_id = '1501';
```

```

mysql> INSERT INTO bd.Footballer VALUES (1501,'Holmes ','Alex','Man',1999,'Abkhazia','Adderiy','Charles','Defender');
ERROR 1142 (42000): INSERT command denied to user 'weak'@'localhost' for table 'footballer'
mysql> INSERT INTO Rating_participant VALUES(1501,2,100,1501,'amateur');
ERROR 1142 (42000): INSERT command denied to user 'weak'@'localhost' for table 'rating_participant'
mysql> INSERT INTO bd.Rating_list VALUES(1505, 1501, 1995, 5, 20);
ERROR 1142 (42000): INSERT command denied to user 'weak'@'localhost' for table 'rating_list'
mysql> SELECT * FROM bd.rating_info WHERE rating_info.footballer_id = '1501';
+-----+-----+-----+
| footballer_id | position | total_price |
+-----+-----+-----+
| 1501 | 5 | 1995 |
+-----+-----+-----+
1 row in set (0,00 sec)

mysql> INSERT INTO bd.Footballer VALUES (1501,'Holmes ','Alex','Man',1999,'Abkhazia','Adderiy','Charles','Defender');
Query OK, 1 row affected (0,03 sec)
mysql> SELECT * FROM bd.rating_info WHERE rating_info.footballer_id = '1501';
Empty set (0,02 sec)
mysql> INSERT INTO Rating_participant VALUES(1501,2 ,100,1501,'amateur');
Query OK, 1 row affected (0,01 sec)
mysql> INSERT INTO bd.Rating_list VALUES(1505, 1501, 1995, 5, 20);
Query OK, 1 row affected (0,00 sec)
mysql> SELECT * FROM bd.rating_info WHERE rating_info.footballer_id = '1501';
+-----+-----+-----+
| footballer_id | position | total_price |
+-----+-----+-----+
| 1501 | 5 | 1995 |
+-----+-----+-----+
1 row in set (0,01 sec)

```

Рис. 8: Добавление записи. Пользователь weak - слева, пользователь strong - справа

Результат выполнения запроса на добавление записи в таблицу `bd.Country`, которая не связана с View, оказался не успешным как для пользователя `weak`, так и для пользователя `strong` (Рис. 9).

Выводятся ошибки:

```

«INSERT command denied to user 'weak'@'localhost' for table 'country'»
«INSERT command denied to user 'strong'@'localhost' for table 'country'»
SQL-код вставки новой записи:
INSERT INTO bd.Country VALUES(183, 'assyria');

```

```

mysql> INSERT INTO bd.Country VALUES(183, 'assyria');
ERROR 1142 (42000): INSERT command denied to user 'weak'@'localhost' for table 'country'

