МИНОБРНАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский политехнический университет Петра Великого»
Институт компьютерных наук и технологий
Высшая школа искусственного интеллекта

Отчет о лабораторных работах

Дисциплина: «Методы проектирования баз данных»

Сдал: _____ Черников С. Г. группа 3530201/80101
Принял: _____ к.т.н. доц. Попов С.Г.

Санкт-Петербург, 2021

Содержание

Постановка задачи

В рамках комплекса лабораторных работ необходимо, используя разработанную в течение курса «Теоретические основы баз данных» предыдущего семестра базу данных шахматных партий, выполнить следующие задания:

1. Создать два представления (VIEW): первое – для сбора таблицы количества шахматистов для каждого звания, второе – для сбора таблицы количества побед и поражений шахматистов для каждого звания.

2. Создать таблицу для сбора статистики количества завершенных игр в течение дня, месяца и года. Для изменения таблицы статистики создать набор триггеров, которые будут актуализировать информацию о завершенных играх.

3. Создать пользовательскую функцию и процедуру. Продемонстрировать их работу.

4. Исследовать применение и администрирование прав пользователей базы данных. Создать 2 пользователей с разным набором прав, продемонстрировать их возможности.

5. Выбрать уровень изоляции и с помощью пользователей, созданных в задании 4, продемонстрировать выполнение или невыполнение феноменов.

Схема используемой базы данных, разработанной в рамках курса предыдущего семестра данного курса, приведена на рис. 1.

