

ГУАП

КАФЕДРА № 42

ОТЧЕТ  
ЗАЩИЩЕН С ОЦЕНКОЙ \_\_\_\_\_

ПРЕПОДАВАТЕЛЬ

доцент, кандидат тех. наук				Бржезовский А. В.
должность, уч. степень, звание		подпись, дата		инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2

**ТРАНЗАКЦИИ И БЛОКИРОВКИ**

Вариант 5

по курсу: МЕТОДЫ И СРЕДСТВА ПРОЕКТИРОВАНИЯ  
ИНФОРМАЦИОННЫХ СИСТЕМ И ТЕХНОЛОГИЙ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	4128			Воробьев В.А.
			подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

## СОДЕРЖАНИЕ

<b>1</b>	<b>Постановка задачи</b>	<b>3</b>
1.1	Задание	3
<b>2</b>	<b>Выполнение работы</b>	<b>4</b>
2.1	Тупик	4
2.2	Грязное чтение	6
2.3	Неповторяющегося чтение	8
2.4	Фантомное чтение	10
<b>3</b>	<b>Вывод</b>	<b>13</b>
	<b>ПРИЛОЖЕНИЕ</b>	<b>14</b>

## **1 Постановка задачи**

**Цель работы:** Изучить понятия транзакции и блокировки и их назначение. Реализовать транзакции и блокировки при выполнении запросов в собственной базе данных и проанализировать полученные результаты.

### **1.1 Задание**

- Смоделировать в БД грязное чтение, неповторяемое чтение, фантомы, изменяя уровень изоляции транзакций продемонстрировать их исключение, сформировать отчеты о блокировках, пояснить их содержание.
- Смоделировать в БД тупик (взаимную блокировку), получить с помощью приложения SQL Server Profiler отчет о тупике, пояснить его содержание.

## 2 Выполнение работы

Текст запросов представлен в Приложении, а также в репозитории GitHub (URI - [https://github.com/vladcto/suai-labs/tree/a86b410d6c9e3836fd995d29d47279f17f225253/6\\_semester/%D0%9C%D0%A1%D0%9F%D0%98%D0%A1%D0%A2/10](https://github.com/vladcto/suai-labs/tree/a86b410d6c9e3836fd995d29d47279f17f225253/6_semester/%D0%9C%D0%A1%D0%9F%D0%98%D0%A1%D0%A2/10)).

### 2.1 Тупик

Листинг демонстрирует возникновение тупиковой ситуации при использовании двух подключений к базе данных, одновременно пытающихся обновлять записи в таблице student. Создаются два соединения, connection1 и connection2, каждый из которых начинает транзакцию. cursor1 сначала изменяет имя студента с id=1 на “Alice”, в то время как cursor2 меняет имя студента с id=2 на “Bob”. Затем, в попытке создать тупик, cursor1 пытается обновить id=2, в то время как cursor2 пытается обновить id=1. Эти перекрестные обновления приводят к тупику, поскольку каждое соединение блокирует ресурс, необходимый другому. Обнаруженная ошибка вынуждает вернуть предыдущие состояния через rollback(). В конце, чтобы получить подробную информацию о тупике, выполняется запрос SHOW ENGINE INNODB STATUS, и результаты выводятся на экран.

```
1 import mysql.connector
2
3 def deadlock_example():
4     connection1 = mysql.connector.connect(
5         host='localhost',
6         user='root',
7         password='',
8         database='conference_db_lab1'
9     )
10
11     connection2 = mysql.connector.connect(
12         host='localhost',
13         user='root',
14         password='',
15         database='conference_db_lab1'
16     )
17
18     try:
```

```

19         cursor1 = connection1.cursor()
20         cursor2 = connection2.cursor()
21
22         connection1.start_transaction()
23         cursor1.execute("UPDATE student SET name='Alice'
24                           WHERE id=1")
25
26         connection2.start_transaction()
27         cursor2.execute("UPDATE student SET name='Bob' WHERE
28                           id=2")
29
30         try:
31             cursor1.execute("UPDATE student SET name='Alice'
32                               WHERE id=2")
33             connection1.commit()
34         except mysql.connector.Error:
35             connection1.rollback()
36
37         try:
38             cursor2.execute("UPDATE student SET name='Bob'
39                               WHERE id=1")
40             connection2.commit()
41         except mysql.connector.Error:
42             connection2.rollback()
43
44         report_cursor = connection1.cursor()
45         report_cursor.execute("SHOW ENGINE INNODB STATUS")
46         status = report_cursor.fetchone()[2]
47
48         print("\nОтчет о последнем тупике:\n")
49         print(status)
50
51     except mysql.connector.Error as err:
52         print(f"Общая ошибка: {err}")
53
54     finally:
55         connection1.close()
56         connection2.close()
57
58     deadlock_example()