mysql> INSERT INTO bd.Country VALUES(183, 'assyria');
ERROR 1142 (42000): INSERT command denied to user 'strong'@'localhost' for table 'country'

```

Рис. 9: Добавление записи. Пользователь weak - слева, пользователь strong - справа

В сравнительной таблице (Табл. 1) приведены действия и результаты созданных пользователей.

Действие	Пользователь strong	Пользователь weak
Просмотр view <code>bd.rating_info</code>	1 row in set (0,00 sec)	1 row in set (0,01 sec)
Добавление записи в таблицу, связанную с view: <code>bd.Footballer</code>	Query OK, 1 row affected (0,03 sec)	INSERT command denied to user 'weak'@'localhost' for table 'footballer'
Обновление записи в таблице, связанной с view: <code>bd.Footballer</code>	Query OK, 1 row affected (0,01 sec) Rows matched: 1 Changed: 1 Warnings: 0	UPDATE command denied to user 'weak'@'localhost' for table 'footballer'
Удаление записи из таблицы, связанной с view: <code>bd.Footballer</code>	Query OK, 1 row affected (0,02 sec)	DELETE command denied to user 'weak'@'localhost' for table 'footballer'
Добавление записи в таблицу, не связанную с view: <code>bd.Country</code>	INSERT command denied to user 'strong'@'localhost' for table 'country'	INSERT command denied to user 'weak'@'localhost' for table 'country'
Обновление записи в таблице, не связанной с view: <code>bd.Country</code>	UPDATE command denied to user 'strong'@'localhost' for table 'country'	UPDATE command denied to user 'weak'@'localhost' for table 'country'
Удаление записи из таблицы, не связанной с view: <code>bd.Country</code>	DELETE command denied to user 'strong'@'localhost' for table 'country'	DELETE command denied to user 'weak'@'localhost' for table 'country'

Таблица 1: Права пользователей

6 Задание 4: Создание процедур и функций

6.1 Процедура

Задача:

Реализовать PROCEDURE.

Формулировка задачи:

По заданному id чемпионата показать все стоимости этого чемпионата в сезон.

SQL-код создания процедуры:

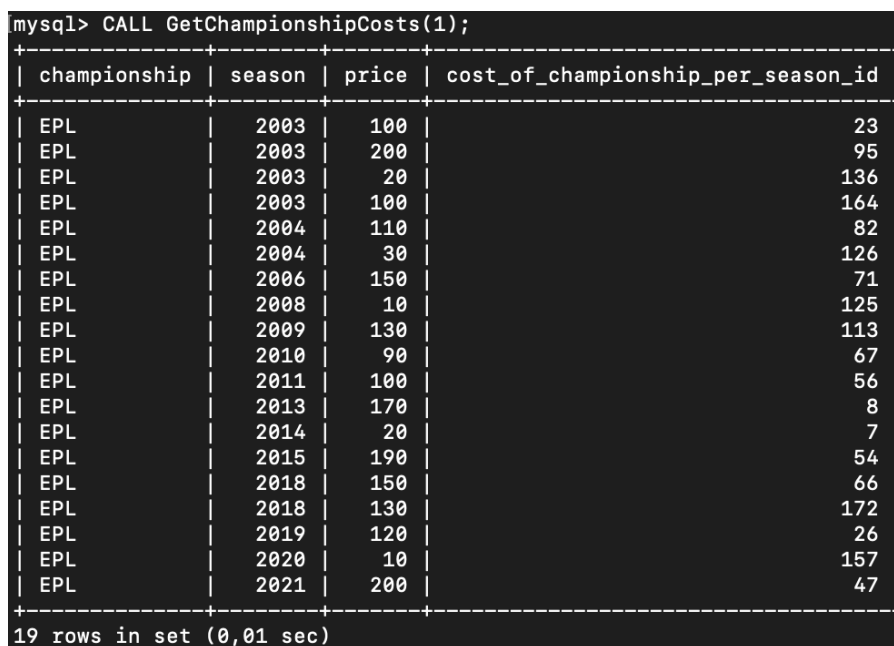
```
DELIMITER //
CREATE PROCEDURE GetChampionshipCosts(IN championshipId INT) BEGIN
SELECT c.championship,
      s.season,
      p.price
FROM Cost_of_championship_per_season ccps
INNER JOIN Price p ON ccps.price_id = p.price_id
INNER JOIN Championship c ON ccps.championship_id = c.championship_id
INNER JOIN Season s ON ccps.season_id = s.season_id
WHERE c.championship_id = championshipId;
ORDER BY s.season ASC;
END //
DELIMITER ;
```

Выполним процедуру для чемпионата с championship_id = 1.

SQL-код запроса:

```
CALL GetChampionshipCosts(1);
```

На Рис.10 представлен результат выполнения процедуры для чемпионата с championship_id = 1.



```
mysql> CALL GetChampionshipCosts(1);
```

championship	season	price	cost_of_championship_per_season_id
EPL	2003	100	23
EPL	2003	200	95
EPL	2003	20	136
EPL	2003	100	164
EPL	2004	110	82
EPL	2004	30	126
EPL	2006	150	71
EPL	2008	10	125
EPL	2009	130	113
EPL	2010	90	67
EPL	2011	100	56
EPL	2013	170	8
EPL	2014	20	7
EPL	2015	190	54
EPL	2018	150	66
EPL	2018	130	172
EPL	2019	120	26
EPL	2020	10	157
EPL	2021	200	47

19 rows in set (0,01 sec)

Рис. 10: Результат выполнения процедуры для чемпионата с championship_id = 1.

На Рис.11 представлен результат выполнения запроса для чемпионата с championship_id = 1 и cost_of_championship_per_season_id = 7.

```
mysql> SELECT * FROM Cost_of_championship_per_season where championship_id=1 and cost_of_championship_per_season_id=7;
+-----+-----+-----+-----+
| cost_of_championship_per_season_id | price_id | championship_id | season_id |
+-----+-----+-----+-----+
| 7 | 2 | 1 | 9 |
+-----+-----+-----+-----+
1 row in set (0,00 sec)
```

Рис. 11: Результат выполнения запроса для чемпионата с `championship_id = 1` и `cost_of_championship_per_season_id = 7`.

На Рис.12 представлен результат выполнения запроса для цены с `price_id = 2`.