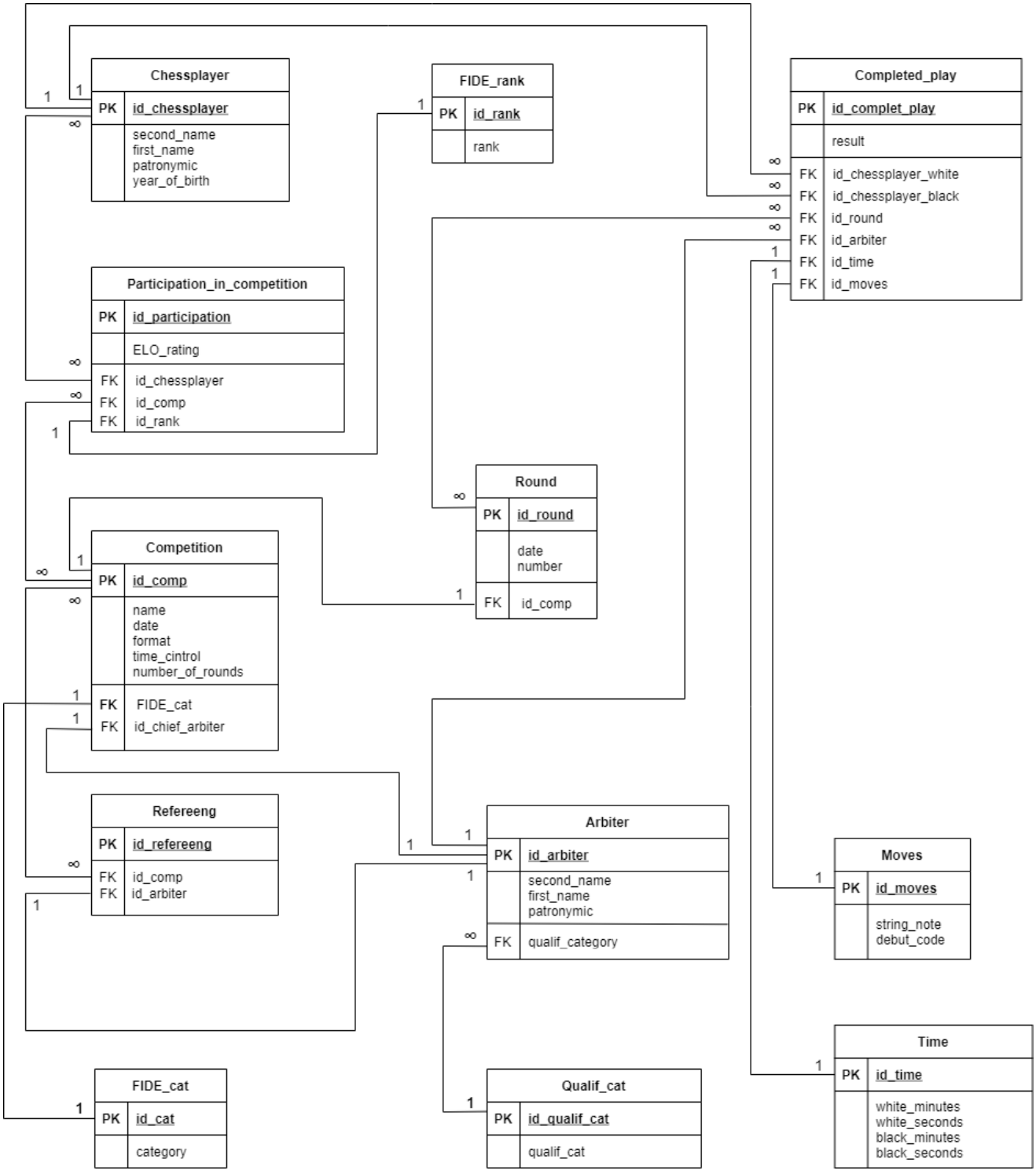

Рис. 1: Схема базы данных

1 Задание 1: view

Необходимо создать представление (view), которые отражают информацию о:
а) количестве шахматистов, имеющих определенное звание, для каждого из званий ФИДЕ;
б) количестве выигранных и проигранных партий шахматистами, имеющими определенного звание, для каждого из званий ФИДЕ.

Ниже приведен код для создания view с функционалом из п. а).

```
CREATE OR REPLACE VIEW players_statistics_simple_view AS
  SELECT _rank, COUNT(*) AS number_of_players
  FROM (SELECT _rank
    FROM Participation_in_competition
      JOIN FIDE_rank
      ON FIDE_rank.id_rank = Participation_in_competition.FIDE_rank
    GROUP BY id_chessplayer, _rank) AS t
  GROUP BY _rank;
```

На рис. 2 приведен результат выполнения запроса, соответствующего представлению из п. а).

_rank	number_of_players
GM	723
IM	713
FM	692
CM	660
WGM	695
WIM	692
WFM	681
	712

Рис. 2: Вывод view для п. а)

код для создания view с функционалом из п. б).

```
CREATE OR REPLACE VIEW players_statistics_view AS
  SELECT _rank, COUNT(*) AS number_of_players, SUM(number_of_wins), SUM
    (number_of_loses)
  FROM
    (SELECT _rank, COUNT(CASE
    WHEN id_chessplayer_white = Participation_in_competition.
   id_chessplayer AND result = '1-0' OR
    id_chessplayer_black = Participation_in_competition.id_chessplayer
   AND result = '0-1' THEN 1
    ELSE NULL
    END) AS number_of_wins,
    COUNT(CASE
    WHEN id_chessplayer_black = Participation_in_competition.
   id_chessplayer AND result = '1-0' OR
```

```
    id_chessplayer_white = Participation_in_competition.id_chessplayer
   AND result = '0-1' THEN 1
    ELSE NULL
    END) AS number_of_loses
    FROM Participation_in_competition
      JOIN FIDE_rank
      ON FIDE_rank.id_rank = Participation_in_competition.FIDE_rank
      JOIN _Round
      ON _Round.id_comp = Participation_in_competition.id_comp
      LEFT JOIN Completed_play
      ON Completed_play.id_round = _Round.id_round AND
        (Completed_play.id_chessplayer_white =
   Participation_in_competition.id_chessplayer OR
   Participation_in_competition.id_chessplayer = id_chessplayer_black)
    GROUP BY Participation_in_competition.id_chessplayer, _rank) as t

  GROUP BY _rank;
```

На рис. 3 приведен результат выполнения запроса, соответствующего представлению из п. б).

