МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

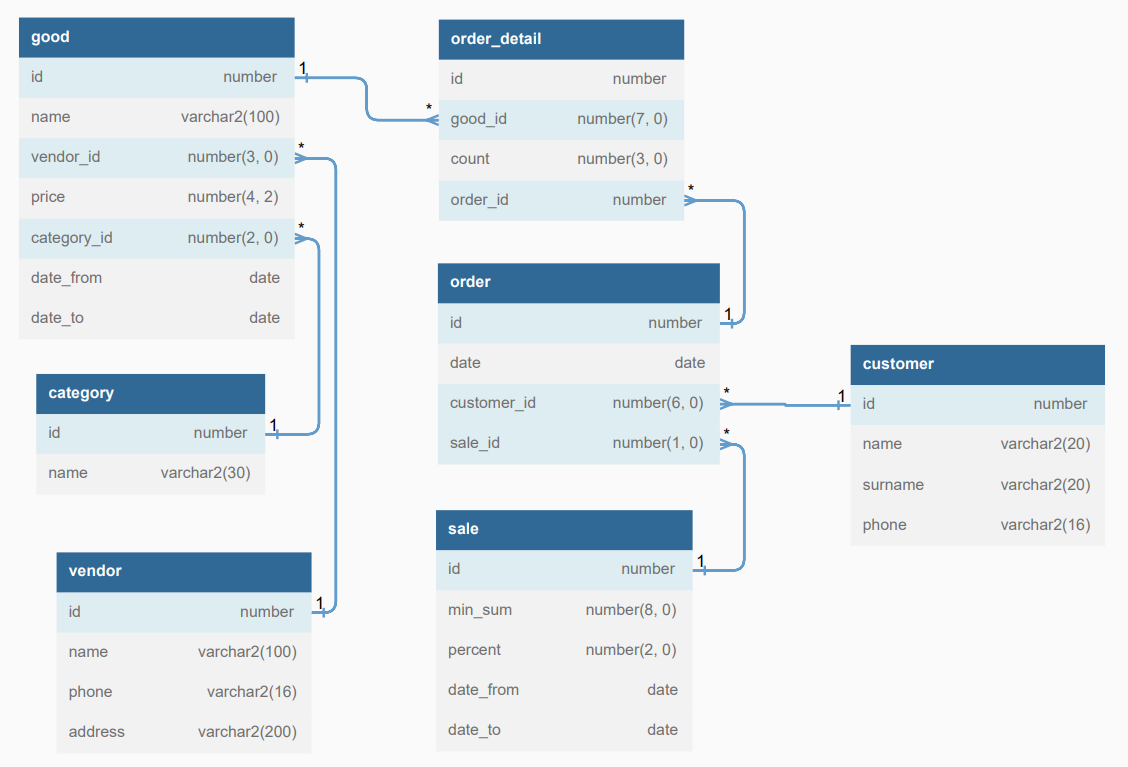
Федеральное государственное автономное образовательное учреждение

высшего образования

«Национальный исследовательский ядерный университет «МИФИ»»

ЛАБОРАТОРНАЯ РАБОТА №2-1:

«Транзакции. Изоляция транзакций.»



Выполнила студентка группы Б19-515

Щербакова Александра

Москва, 2022 г.

**1. Создание и базы данных.**

Так как с предыдущего семестра база данных не сохранилась, все операции выполняются заново.

1. Создана новая PDB shop\_pdb с администратором shop\_admin, которому предоставлены все возможные привилегии внутри shop\_pdb (код см. в Приложении А).

Изображение выглядит как текст

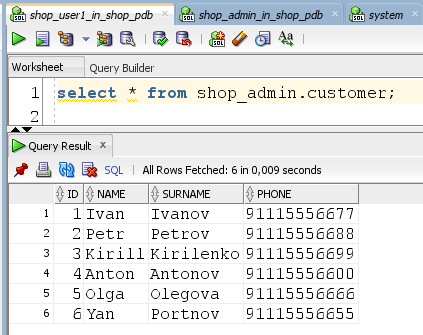
Автоматически созданное описание

1. Созданы таблицы, описывающие предметную область – продуктовый магазин. ER-диаграммы представлены на титульном листе. Код в приложении B.
2. Таблицы заполнены минимально необходимым набором данных. Код в приложении C.