```
mysql> SELECT * FROM Price where price_id=2;
+-----+-----+
| price_id | price |
+-----+-----+
| 2 | 20 |
+-----+-----+
1 row in set (0,00 sec)
```

Рис. 12: Результат выполнения запроса для цены с `price_id = 2`.

На Рис.13 представлен результат выполнения запроса для сезона с `season_id = 9`.

```
mysql> SELECT * FROM Season where season_id=9;
+-----+-----+
| season_id | season |
+-----+-----+
| 9 | 2014 |
+-----+-----+
1 row in set (0,00 sec)
```

Рис. 13: Результат выполнения запроса для сезона с `season_id = 9`.

Ожидаемый результат совпал с действительным, что свидетельствует о работе процедуры в соответствии с ожидаемым поведением.

6.2 Функция

Задача:

Реализовать FUNCTION.

Формулировка задачи:

По заданному id объединить ФИ - фамилию и имя, футболиста. Вывести инициалы.

SQL-код создания функции:

```
SET GLOBAL log_bin_trust_function_creators = 1;
DROP FUNCTION IF EXISTS full_name;
DELIMITER //
CREATE FUNCTION `full_name` (footb_id INTEGER) RETURNS VARCHAR(101) BEGIN DECLARE
↪ full_name VARCHAR(101);
SELECT CONCAT(surname, ' ', LEFT(name, 1), '.') INTO full_name
FROM Footballer
WHERE footballer_id = footb_id; RETURN full_name; END //
DELIMITER ;
```

Выполним запрос для футболиста с `footballer_id = 1`.

SQL-код запроса:

```
SELECT *
FROM Footballer
WHERE footballer_id=1;
```

На Рис.14 представлен результат выполнения запроса для футболиста с `footballer_id = 1`.

```
mysql> SELECT * from Footballer where footballer_id=1;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| footballer_id | surname | name  | gender | age | citizenship | coachName | coachSurname | field_position |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1             | Garrison | Harold | Man    | 1933 | Costa       | Livingston | Herbert      | Defender       |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0,01 sec)
```

Рис. 14: Результат выполнения запроса для футболиста с `footballer_id = 1`.

Ожидаемый результат выполнения функции: `Garrison Harold`.

Выполним запрос для функции с `footb_id = 1`.

SQL-код запроса:

```
SELECT full_name(1);
```

На Рис.15 представлен результат выполнения запроса для футболиста с `footb_id = 1`.

```
[mysql> SELECT full_name(1);
+-----+
| full_name(1) |
+-----+
| Garrison H.   |
+-----+
1 row in set (0,00 sec)
```

Рис. 15: Результат выполнения запроса для футболиста с `footb_id = 1`.

Ожидаемый результат совпал с действительным, что свидетельствует о работе функции в соответствии с ожидаемым поведением.

7 Задание 5: Транзакции

Задача:

Настроить уровень изоляции на READ-COMMITTED.

Формулировка задачи:

Необходимо проверить наличие феномена DIRTY READ при работе с таблицами.

Проверка уровня изоляции.

SQL-код запроса:

```
SELECT @@transaction_ISOLATION;
```

На Рис.16 представлен результат выполнения запроса для определения текущего уровня изоляции.

```
mysql> SELECT @@transaction_ISOLATION;
+-----+
| @@transaction_ISOLATION |
+-----+
| REPEATABLE-READ        |
+-----+
1 row in set (0,01 sec)
```

Рис. 16: Результат выполнения запроса для определения текущего уровня изоляции

Изменение уровня изоляции на READ-COMMITTED.

SQL-код запроса:

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

На Рис.17 представлен результат изменения уровня изоляции.