_rank	number_of_players	SUM(number_of_wins)	SUM(number_of_loses)
GM	723	2436	2325
IM	713	2353	2360
FM	692	2275	2306
CM	660	2127	2115
WGM	695	2262	2231
WIM	692	2274	2252
WFM	681	2155	2170
	712	2240	2336

Рис. 3: Вывод view для п. б)

Применим представление из п. а) в следующем запросе: необходимо для каждого звания ФИДЕ и соответствующего ему числа шахматистов вывести фамилию одного шахматиста, который последним играл партию и дату игры.

Код запроса приведен ниже:

```
SELECT t._rank, number_of_players, second_name, t._date FROM
players_statistics_simple_view
JOIN
  (
    SELECT second_name, _rank, _Round._date
    FROM FIDE_rank
    JOIN Participation_in_competition ON Participation_in_competition.
   FIDE_rank = FIDE_rank.id_rank
    JOIN Chessplayer ON Chessplayer.id_chessplayer =
   Participation_in_competition.id_chessplayer
    JOIN _Round
```

```
10        ON _Round.id_comp = Participation_in_competition.id_comp
11      JOIN Completed_play ON Completed_play.id_round = _Round.id_round
      AND
12         (Completed_play.id_chessplayer_white =  Chessplayer.
      id_chessplayer OR Chessplayer.id_chessplayer = id_chessplayer_black)
13      GROUP BY _rank, Chessplayer.id_chessplayer, _Round._date
14      ORDER BY _Round._date DESC
15    ) AS t
16 ON players_statistics_simple_view._rank = t._rank
17 GROUP BY t._rank;
```

На рис. 4 приведен вывод для данного запроса.

	_rank	number_of_players	second_name	_date
▶	CM	660	Фомин	2020-11-09
		712	Блинов	2020-11-09
	GM	723	Волков	2020-11-09
	WIM	692	Щукин	2020-11-09
	FM	692	Вист	2020-11-09
	IM	713	Иванов	2020-11-09
	WFM	681	Волобуев	2020-11-09
	WGM	695	Богданов	2020-11-09

Рис. 4: Вывод для запроса, использующего view из п.а)

2 Задание 2: триггеры

Необходимо вести учет числа завершенных игр за последний день, месяц, год. Для этого создать таблицу Num_last_plays, отражающую число завершенных игр за день, месяц, год и триггеры, которые при изменении данных в базе будут обновлять содержимое этой таблицы.

Дата сыгранной партии хранится в таблице _Round (тур), на которую ссылается завершенная игра. Триггеры должны отслеживать: добавление и удаление игр, обновление игры (изменение ссылки на тур), обновление туров (изменение даты), удаление тура (ведущее к удалению всех сыгранных партий). Ниже приведен код разработанных триггеров.

```
drop trigger if exists insert_play_trigger;
# добавление игры
DELIMITER $$
CREATE TRIGGER insert_play_trigger
AFTER INSERT ON Completed_play FOR EACH ROW
BEGIN
  CALL add_for_period(NEW.id_round, 1);
END$$
DELIMITER ;

drop trigger if exists delete_play_trigger;
# удаление игры
DELIMITER $$
CREATE TRIGGER delete_play_trigger
AFTER DELETE ON Completed_play FOR EACH ROW
BEGIN
  CALL sub_for_period(OLD.id_round, 1);
END$$
DELIMITER ;

drop trigger if exists update_play_trigger;
# обновление игры
DELIMITER $$
CREATE TRIGGER update_play_trigger
AFTER UPDATE ON Completed_play FOR EACH ROW
BEGIN
  IF date_of_play(NEW.id_round) != date_of_play(OLD.id_round)
  THEN
    CALL sub_for_period(OLD.id_round, 1);
    CALL add_for_period(NEW.id_round, 1);
  END IF;
END$$
DELIMITER ;

drop trigger if exists update_round_trigger;
# обновление тура
```

```
37 DELIMITER $$
38 CREATE TRIGGER update_round_trigger
39 AFTER UPDATE ON _Round FOR EACH ROW
40 BEGIN
41   IF NEW._date != OLD._date
42   THEN
43     CALL sub_for_period(OLD.id_round, num_of_plays_for_round(OLD.
    id_round));
44     CALL add_for_period(NEW.id_round, num_of_plays_for_round(OLD.
    id_round));
45   END IF;
46 END$$
47 DELIMITER ;
48
49 drop trigger if exists delete_round_trigger;
50 # удаление тура => удаление всех игр в этом туре
51 DELIMITER $$
52 CREATE TRIGGER delete_round_trigger
53 AFTER DELETE ON _Round FOR EACH ROW
54 BEGIN
55   DELETE FROM Completed_play WHERE id_round = OLD.id_round;
56 END$$
57 DELIMITER ;
```

