ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ

Кафедра №42
«Криптология и кибербезопасность»

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский ядерный университет «МИФИ»»

ЛАБОРАТОРНАЯ РАБОТА №2-1:

«Транзакции. Изоляция транзакций.»

Аверин Владислав

Группа: Б19-505

Октябрь, 2022

Содержание

Цель работы

Изучение механизмов базы данных, обеспечивающих целостность данных в условиях многопользовательского доступа.

Ход работы

1. Создать дополнительного пользователя в разрабатываемой подключаемой базе данных (один пользователь уже должен существовать). Необходимо предоставить этому пользователю доступ к разработанным таблицам. Для этого допустимо предоставить новому пользователю привилегии ***SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE, DELETE ANY TABLE***. Можно также предоставить доступ к каждой таблице индивидуально;
2. Убедиться, что созданный пользователь имеет доступ к созданным ранее таблицам. Для этого попробовать выполнить инструкцию ***SELECT***. Важно, что для доступа к таблицам других пользователей необходимо явно указать владельца таблицы: ***SELECT * FROM testuser.testtable;***
3. Подключиться к экземпляру базы данных от лица каждого из пользователей одновременно (потребуется два одновременно работающих процесса SQL*Plus);
4. Начать транзакцию от лица первого пользователя; для этого выполнить изменение данных в таблице;
5. Выполнить запрос к изменённым данным от имени обоих пользователей. Сравнить результаты и объяснить результаты сравнения;
6. Выполнить откат транзакции первого пользователя с помощью инструкции ***ROLLBACK***. Повторно выполнить п. 5;
7. От лица первого пользователя выполнить ещё одно изменение данных в таблицах. Выполнить фиксацию транзакции при помощи инструкции ***COMMIT***. Повторно выполнить п. 5;
8. Если от лица второго пользователя начата транзакция, завершить её с помощью инструкции ***ROLLBACK*** или ***COMMIT***. Начать от лица данного пользователя новую сериализованную транзакцию, используя инструкцию
9. Повторить пп. 7 и 5;
10. Оформить отчёт.

1. *Создание пользователя*

Создадим стандартного пользователя от имени администратора

```
SQL> show user
USER is "SYS"
SQL> CREATE USER commituser
  2  IDENTIFIED BY[redacted]
  3  DEFAULT TABLESPACE users
  4  TEMPORARY TABLESPACE temp
  5  QUOTA 100M ON users;

User created.
```

И выдадим ему список необходимых привилегий для доступа ко всем сущностям БД:

```
SQL> GRANT CREATE SESSION TO commituser;

Grant succeeded.

SQL>
```

```
SQL> GRANT SELECT ANY TABLE TO commituser;

Grant succeeded.

SQL> GRANT INSERT ANY TABLE TO commituser;

Grant succeeded.

SQL> GRANT UPDATE ANY TABLE TO commituser;

Grant succeeded.

SQL> GRANT DELETE ANY TABLE TO commituser;

Grant succeeded.

SQL>
```

(Note: с точки зрения ИБ, данный подход категорически не рекомендуется, т.к. нужно придерживаться концепции WhiteLists: запрещать все, кроме того, что явно разрешается. А тут мы позволили нашему пользователю делать со всеми сущностями практически все, хотя необходимо было дать доступ только к определенным таблицам. Однако для данной цели у нас есть лаба 2-2 :) Так что чтобы не сгребать все в одну кучу, просто оставим здесь это как замечание)

Перейдем в профиль нового пользователя и сделаем тестовый запрос из созданной другим пользователем таблицы:

```
C:\Users\vladi>sqlplus commituser/1204@"127.0.0.1:1521/XEPDB1"

SQL*Plus: Release 18.0.0.0.0 - Production on Sat Oct 1 18:41:39 2022
Version 18.4.0.0.0

Copyright (c) 1982, 2018, Oracle.  All rights reserved.


Connected to:
Oracle Database 18c Express Edition Release 18.0.0.0.0 - Production
Version 18.4.0.0.0

SQL> show user
USER is "COMMITUSER"
SQL> SELECT * FROM Infernal.AssignedCases
  2
SQL> SELECT * FROM Infernal.AssignedCases;

EMPLOYEE_ID    CASE_ID
----------- ----------
          1         32
          1         33
          2          5
          2         10
          2         11
          2         12
          2         13
          2         19
```