```

Отчет о последнем тупике:

```
=====
2024-12-27 05:10:37 0x1710bb000 INNODB MONITOR OUTPUT
=====
```

Per second averages calculated from the last 26 seconds

```
=====
BACKGROUND THREAD
```

```
srv_master_thread loops: 16 srv_active, 0 srv_shutdown, 4044 srv_idle
srv_master_thread log flush and writes: 0
```

```
=====
SEMAPHORES
```

```
OS WAIT ARRAY INFO: reservation count 156
OS WAIT ARRAY INFO: signal count 162
RW-shared spins 0, rounds 0, OS waits 0
RW-excl spins 0, rounds 0, OS waits 0
RW-sx spins 0, rounds 0, OS waits 0
Spin rounds per wait: 0.00 RW-shared, 0.00 RW-excl, 0.00 RW-sx
```

```
=====
TRANSACTIONS
```

```
Trx id counter 21015
Purge done for trx's n:o < 21011 undo n:o < 0 state: running but idle
History list length 2
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 281479978423752, not started
0 lock struct(s), heap size 1128, 0 row lock(s)
---TRANSACTION 281479978422960, not started
0 lock struct(s), heap size 1128, 0 row lock(s)
---TRANSACTION 281479978422168, not started
0 lock struct(s), heap size 1128, 0 row lock(s)
---TRANSACTION 281479978421376, not started
0 lock struct(s), heap size 1128, 0 row lock(s)
---TRANSACTION 281479978420584, not started
0 lock struct(s), heap size 1128, 0 row lock(s)
---TRANSACTION 281479978419792, not started
0 lock struct(s), heap size 1128, 0 row lock(s)
```

Рисунок 2.1 - Результат выполнения

## 2.2 Грязное чтение

Листинг демонстрирует пример грязного чтения, которое может произойти в базе данных MySQL при низком уровне изоляции транзакций. Создаются два соединения с базой данных `conference_db_lab1`. Каждое из них устанавливает уровень изоляции транзакции `READ UNCOMMITTED`, при котором транзакции могут читать неподтвержденные изменения других транзакций. `connection1` начинает транзакцию и обновляет имя студента с `id=1` на “John Doe”. До того, как эта транзакция подтверждена или отмене-

на, `connection2` выполняет чтение имени того же студента, тем самым демонстрируя грязное чтение — `cursor2` считывает значение “John Doe”, которое может быть впоследствии откатано назад. Результат запроса выводится на экран как “Грязное чтение”. После этого транзакция в `connection1` откатывается, возвратив прежние значения данных. В блоке `finally` обеспечивается закрытие обоих соединений, чтобы гарантировать освобождение ресурсов.

```
1  import mysql.connector
2
3  def dirty_read_example():
4      connection1 = mysql.connector.connect(
5          host='localhost',
6          user='root',
7          password='',
8          database='conference_db_lab1'
9      )
10
11     connection2 = mysql.connector.connect(
12         host='localhost',
13         user='root',
14         password='',
15         database='conference_db_lab1'
16     )
17
18     try:
19         cursor1 = connection1.cursor()
20         cursor2 = connection2.cursor()
21
22         cursor1.execute("SET SESSION TRANSACTION ISOLATION
23             LEVEL READ UNCOMMITTED")
24         cursor2.execute("SET SESSION TRANSACTION ISOLATION
25             LEVEL READ UNCOMMITTED")
26
27         connection1.start_transaction()
28         cursor1.execute("UPDATE student SET name='John Doe'
29             WHERE id=1")
30
31         cursor2.execute("SELECT name FROM student WHERE id
32             =1")
33         result = cursor2.fetchone()
```

```

30         print(f"Грязное чтение: {result[0]}")
31
32         connection1.rollback()
33
34     finally:
35         connection1.close()
36         connection2.close()
37
38 if __name__ == "__main__":
39     dirty_read_example()