В данных триггерах вызываются вспомогательные процедуры sub_for_period, add_for_period, а также функция date_of_play. Ниже приведено их определение.

```
1  DELIMITER $$
2  CREATE PROCEDURE add_for_period ( round_id INT, delta INT )
3  BEGIN
4    IF DATE_SUB(CURRENT_DATE(), INTERVAL 1 DAY) < date_of_play(round_id)
5    THEN
6      UPDATE Num_last_plays
7      SET for_last_day = for_last_day + delta;
8    END IF;
9    IF DATE_SUB(CURRENT_DATE(), INTERVAL 1 MONTH) < date_of_play(round_id
     )
10   THEN
11     UPDATE Num_last_plays
12     SET for_last_month = for_last_month + delta;
13   END IF;
14   IF DATE_SUB(CURRENT_DATE(), INTERVAL 1 YEAR) < date_of_play(round_id)
15   THEN
16     UPDATE Num_last_plays
17     SET for_last_year = for_last_year + delta;
18   END IF;
19 END; $$
20 DELIMITER ;
21
```

```
22
23 DELIMITER $$
24 CREATE PROCEDURE sub_for_period ( round_id INT, delta INT )
25 BEGIN
26   IF DATE_SUB(CURRENT_DATE(), INTERVAL 1 DAY) < date_of_play(round_id)
27   THEN
28     UPDATE Num_last_plays
29     SET for_last_day = for_last_day - delta;
30   END IF;
31   IF DATE_SUB(CURRENT_DATE(), INTERVAL 1 MONTH) < date_of_play(round_id
     )
32   THEN
33     UPDATE Num_last_plays
34     SET for_last_month = for_last_month - delta;
35   END IF;
36   IF DATE_SUB(CURRENT_DATE(), INTERVAL 1 YEAR) < date_of_play(round_id)
37   THEN
38     UPDATE Num_last_plays
39     SET for_last_year = for_last_year - delta;
40   END IF;
41 END; $$
42 DELIMITER ;
43
44
45 DELIMITER $$
46 # по id тура определяет, когда была сыграна партия, те.. когда тур проводился
47 CREATE FUNCTION date_of_play ( round_id INT )
48 RETURNS DATE DETERMINISTIC
49 BEGIN
50   RETURN (SELECT _date FROM _Round WHERE id_round = round_id);
51 END; $$
52 DELIMITER ;
53
54
55 DELIMITER $$
56 CREATE FUNCTION num_of_plays_for_round ( round_id INT)
57 RETURNS INT DETERMINISTIC
58 BEGIN
59   RETURN (SELECT COUNT(*) FROM Completed_play WHERE id_round = round_id
     );
60 END; $$
61 DELIMITER ;
```

3 Задание 3: функции

Необходимо написать 1 процедуру и 1 функцию. Описание и определение разработанной процедуры приведено в разделе, соответствующем заданию 2.