(Note x2: по понятным причинам без явного указания принадлежности сущности sql не поймет, что это за сущность)

```
SQL> SELECT * FROM AssignedCases;
SELECT * FROM AssignedCases
              *
ERROR at line 1:
ORA-00942: table or view does not exist


SQL>
```

2. Транзакция от имени нового пользователя

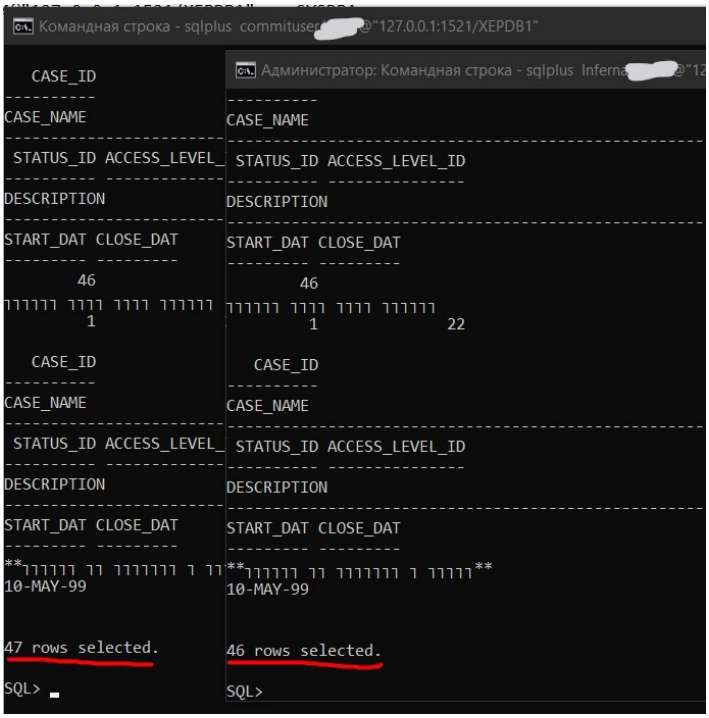
Подключаемся к обоим пользователям (Infernal – владелец таблиц, и commituser – получивший доступ) и выполняем от лица commituser обычный insert:

```
SQL> INSERT INTO Infernal.Cases
  2  (case_id, case_name, status_id, access_level_id, description, start_date)
  3  VALUES
  4  (105, 'teststring', 1, 11, 'something something', TO_DATE('17-MAY-2020'));

1 row created.

SQL>
```

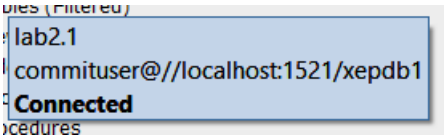
Как видно, строка якобы “создалась” (технически – да, фактически в самой таблице – нет):

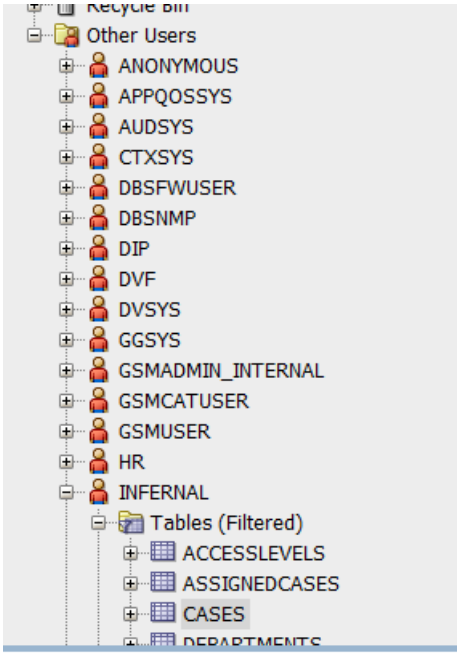
```
   CASE_ID
----------
CASE_NAME
--------------------------------------------------------------------------------
 STATUS_ID ACCESS_LEVEL_ID
---------- ---------------
DESCRIPTION
--------------------------------------------------------------------------------
START_DAT CLOSE_DAT
--------- ---------
       105
teststring
         1             11
```