**2. Создание второго пользователя.**

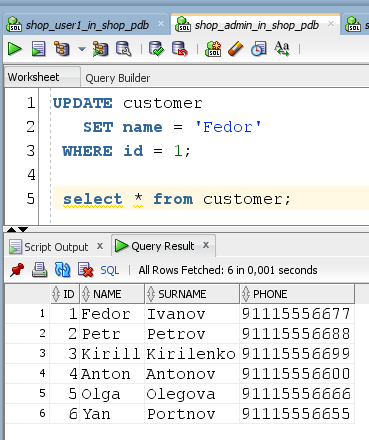
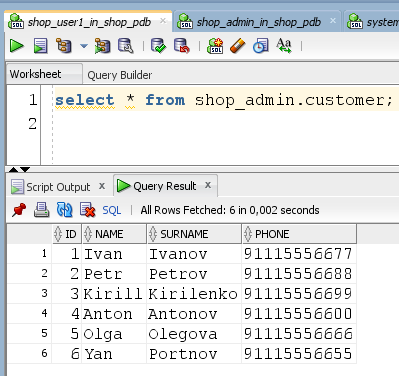
Создан пользователь базы данных shop\_user1 с ограниченными привилегиями: только читать и добавлять/удалять строки из таблиц. Код в приложении D.

Чтобы убедиться, что привилегии работают корректно, выполнен тестовый запрос:

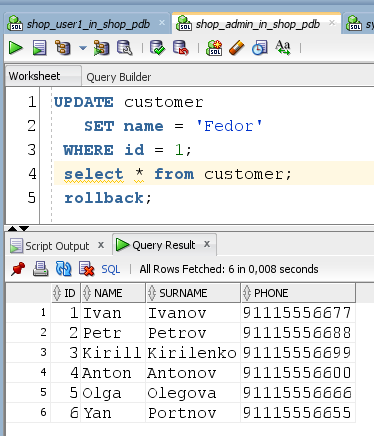


**3. Работа с транзакциями.**

1. От имени администратора создана транзакция по изменению данных клиента (изменить имя первого клиента с ‘Ivan’ на ‘Fedor’). При просмотре таблицы от администратора изменения видны (так как он эту транзакцию создал), а от пользователя shop\_user1 – нет (так как транзакция не была зафиксирована).

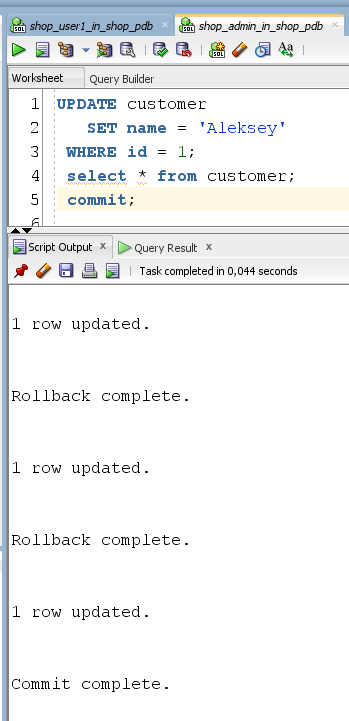
* *

1. Выполнен откат (*rollback*) транзакции. Для обоих пользователей таблица вернулась к состоянию до начала транзакции.

 Изображение выглядит как текст

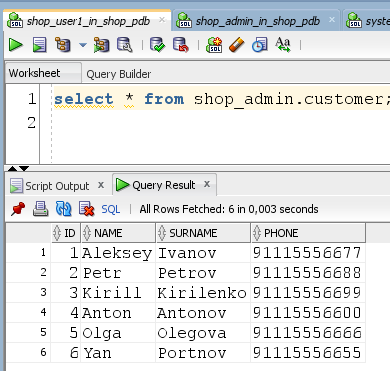
Автоматически созданное описание

1. От имени администратора создана транзакция по изменению данных клиента (изменить имя первого клиента с ‘Ivan’ на ‘Aleksey’).

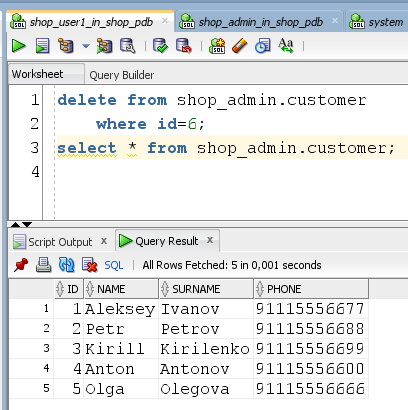


Для данной транзакции выполнена фиксация (*commit*). Результат – для обоих пользователей сохранились изменения в таблице.