Была реализована функция **player_wins**:
Вход: id_chessplayer INT – id рассматриваемого игрока,
id_chessplayer_white INT, id_chessplayer_black INT – id игроков, игравших белыми и черными, result CHAR(30) – строковое обозначение результата партии ("1-0"0-1"1/2-1/2")
Выход: BOOL: TRUE, если рассматриваемый игрок с id_chessplayer играл и выиграл в данной партии, иначе FALSE

```
DELIMITER $$
CREATE FUNCTION player_wins ( id_chessplayer INT, id_chessplayer_white
   INT, id_chessplayer_black INT, result CHAR(30) )
RETURNS BOOL DETERMINISTIC

BEGIN
  RETURN id_chessplayer = id_chessplayer_white AND result = "1-0" OR
    id_chessplayer = id_chessplayer_black AND result = "0-1";
END; $$
DELIMITER ;
```

Функция может использоваться в следующем запросе:

```
# 1 б Вывести всех шахматистов, игры которых судил судья A и которые выиграли
   свою игру в м5 type; choose Completed_play.id_arbiter = 1, Arbiter.
   id_arbiter = 1
SELECT Chessplayer.id_chessplayer AS id,
Chessplayer.second_name AS second_name,
Chessplayer.first_name AS first_name,
Chessplayer.pathronymic AS pathronymic,
Arbiter.second_name AS arbiter
FROM Chessplayer
  JOIN (Completed_play
  JOIN _Round ON Completed_play.id_round = _Round.id_round AND _Round.
    _number = 5
  )
  ON player_wins(Chessplayer.id_chessplayer, id_chessplayer_white,
    id_chessplayer_black, result)
  JOIN Arbiter ON Arbiter.id_arbiter = Completed_play.id_arbiter
WHERE Completed_play.id_arbiter = 1;
```

Результат выполнения запроса приведен на рис. 5.

	id	second_name	first_name	pathronymic	arbiter
▶	3185971	Щукин	Петр	Николаевич	Блинов
	6123678	Гусев	Марк	Геннадьевич	Блинов

Рис. 5: Результат выполнения запроса, использующего функцию

Была реализована процедура **add_for_period**, выполняющая изменение таблицы Num_last_plays, учитывающей число сыгранных игр за последний день, месяц, год. Изменения состоят в том, что если дата сыгранной партии (партий) лежит в интервале одних суток, одного месяца или одного года с текущего момента времени, то происходит увеличение счетчика игр в соответствующем поле таблицы на величину delta, являющуюся аргументом процедуры.

Код процедуры приведен ниже.

```
CREATE PROCEDURE add_for_period ( round_id INT, delta INT )
BEGIN
  IF DATE_SUB(CURRENT_DATE(), INTERVAL 1 DAY) < date_of_play(round_id)
  THEN
    UPDATE Num_last_plays
    SET for_last_day = for_last_day + delta;
    END IF;
  IF DATE_SUB(CURRENT_DATE(), INTERVAL 1 MONTH) < date_of_play(round_id
   )
  THEN
    UPDATE Num_last_plays
    SET for_last_month = for_last_month + delta;
  END IF;
  IF DATE_SUB(CURRENT_DATE(), INTERVAL 1 YEAR) < date_of_play(round_id)
  THEN
    UPDATE Num_last_plays
    SET for_last_year = for_last_year + delta;
  END IF;
END; $$
DELIMITER ;
```

Процедура использовалась в триггерах, описанных в разделе 2.

4 Задание 4: права пользователей

Необходимо создать двух пользователей с различными правами:
пользователь oppressor наделен правами SELECT, UPDATE, DELETE на таблицу Chessplayer;
пользователь discriminated имеет лишь права на SELECT на view из задания 1.