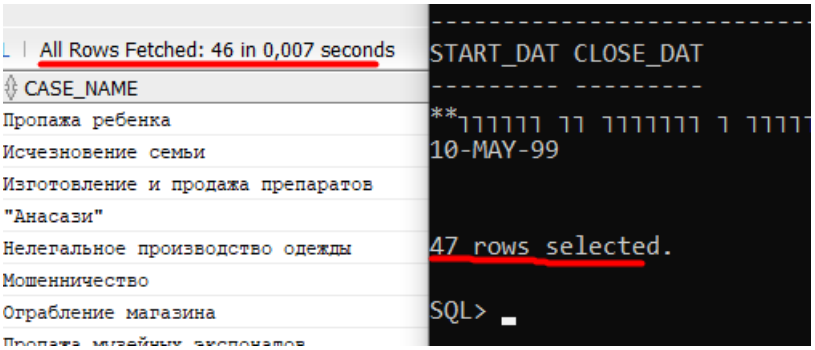
Однако никаких изменений таблицы со стороны ее владельца мы не увидим:



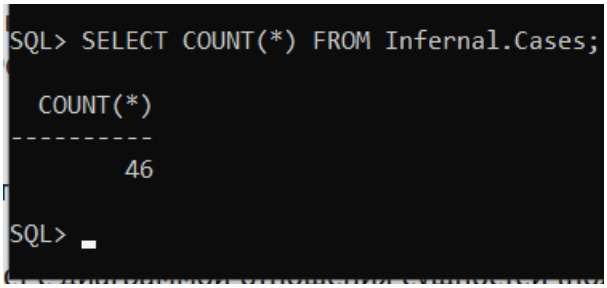
(Note x3: несмотря на то, что от имени одного профиля могут подключиться одновременно несколько пользователей, судя по всему, для каждой сессии транзакция будет своя, не зависящая от других локальных транзакций. Хотя это странно, т.к. в моем понимании блоки должны же как-то идентифицироваться – кем вызвано каждое изменение? Однако одновременная работа в SQLPlus и sql loader показала, что вторая утилита ничего не знает о действиях, которые выполняет первая, и может работать отдельно, будто бы из-под другого профиля)



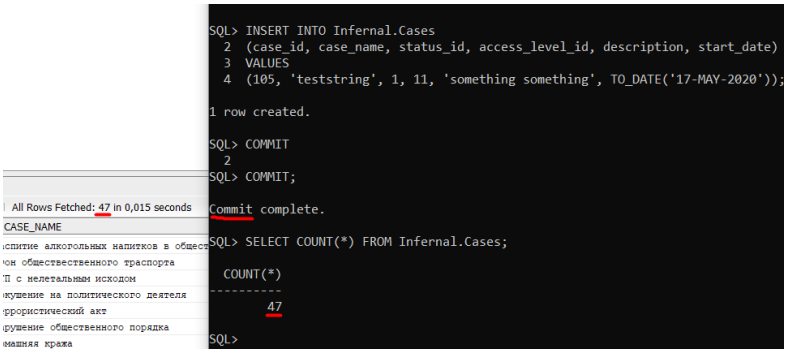


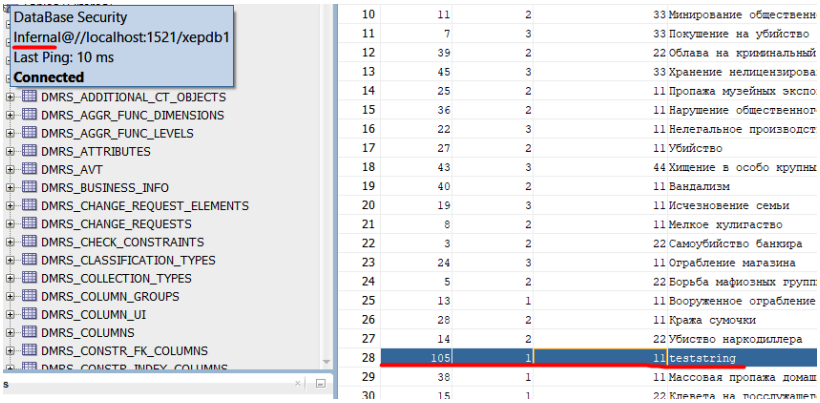


При применении инструкции ***ROLLBACK*** изменения, внесенные в таблицу пользователем commituser, пропадут, т.к. она выполняет “откат” (полный или частичный до последнего savepoint’a) БД до исходного состояния:



При фиксации изменений БД подтверждает изменения, и теперь они будут видны всем пользователям:





3. *Сериализованная транзакция*

Теперь выполним изоляцию выполняемой транзакции (чтобы какой-нибудь Петя, который параноидально коммитит каждое изменение параллельно с нами, не мог “вклиниться” в нашу работу, так, будто его и нет):

```
SQL> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Transaction set.
```