```

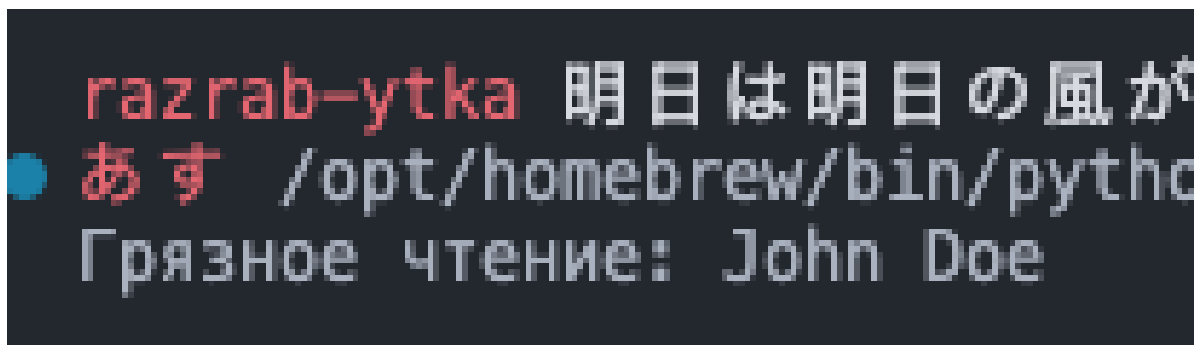


Рисунок 2.2 - Результат выполнения

### 2.3 Неповторяющегося чтение

Листинг иллюстрирует пример неповторяющегося чтения в базе данных MySQL, который может возникнуть при использовании уровня изоляции транзакций `READ COMMITTED`. Создаются два соединения с базой данных `conference_db_lab1`. Первое соединение `connection1` устанавливает уровень изоляции `READ COMMITTED` и начинает транзакцию, в которой считывает имя студента с `id=1`. Результат первого чтения выводится на экран. Затем второе соединение `connection2` начинает транзакцию, обновляет имя того же студента на “Jane Doe” и фиксирует изменения, что делает их видимыми для других транзакций. Возвращаясь к первому соединению, выполняется повторное чтение имени студента, что демонстрирует неповторяющееся чтение — текущие результаты запроса отличается от первоначального, поскольку вторая транзакция изменила данные. Это изменение выводится как “Измененное чтение”. Наконец, транзакция в `connection1` фиксируется, и оба соединения закрываются в блоке `finally`, чтобы гарантировать корректное освобождение ресурсов.



```

1  import mysql.connector
2
3  def non_repeatable_read_example():
4      connection1 = mysql.connector.connect(
5          host='localhost',
6          user='root',
7          password='',
8          database='conference_db_lab1'
9      )
10
11     connection2 = mysql.connector.connect(
12         host='localhost',
13         user='root',
14         password='',
15         database='conference_db_lab1'
16     )
17
18     try:
19         cursor1 = connection1.cursor()
20         cursor2 = connection2.cursor()
21
22         cursor1.execute("SET SESSION TRANSACTION ISOLATION
23                             LEVEL READ COMMITTED")
24
25         connection1.start_transaction()
26         cursor1.execute("SELECT name FROM student WHERE id
27                             =1")
28         result1 = cursor1.fetchone()
29         print(f"Первое чтение: {result1[0]}")
30
31         connection2.start_transaction()
32         cursor2.execute("UPDATE student SET name='Jane Doe'
33                             WHERE id=1")
34         connection2.commit()
35
36         cursor1.execute("SELECT name FROM student WHERE id
37                             =1")
38         result2 = cursor1.fetchone()
39         print(f"Измененное чтение: {result2[0]}")
40
41         connection1.commit()