Код для создания пользователей и наделения их правами:

```
1 CREATE USER oppressor@localhost IDENTIFIED BY 'oppressor';
2 CREATE USER discriminated@localhost IDENTIFIED BY 'discriminated';
3
4 GRANT SELECT, UPDATE, DELETE ON Chessbase.Chessplayer TO
    oppressor@localhost;
5 GRANT SELECT ON Chessbase.players_statistics_view TO
    discriminated@localhost;
6 GRANT SELECT ON Chessbase.players_statistics_simple_view TO
    discriminated@localhost;
```

Для пользователя oppressor запрос `SHOW TABLES` выводит следующую таблицу:

```
+---------------------+
| Tables_in_chessbase |
+---------------------+
| chessplayer         |
+---------------------+
```

Этот же запрос для пользователя discriminated выводит:

```
+-------------------------------+
| Tables_in_chessbase           |
+-------------------------------+
| players_statistics_simple_view |
| players_statistics_view       |
+-------------------------------+
```

Можно видеть, как для discriminated разрешено обращение ко view. Например:

```
mysql> SELECT * FROM players_statistics_simple_view;
+-------+------------------+
| _rank | number_of_players |
+-------+------------------+
| GM    |              723 |
| IM    |              713 |
| FM    |              692 |
| CM    |              660 |
| WGM   |              695 |
| WIM   |              692 |
| WFM   |              681 |
|       |              712 |
+-------+------------------+
8 rows in set (0.02 sec)
```

В то же время, обращение к Chessplayer недоступно:

```
mysql> SELECT * FROM Chessplayer;
ERROR 1142 (42000): SELECT command denied to user 'discriminated'@
'localhost' for table 'chessplayer'
```

Для oppressor, наоборот, есть доступ к таблице Chessplayer, но нет доступа к view:

```
mysql> SELECT * FROM players_statistics_simple_view;
ERROR 1142 (42000): SELECT command denied to user 'oppressor'@
'localhost' for table 'players_statistics_simple_view'
mysql> SELECT id_chessplayer FROM Chessplayer
WHERE year_of_birth = 1940 AND second_name = 'Вист';
+----------------+
| id_chessplayer |
+----------------+
|        1386455 |
|        2471211 |
|        4304849 |
|        5006282 |
|        7300790 |
+----------------+
```

5 Задание 5: транзакции

Необходимо для уровня изоляции транзакций 2 - Repeatable Read, проверить при их выполнении наличие следующих феноменов:

1. Dirty read — артефакт чтения данных, внесённых неподтверждённой транзакцией. Условие возникновения: в рамках одновременных транзакций происходит обращение к одним и тем же строкам таблицы, одна транзакция изменяет данные в таблице, вторая в это время читает эти данные, первая транзакция откатывает изменения. Артефакт проявляется, если в результате чтения были получены не зафиксированные коммитом данные.

2. Lost update — артефакт, возникающий при последовательной записи данных в одни и те же поля из разных одновременных транзакций. Проявляется как утеря всех изменений, кроме последней.

3. Non-repeatable read — артефакт, проявляющийся при чтении из одной транзакции данных, которые в это время изменяются в другой транзакции. При наличии такого артефакта, результат чтения одних и тех же данных может отличаться.

4. Phantom read — артефакт, который возникает, когда в одной транзакции идёт выборка из множества полей таблицы с некоторой агрегирующей функцией и, при этом, другая транзакция изменяет данную таблицу. Артефакт проявляется в том, что результат выборки читающей транзакции изменяется без видимых причин (с точки зрения этой транзакции).

5.1 Проверка Dirty read

Пусть пользователь root будет совершать транзакцию, в то время как test будет периодически обращаться к изменяемым в рамках транзакции строкам таблицы.
root:

```
mysql> SELECT * from chessplayer WHERE id_chessplayer = 1026407;
+----------------+-------------+------------+-------------+---------------+
| id_chessplayer | second_name | first_name | pathronymic | year_of_birth |
+----------------+-------------+------------+-------------+---------------+
|        1026407 | Волобуев    | Леонид     | Николаевич  |          1982 |
+----------------+-------------+------------+-------------+---------------+
1 row in set (0.00 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE Chessplayer
-> SET year_of_birth = 2010
-> WHERE id_chessplayer = 1026407;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

test:

```
mysql> SELECT * from chessplayer WHERE id_chessplayer = 1026407;
+----------------+-------------+------------+-------------+---------------+
| id_chessplayer | second_name | first_name | pathronymic | year_of_birth |
+----------------+-------------+------------+-------------+---------------+
|        1026407 | Волобуев    | Леонид     | Николаевич  |          1982 |
+----------------+-------------+------------+-------------+---------------+
1 row in set (0.00 sec)
```

root:

```
mysql> ROLLBACK;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * from chessplayer WHERE id_chessplayer = 1026407;
+----------------+-------------+------------+-------------+---------------+
| id_chessplayer | second_name | first_name | pathronymic | year_of_birth |
+----------------+-------------+------------+-------------+---------------+
|        1026407 | Волобуев    | Леонид     | Николаевич  |          1982 |
+----------------+-------------+------------+-------------+---------------+
1 row in set (0.00 sec)
```

Для выбранной степени изоляции артефакт Dirty read не обнаружен — незафиксированные данные не видны вне транзакции.

5.2 Проверка Lost update

Выполним одновременное изменение одной и той же записи в рамках двух транзакций. **root**:

```
mysql> SELECT * from chessplayer WHERE id_chessplayer = 1026407;
+----------------+-------------+------------+-------------+---------------+
| id_chessplayer | second_name | first_name | pathronymic | year_of_birth |
+----------------+-------------+------------+-------------+---------------+
|        1026407 | Волобуев    | Леонид     | Николаевич  |          1982 |
+----------------+-------------+------------+-------------+---------------+
1 row in set (0.00 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
```

test:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
```

root

```
mysql> UPDATE Chessplayer
-> SET year_of_birth = year_of_birth + 5
-> WHERE id_chessplayer = 1026407;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

test

```
mysql> UPDATE Chessplayer
-> SET year_of_birth = year_of_birth + 7
-> WHERE id_chessplayer = 1026407;
```

<ожидание подтверждения транзакции в root>

root

```
mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)
```

test

```
Query OK, 1 row affected (24.48 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)
```

root

```
mysql> SELECT * from chessplayer WHERE id_chessplayer = 1026407;
+----------------+-------------+------------+-------------+---------------+
| id_chessplayer | second_name | first_name | pathronymic | year_of_birth |
+----------------+-------------+------------+-------------+---------------+
|        1026407 | Волобуев    | Леонид     | Николаевич  |          1994 |
+----------------+-------------+------------+-------------+---------------+
1 row in set (0.00 sec)
```

Можно видеть, что были применены изменения, предусматриваемые обеими транзакциями – увеличение года рождения и на 5, и на 7. Феномена Lost update не обнаружено.

5.3 Проверка Non-repeatable read

Будем выполнять две транзакции: в рамках одной будем производить изменение данных, а в рамках другой – читать изменяемые в первой данные.

root

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
```

test

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * from chessplayer WHERE id_chessplayer = 1026407;
+----------------+-------------+------------+-------------+---------------+
| id_chessplayer | second_name | first_name | pathronymic | year_of_birth |
+----------------+-------------+------------+-------------+---------------+
|        1026407 | Волобуев    | Леонид     | Николаевич  |          1994 |
+----------------+-------------+------------+-------------+---------------+
1 row in set (0.00 sec)
```

root

```
mysql> UPDATE Chessplayer
-> SET year_of_birth = year_of_birth + 5
-> WHERE id_chessplayer = 1026407;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)
```

test