При добавлении новой информации основным пользователем, пользователь commituser сможет использовать данную таблицу:

```
       47

SQL> INSERT INTO Infernal.Cases
  2  (case_id, case_name, status_id, access_level_id, description, start_date)
  3  VALUES
  4  (106, 'testing2', 1, 12, 'ss', TO_DATE('16-MAY-2020'));
                                  ...Я не завис, я просто жду :(
```

```
SQL> INSERT INTO Cases
  2  (case_id, case_name, status_id, access_level_id, descrip
  3  VALUES
  4  (106, 'testing2', 1, 12, 'ss', TO_DATE('16-MAY-2020'));

1 row created.
```

Только после коммита того, кто первым начал ней работу (причем, все изменения с таблицей commituser также увидит):

```
  3  VALUES
  4  (106, 'testing2', 1, 12, 'ss', TO_DATE('16-MAY-2020'));
INSERT INTO Infernal.Cases
*
ERROR at line 1:
ORA-00001: unique constraint (INFERNAL.CASES_PK) violated
```

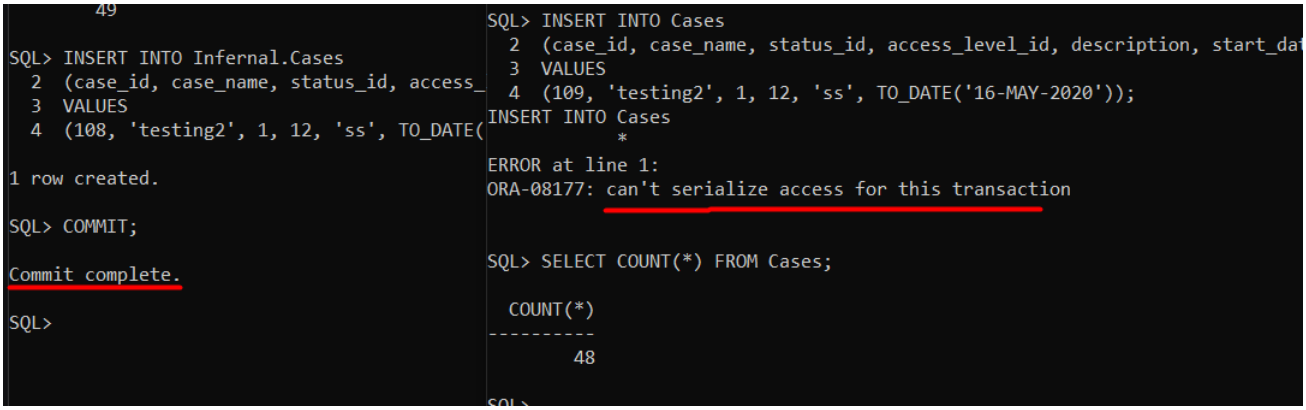
```
SQL> COMMIT;

Commit complete.

SQL>
```

Если же пользователь установит сериализованную транзакцию перед тем, как в таблицу поступит новая зафиксированная информация, то этот пользователь, даже после коммита изменений, будет работать со “старой” версией (взятой из CR-клона).

```
1 row created.

SQL> SELECT COUNT(*) FROM Infernal.Cases;

  COUNT(*)
----------
        49

SQL> COMMIT;

Commit complete.

SQL>
```

```
SQL> SELECT COUNT(*) FROM Cases;

  COUNT(*)
----------
        48

SQL> SELECT COUNT(*) FROM Cases;

  COUNT(*)
----------
        48

SQL>
```

Однако модифицировать измененную во время этой транзакции кем-то другим таблицу, мы все равно не сможем (логично: как мы гарантируем, что изменяя старый CR-клон таблицы, мы не пойдем в противоречия с только что зафиксированными изменениями в ней):



Таким образом, сериализованная транзакция позволяет нам работать со “старой” версией БД, абстрагируясь от внесенных в нее во время нашей работы внешних изменений. Но по понятным причинам, в случае потенциальной угрозы целостности данных, СУБД все также блокирует доступ к ее сущностям до окончания работы с ними других пользователей. Такова цена за общий доступ к данным :)

Выводы

В ходе данной лабораторной работы были изучены механизмы обеспечения синхронизированной работы с базой данных нескольких пользователей, получены практические знания о работе транзакций и приобретены навыки в регулировании данного процесса.