```

```

38
39     finally :
40         connection1.close()
41         connection2.close()
42
43 if __name__ == "__main__":
44     non_repeatable_read_example()

```

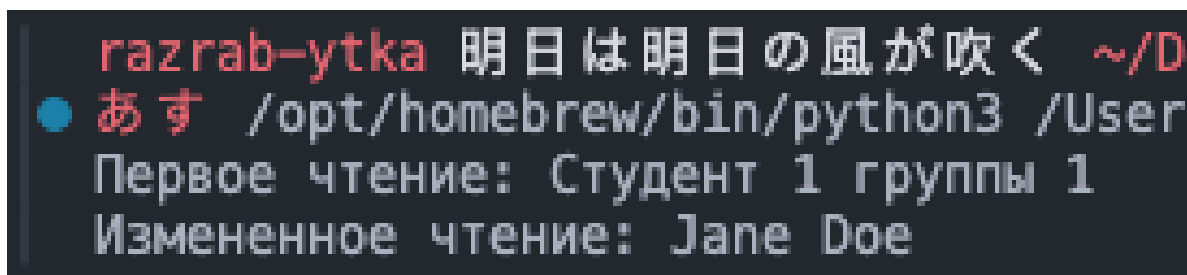


Рисунок 2.3 - Результат выполнения

## 2.4 Фантомное чтение

Листинг демонстрирует возникновение фантомных чтений в базе данных MySQL, которые могут появляться даже при уровне изоляции транзакций REPEATABLE READ. Создаются два подключения к базе данных `conference_db_lab1`. Оба соединения устанавливают уровень изоляции REPEATABLE READ, что гарантирует стабильность выборок, но не защищает полностью от фантомных чтений. Первое соединение `connection1` начинает транзакцию и выполняет подсчет записей в таблице `student` для группы с `group_id=1`. Результат этого подсчета выводится на экран как “Количество перед вставкой”. Затем второе соединение `connection2` начинает свою транзакцию и вставляет новую запись в ту же группу, после чего подтверждает изменения, делая новую запись немедленно видимой для других транзакций. `connection1` снова выполняет подсчет записей в той же группе, и результат показывает увеличение количества, отображая “Количество после вставки”. Это увеличение количества записей, несмотря на установку уровня изоляции REPEATABLE READ, иллюстрирует фантомное чтение. В итоге, чтобы завершить операции корректно, обе транзакции фиксируются, их соединения закрываются в блоке `finally`, обеспечивая освобождение ресурсов системы.

```

1 import mysql.connector

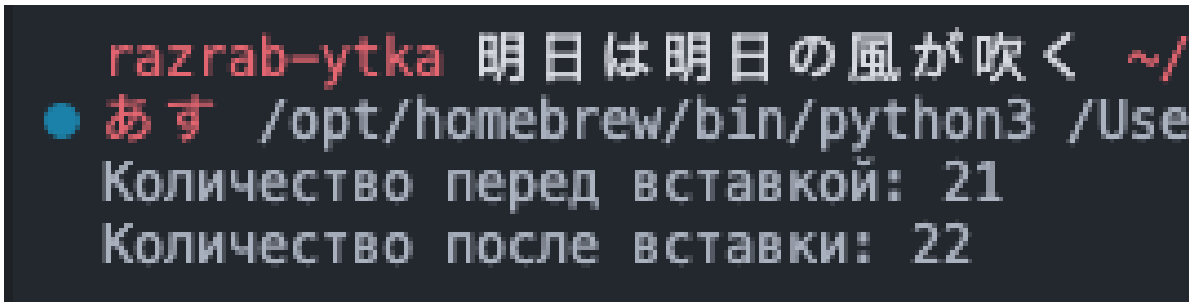
```

```

2
3 def phantom_read_example():
4     connection1 = mysql.connector.connect(
5         host='localhost',
6         user='root',
7         password='',
8         database='conference_db_lab1'
9     )
10
11     connection2 = mysql.connector.connect(
12         host='localhost',
13         user='root',
14         password='',
15         database='conference_db_lab1'
16     )
17
18     try:
19         cursor1 = connection1.cursor()
20         cursor2 = connection2.cursor()
21
22         cursor1.execute("SET SESSION TRANSACTION ISOLATION
23             LEVEL REPEATABLE READ")
24         cursor2.execute("SET SESSION TRANSACTION ISOLATION
25             LEVEL REPEATABLE READ")
26
27         connection1.start_transaction()
28         cursor1.execute("SELECT COUNT(*) FROM student WHERE
29             group_id=1")
30         result1 = cursor1.fetchone()
31         print(f"Количество перед вставкой: {result1[0]}")
32
33         connection2.start_transaction()
34         cursor2.execute("INSERT INTO student (group_id, name)
35             VALUES (1, 'New Phantom Student')")
36         connection2.commit()
37
38         cursor1.execute("SELECT COUNT(*) FROM student WHERE
39             group_id=1")
40         result2 = cursor1.fetchone()
41         print(f"Количество после вставки: {result2[0]}")