```
mysql> SELECT * from chessplayer WHERE id_chessplayer = 1026407;
+----------------+-------------+------------+-------------+---------------+
| id_chessplayer | second_name | first_name | pathronymic | year_of_birth |
+----------------+-------------+------------+-------------+---------------+
|        1026407 | Волобуев    | Леонид     | Николаевич  |          1994 |
+----------------+-------------+------------+-------------+---------------+
1 row in set (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * from chessplayer WHERE id_chessplayer = 1026407;
+----------------+-------------+------------+-------------+---------------+
| id_chessplayer | second_name | first_name | pathronymic | year_of_birth |
+----------------+-------------+------------+-------------+---------------+
|        1026407 | Волобуев    | Леонид     | Николаевич  |          1999 |
+----------------+-------------+------------+-------------+---------------+
1 row in set (0.00 sec)
```

Можно видеть, что изменения, выполненные в рамках транзакции root не повлияли на чтение, производимое в рамках транзакции test, при этом после завершения этой транзакции изменения данных заметно при чтении.

5.4 Проверка Phantom read

Смоделируем ситуацию, похожую на предыдущий пункт, но при чтении будем вызывать агрегирующую функцию для всей таблицы.

root

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
```

test

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT COUNT(*) FROM Chessplayer;
+----------+
| COUNT(*) |
+----------+
|     5000 |
+----------+
1 row in set (0.01 sec)
```

root

```
mysql> INSERT INTO Chessplayer
-> VALUES (99999999, 'Заборовский', 'Владимир', 'Сергеевич', 2021);
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)
```

test

```
mysql> SELECT COUNT(*) FROM Chessplayer;
+----------+
| COUNT(*) |
+----------+
|     5000 |
+----------+
1 row in set (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT COUNT(*) FROM Chessplayer;
+----------+
| COUNT(*) |
+----------+
|     5001 |
+----------+
1 row in set (0.01 sec)
```

Для выбранной степени изоляции артефакт Phantom read не обнаружен — в рамках транзакции агрегирующая функция на выборке работает предсказуемо и одинаково, в то время, как эта же выборка изменяется в другой транзакции. После фиксации обеих транзакций изменения публикуются и становятся доступны для всех пользователей базы данных.

Заключение

В результате выполнения комплекса лабораторных работ были созданы view в базе данных, были написаны триггеры для ведения статистики завершенных игр за последний день, месяц, год, была реализована пользовательская функция и процедура, были созданы пользователи, и им были назначены права, а также для уровня изоляции транзакций 2 - Repeatable Read были проверены феномены параллельного доступа. Для выполнения комплекса работ использовавалась база данных шахматных партий. Использоваласьреализация базы данных MySQL 8, для подключения использовалось средство MySQL Workbench CE.

View позволяет создавать представления, предназначенные, например, для ограничения доступа к базе данных отдельных пользователей – для этого достаточно конкретному пользователю предоставить права на view. Такой пользователь получит право выполнять конкретный запрос (без возможности выполнять прочие запросы, в т.ч. к тем же таблицам). Следует отметить, что обращение к view – отдельный запрос к базе данных, а хранящиеся метаданные – лишь сам код запроса. Materialized view в MySQL отсутствует, и может быть реализован лишь с помощью дополнительных средств, таких как триггеры.

Триггер в базе данных – инструмент событийного программирования, он позволяет предусмотреть определенную логику, соответствующую какой-либо модификации данных в базе. В выполненном задании триггеры использовались для сбора статистики, учитывающей количество сыгранных партий за последний день, месяц, год. Триггер при каждом изменении значимых полей таблиц изменял счетчик сыгранных партий на 1, что обеспечивает высокое быстродействие по сравнению с реализацией, при которой при каждом изменении целиком выполнялся бы запрос с подсчетом сыгранных партий.

Пользовательские функции и процедуры, являясь инструментом процедурного программирования в базе данных, позволили создать эффективные инструменты, сокращающие дублирование кода.

Многопользовательские модели позволили разграничить доступ пользователей к таблицам, преддоставляя каждому пользователю его отдельную версию схемы, что упрощает реализацию требуемой политики безопасности.

Транзакционная модель обеспечивает параллельный доступ к данным для нескольких пользователей. Степень согласованности данных в разных транзакциях определяется уровнем изоляции транзакции. В работе был исследован уровень изоляции 2 - Repeatable Read.