Изображение выглядит как текст

Автоматически созданное описание

1. От имени пользователя shop\_user1 создана транзакция «удалить клиента Yan Portnov» (последний в списке).

 Изображение выглядит как текст

Автоматически созданное описание

При откате транзакции для обоих пользователей таблица вернулась к состоянию до начала транзакции.

Изображение выглядит как текст

Автоматически созданное описание Изображение выглядит как текст

Автоматически созданное описание

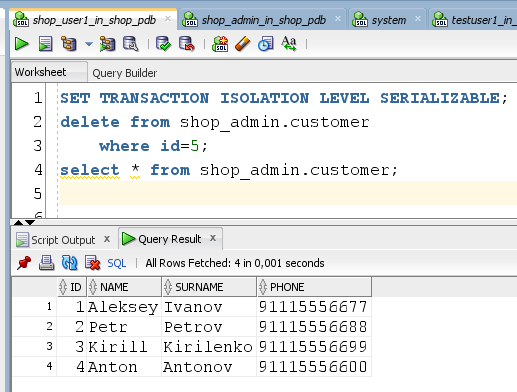
При фиксации транзакции изменения сохранились для обоих пользователей.

Изображение выглядит как текст

Автоматически созданное описание Изображение выглядит как текст

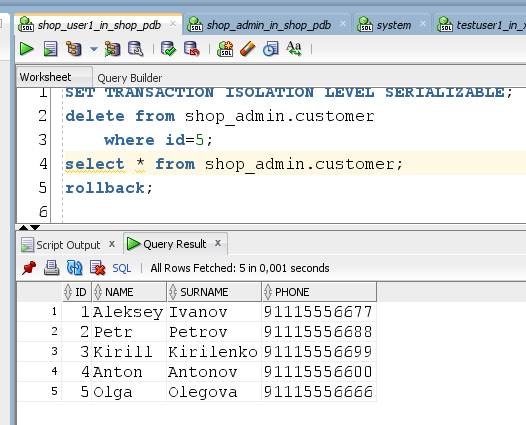
Автоматически созданное описание

1. От лица второго пользователя shop\_user1 начата сериализованная транзакция (*SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;).* Задача транзакции – удалить из таблицы клиента ‘Olga Olegova’ (последняя в списке).

 Изображение выглядит как текст

Автоматически созданное описание

При откате транзакции для обоих пользователей таблица вернулась к состоянию до начала транзакции.

 Изображение выглядит как текст

Автоматически созданное описание

При фиксации транзакции изменения сохранились для обоих пользователей.

**Изображение выглядит как текст

Автоматически созданное описание Изображение выглядит как текст

Автоматически созданное описание**

Вывод: результаты для уровней изоляции READ COMMITTED и SERIALIZABLE идентичны, значит, рассмотрения только операции select недостаточно, чтобы выявить разницу между ними.

1. **Одновременное изменение данных в таблице.**

READ COMMITTED: это уровень по умолчанию. Каждый запрос (в рамках транзакции) может видеть только данные, зафиксированные до запуска этого *запроса*. Таким образом, если дважды запущен один и тот же запрос в одной и той же транзакции, могут быть разные результаты и фантомы.

