ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ

Кафедра
«Криптология и кибербезопасность»

Лабораторная работа №2

по предмету «Безопасность систем баз данных»

Выполнил студент группы Б20-505

Сорочан Илья

Москва – 2024

Содержание

1. Read commited

Для начала выберем две роли, подходящие для выполнения данной операции. Пусть это будет sales_role для совершения операции (INSERT в sales.sales) и staff_role для наблюдения за операцией.

```
mydb=> BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;
BEGIN
mydb=*> INSERT INTO sales.sale (id, date, employee, customer, bonus, bonus_used) VALUES (4, '2023-06-23 12:00:00', 3,
 3, 0, 0);
INSERT 0 1
mydb=*> SELECT * FROM sales.sale;
 id |        date         | employee | customer | bonus | bonus_used
----+---------------------+----------+----------+-------+------------
  1 | 2023-06-20 10:30:00 |        1 |        1 |    50 |          0
  2 | 2023-06-22 14:15:00 |        2 |        2 |    25 |          0
  3 | 2023-06-23 11:00:00 |        3 |        3 |    75 |         75
  4 | 2023-06-23 12:00:00 |        3 |        3 |     0 |          0
(4 rows)

mydb=*> ROLLBACK
```

Рис. 1. READ COMMITED транзакция

При этом при попытке чтения из-под staff_role:

```
mydb=> SELECT * FROM sales.sale;
 id |        date         | employee | customer | bonus | bonus_used
----+---------------------+----------+----------+-------+------------
  1 | 2023-06-20 10:30:00 |        1 |        1 |    50 |          0
  2 | 2023-06-22 14:15:00 |        2 |        2 |    25 |          0
  3 | 2023-06-23 11:00:00 |        3 |        3 |    75 |         75
(3 rows)
```

Рис. 2. Чтение из роли staff_role

Как и должно быть, изменения, внесенные транзакцией READ COMMITED не должны быть видимы другим пользоватям до того как они будут совершены. Теперь попробуем COMMIT:

```
mydb=> BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;
BEGIN
mydb=*> INSERT INTO sales.sale (id, date, employee, customer, bonus, bonus_used) VALUES (4, '2023-06-23 12:00:00', 3,
 3, 0, 0);
INSERT 0 1
mydb=*> COMMIT
mydb-*> ;
COMMIT
mydb=> SELECT * FROM sales.sale;
 id |        date         | employee | customer | bonus | bonus_used
----+---------------------+----------+----------+-------+------------
  1 | 2023-06-20 10:30:00 |        1 |        1 |    50 |          0
  2 | 2023-06-22 14:15:00 |        2 |        2 |    25 |          0
  3 | 2023-06-23 11:00:00 |        3 |        3 |    75 |         75
  4 | 2023-06-23 12:00:00 |        3 |        3 |     0 |          0
(4 rows)
```

Рис. 3. Запись из sales_role

```
mydb=> SELECT * FROM sales.sale;
 id |        date         | employee | customer | bonus | bonus_used
----+---------------------+----------+----------+-------+------------
  1 | 2023-06-20 10:30:00 |        1 |        1 |    50 |          0
  2 | 2023-06-22 14:15:00 |        2 |        2 |    25 |          0
  3 | 2023-06-23 11:00:00 |        3 |        3 |    75 |         75
  4 | 2023-06-23 12:00:00 |        3 |        3 |     0 |          0
(4 rows)
```

Рис. 4. Чтение staff_role после завершения транзакции

2. Repeatable read

```
mydb=> BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
BEGIN
mydb=*> INSERT INTO sales.sale (id, date, employee, customer, bonus, bonus_used) VALUES (5, '2023-06-23 13:00:00', 3,
 3, 0, 0);
INSERT 0 1
mydb=*> SELECT * FROM sales.sale WHERE id = 5
mydb-*> ;
 id |        date         | employee | customer | bonus | bonus_used
----+---------------------+----------+----------+-------+------------
  5 | 2023-06-23 13:00:00 |        3 |        3 |     0 |          0
(1 row)

mydb=*> COMMIT;
COMMIT
```

Рис. 5. Запись из sales_role

```
mydb=> SELECT * FROM sales.sale WHERE id = 5;
 id | date | employee | customer | bonus | bonus_used
----+------+----------+----------+-------+------------
(0 rows)

mydb=> SELECT * FROM sales.sale WHERE id = 5;
 id |        date         | employee | customer | bonus | bonus_used
----+---------------------+----------+----------+-------+------------
  5 | 2023-06-23 13:00:00 |        3 |        3 |     0 |          0
(1 row)
```

Рис. 6. Чтение staff_role до и после завершения транзакции

Изоляция транзакции уровня REPEATABLE READ обеспечивает одинаковое чтени вне зависимости меняется ли таблица или нет во время произведения транзакции.

```
mydb=> BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
BEGIN
mydb=*> SELECT * FROM sales.sale WHERE id = 7;
 id | date | employee | customer | bonus | bonus_used
----+------+----------+----------+-------+------------
(0 rows)
```

Рис. 7. Открываем транзакцию на staff_role, обязательно читаем

```
mydb=> BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
BEGIN
mydb=*> INSERT INTO sales.sale (id, date, employee, customer, bonus, bonus_used) VALUES (7, '2023-06-23 15:00:00', 3,
 3, 0, 0);
INSERT 0 1
mydb=*> COMMIT;
COMMIT
```

Рис. 8. Добавляем запись с sales_role

```
mydb=*> SELECT * FROM sales.sale WHERE id = 7;
 id | date | employee | customer | bonus | bonus_used
----+------+----------+----------+-------+------------
(0 rows)

mydb=*> ROLLBACK;
ROLLBACK
```

Рис. 9. Запись не видна из транзакции staff_role

3. Serializable

Исполняет паралелльные транзакции как будто бы они происходят отдельно.

```
mydb=> BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
BEGIN
mydb=*> INSERT INTO sales.sale (id, date, employee, customer, bonus, bonus_used) VALUES (8, '2023-06-23 16:00:00', 3,
 3, 0, 0);
INSERT 0 1
mydb=*> SELECT * FROM sales.sale WHERE id = 8;
 id |        date         | employee | customer | bonus | bonus_used
----+---------------------+----------+----------+-------+------------
  8 | 2023-06-23 16:00:00 |        3 |        3 |     0 |          0
(1 row)

mydb=*> COMMIT;
COMMIT
```

Рис. 10. Запись из sales_role

```
mydb=> SELECT * FROM sales.sale WHERE id = 8;
 id | date | employee | customer | bonus | bonus_used
----+------+----------+----------+-------+------------
(0 rows)

mydb=> SELECT * FROM sales.sale WHERE id = 8;
 id |        date         | employee | customer | bonus | bonus_used
----+---------------------+----------+----------+-------+------------
  8 | 2023-06-23 16:00:00 |        3 |        3 |     0 |          0
(1 row)
```

Рис. 11. Чтение staff_role до и после завершения транзакции

В отличие если мы вставляем две строки с одинаковым primary key (или другим уникальным значением), то в случае REPEATABLE READ обе транзакции не выдадут ошибку до их завершения (COMMIT). А в случае SERIALIZABLE, ошибка будет на этапе попытки вставки.

Заключение

В данной лабораторной работе были рассмотрены такие важные аспекты PostgreSQ изоляция транзакций Также была произведена опытная проверка их функционирования.