```

```
38         connection1.commit()
39
40     finally:
41         connection1.close()
42         connection2.close()
43
44 if __name__ == "__main__":
45     phantom_read_example()
```



A terminal window with a dark background. The prompt is 'razrab-ytka' in red. The command '明日は明日の風が吹く ~/● あす /opt/homebrew/bin/python3 /Use' is entered. The output shows 'Количество перед вставкой: 21' and 'Количество после вставки: 22'.

```
razrab-ytka 明日は明日の風が吹く ~/
● あす /opt/homebrew/bin/python3 /Use
Количество перед вставкой: 21
Количество после вставки: 22
```

Рисунок 2.4 - Результат выполнения

### **3 Вывод**

В результате выполнения лабораторной работы были изучены операции с транзакциями, уровни изоляций этих транзакций для разрешения конфликтов блокировок и виды самих блокировок. Смоделированы различные ситуации (грязное чтение, неповторяемое чтение, фантомы, тупики) в различных условиях.

## ПРИЛОЖЕНИЕ

```
1 import mysql.connector
2
3 def deadlock_example():
4     connection1 = mysql.connector.connect(
5         host='localhost',
6         user='root',
7         password='',
8         database='conference_db_lab1'
9     )
10
11     connection2 = mysql.connector.connect(
12         host='localhost',
13         user='root',
14         password='',
15         database='conference_db_lab1'
16     )
17
18     try:
19         cursor1 = connection1.cursor()
20         cursor2 = connection2.cursor()
21
22         connection1.start_transaction()
23         cursor1.execute("UPDATE student SET name='Alice'
24             WHERE id=1")
25
26         connection2.start_transaction()
27         cursor2.execute("UPDATE student SET name='Bob' WHERE
28             id=2")
29
30         try:
31             cursor1.execute("UPDATE student SET name='Alice'
32                 WHERE id=2")
33             connection1.commit()
34         except mysql.connector.Error:
35             connection1.rollback()
36
37         try:
38             cursor2.execute("UPDATE student SET name='Bob'
39                 WHERE id=1")
40             connection2.commit()
```

```

37         except mysql.connector.Error:
38             connection2.rollback()
39
40         report_cursor = connection1.cursor()
41         report_cursor.execute("SHOW ENGINE INNODB STATUS")
42         status = report_cursor.fetchone()[2]
43
44         print("\nОтчет о последнем тупике:\n")
45         print(status)
46
47     except mysql.connector.Error as err:
48         print(f"Общая ошибка: {err}")
49
50     finally:
51         connection1.close()
52         connection2.close()
53
54 deadlock_example()