```
mysql> SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
Query OK, 0 rows affected (0,00 sec)
```

Рис. 17: Результат изменения уровня изоляции

На Рис.18 представлен результат последовательного процесса выполнения действий каждым из пользователей и запросы, отображающий изменения.

root	strong
<pre>UPDATE bd.Rating_list SET total_price = '1330' WHERE rating_participant_id = '5';</pre> <pre>mysql> SELECT rating_list_id,position,total_price -> FROM bd.Rating_list WHERE rating_participant_id = '5'; +-----+-----+-----+ rating_list_id position total_price +-----+-----+-----+ 5 534 1330 +-----+-----+-----+ 1 row in set (0,00 sec)</pre>	<pre>SELECT rating_list_id,position,total_price FROM bd.Rating_list WHERE rating_participant_id = '5';</pre>
	<pre>mysql> SELECT rating_list_id,position,total_price -> FROM bd.Rating_list WHERE rating_participant_id = '5'; +-----+-----+-----+ rating_list_id position total_price +-----+-----+-----+ 5 534 1331 +-----+-----+-----+ 1 row in set (0,00 sec)</pre>
<pre>ROLLBACK;</pre> <pre>mysql> SELECT rating_list_id,position,total_price -> FROM bd.Rating_list WHERE rating_participant_id = '5'; +-----+-----+-----+ rating_list_id position total_price +-----+-----+-----+ 5 534 1331 +-----+-----+-----+ 1 row in set (0,00 sec)</pre>	<pre>COMMIT;</pre>
	<pre>mysql> SELECT rating_list_id,position,total_price -> FROM bd.Rating_list WHERE rating_participant_id = '5'; +-----+-----+-----+ rating_list_id position total_price +-----+-----+-----+ 5 534 1331 +-----+-----+-----+ 1 row in set (0,00 sec)</pre>

Рис. 18: Сравнительная таблица

В результате выполненных манипуляций в поле `total_price` сохранилось начальное значение 1331 \Rightarrow феномен *dirty read* не происходит.

Заключение

В ходе выполнения лабораторных работ было:

1. Создано **VIEW** и выполнены действия с ней.
VIEW в базе данных позволяют создавать представления, которые, например, могут быть доступны пользователям, у которых нет доступа к остальной базе. Таким образом можно реализовать определенный запрос, на который у пользователя не может быть прав. Однако каждое обращение к **VIEW** требует выполнения запроса, что увеличивает время работы, но уменьшает количество данных, которые нужно хранить и согласовывать.
2. Создан **TRIGGER** для подсчёта количества обновления записей таблицы **Rating_list**.
Триггер в базе данных является инструментом событийного программирования и позволяет реализовать некоторую логику прямо в базе данных. В данном примере при помощи триггера собиралась статистика, значения обновлялись при каждом изменении данных в таблице.
3. Создано два пользователя с разными правами доступа и показана работа с ними.
Использование многопользовательской модели в базе данных позволяет разграничить доступ пользователей к таблицам, выделяя каждому пользователю свою версию схемы. Это может быть использовано для повышения безопасности базы данных.
4. Создана **PROCEDURE**, которая по заданному **id** чемпионата показывает все стоимости этого чемпионата в сезон.
5. Создана **FUNCTION**, которая по заданному **id** объединяет ФИ - фамилию и имя, футболиста.
Пользовательские функции и процедуры являются инструментом процедурного программирования в базе данных, они позволяют создавать полезные инструменты, например, для форматирования данных. Однако, сложную логику лучше реализовывать на back-end'e.
6. Транзакционная модель обеспечивает параллельный доступ к данным для нескольких пользователей. Уровень изоляции транзакции обеспечивает ту или иную степень согласованности данных, которую можно оценить, например, по выполнению феноменов параллельного доступа.
В рамках работы был проверен уровень изоляции **READ UNCOMMITTED**, который допускает феномены, связанные с чтением.

Список литературы

- [1] Представления, VIEW // Sql-academy URL: <https://sql-academy.org/ru/guide/view> (дата обращения: 10.11.2023).
- [2] MySQL. Хранимые процедуры. Простые примеры // Digital-flame URL: <http://digital-flame.ru/2016/01/27/mysql-hranimyie-protseduryi-i-funktsii/?ysclid=lpu9l1npit261797014> (дата обращения: 23.11.2023).
- [3] Хранимые процедуры и триггеры // Zoonman URL: http://www.zoonman.ru/library/mysql_sr_and_t.htm (дата обращения: 23.11.2023).
- [4] Транзакции в MySQL // Garb URL: <https://www.garb.ru/blog/transaction.html?ysclid=lpu9flg2ld847890643> (дата обращения: 07.12.2023).
- [5] Введение в транзакции в MySQL // Вебисторий: практический учебник для программистов и системных администраторов URL: <https://webistore.ru/sql/tranzakcii-v-mysql/?ysclid=lpu9crepw5755218785> (дата обращения: 07.12.2023).