```

```

1  import mysql.connector
2
3  def dirty_read_example():
4      connection1 = mysql.connector.connect(
5          host='localhost',
6          user='root',
7          password='',
8          database='conference_db_lab1'
9      )
10
11     connection2 = mysql.connector.connect(
12         host='localhost',
13         user='root',
14         password='',
15         database='conference_db_lab1'
16     )
17
18     try:
19         cursor1 = connection1.cursor()
20         cursor2 = connection2.cursor()
21
22         cursor1.execute("SET SESSION TRANSACTION ISOLATION

```

```

23         cursor2.execute("SET SESSION TRANSACTION ISOLATION
                           LEVEL READ UNCOMMITTED")
24
25         connection1.start_transaction()
26         cursor1.execute("UPDATE student SET name='John Doe'
                           WHERE id=1")
27
28         cursor2.execute("SELECT name FROM student WHERE id
                           =1")
29         result = cursor2.fetchone()
30         print(f"Грязное чтение: {result[0]}")
31
32         connection1.rollback()
33
34     finally:
35         connection1.close()
36         connection2.close()
37
38 if __name__ == "__main__":
39     dirty_read_example()

```

```

1 import mysql.connector
2
3 def non_repeatable_read_example():
4     connection1 = mysql.connector.connect(
5         host='localhost',
6         user='root',
7         password='',
8         database='conference_db_lab1'
9     )
10
11     connection2 = mysql.connector.connect(
12         host='localhost',
13         user='root',
14         password='',
15         database='conference_db_lab1'
16     )
17
18     try:
19         cursor1 = connection1.cursor()
20         cursor2 = connection2.cursor()
21

```



```

22         cursor1.execute("SET SESSION TRANSACTION ISOLATION
           LEVEL READ COMMITTED")
23
24         connection1.start_transaction()
25         cursor1.execute("SELECT name FROM student WHERE id
           =1")
26         result1 = cursor1.fetchone()
27         print(f"Первое чтение: {result1[0]}")
28
29         connection2.start_transaction()
30         cursor2.execute("UPDATE student SET name='Jane Doe'
           WHERE id=1")
31         connection2.commit()
32
33         cursor1.execute("SELECT name FROM student WHERE id
           =1")
34         result2 = cursor1.fetchone()
35         print(f"Измененное чтение: {result2[0]}")
36
37         connection1.commit()
38
39     finally:
40         connection1.close()
41         connection2.close()
42
43 if __name__ == "__main__":
44     non_repeatable_read_example()

```

```

1 import mysql.connector
2
3 def phantom_read_example():
4     connection1 = mysql.connector.connect(
5         host='localhost',
6         user='root',
7         password='',
8         database='conference_db_lab1'
9     )
10
11     connection2 = mysql.connector.connect(
12         host='localhost',
13         user='root',
14         password='',

```

```

15         database='conference_db_lab1 '
16     )
17
18     try:
19         cursor1 = connection1.cursor()
20         cursor2 = connection2.cursor()
21
22         cursor1.execute("SET SESSION TRANSACTION ISOLATION
23             LEVEL REPEATABLE READ")
24         cursor2.execute("SET SESSION TRANSACTION ISOLATION
25             LEVEL REPEATABLE READ")
26
27         connection1.start_transaction()
28         cursor1.execute("SELECT COUNT(*) FROM student WHERE
29             group_id=1")
30         result1 = cursor1.fetchone()
31         print(f"Количество перед вставкой: {result1[0]}")
32
33         connection2.start_transaction()
34         cursor2.execute("INSERT INTO student (group_id, name)
35             VALUES (1, 'New Phantom Student')")
36         connection2.commit()
37
38         cursor1.execute("SELECT COUNT(*) FROM student WHERE
39             group_id=1")
40         result2 = cursor1.fetchone()
41         print(f"Количество после вставки: {result2[0]}")
42
43         connection1.commit()
44
45     finally:
46         connection1.close()
47         connection2.close()
48
49 if __name__ == "__main__":
50     phantom_read_example()

```