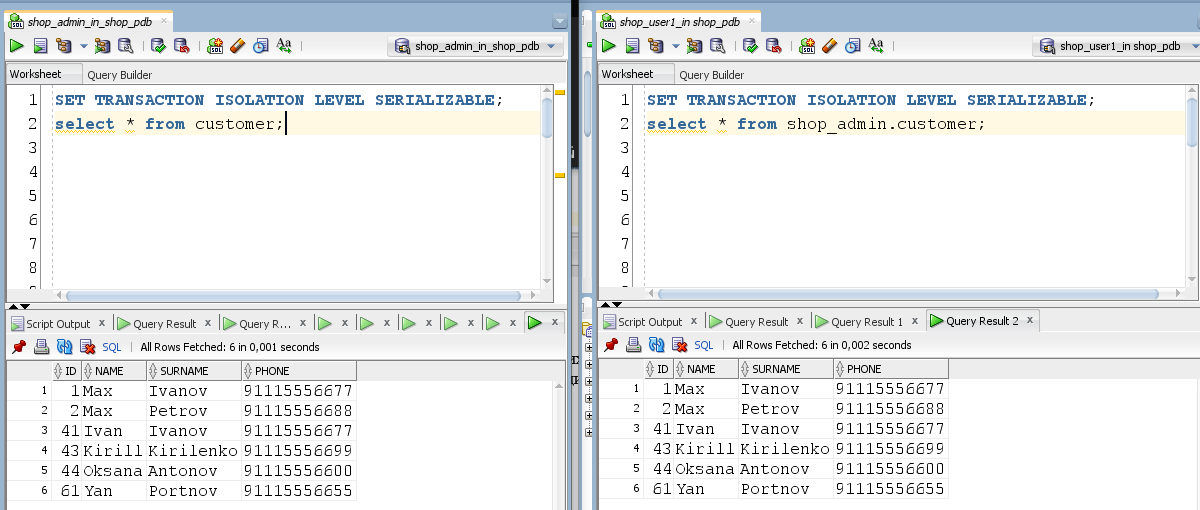
Обеспечивает защиту от *грязного чтения*, при этом присутствует *неповторяющееся чтение* (когда видны обновленные и удаленные строки).

SERIALIZABLE: каждый запрос (в рамках транзакции) может видеть только данные, зафиксированные до начала этой *транзакции*, плюс изменения, внесенные в саму транзакцию.

Обеспечивает самую высокую согласованность, избавляя от эффекта *чтения фантомов.* Транзакции полностью изолируются друг от друга, каждая выполняется так, как будто параллельных транзакций не существует. Пока не завершена одна транзакция, другие транзакции ждут ее завершения.

1. *ISOLATION LEVEL SERIALIZABLE*

Состояние таблицы до начала транзакций.

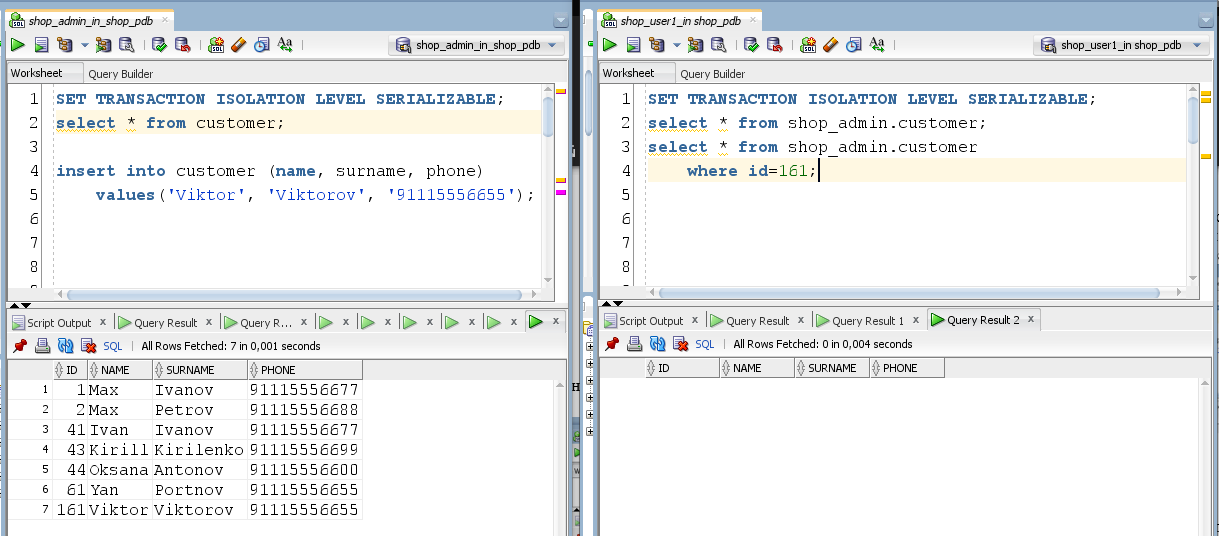


Пользователь shop\_admin открыл транзакцию на добавление строки – ему видны его изменения. При этом пользователь shop\_user1 видит состояние таблицы до начала этой транзакции.

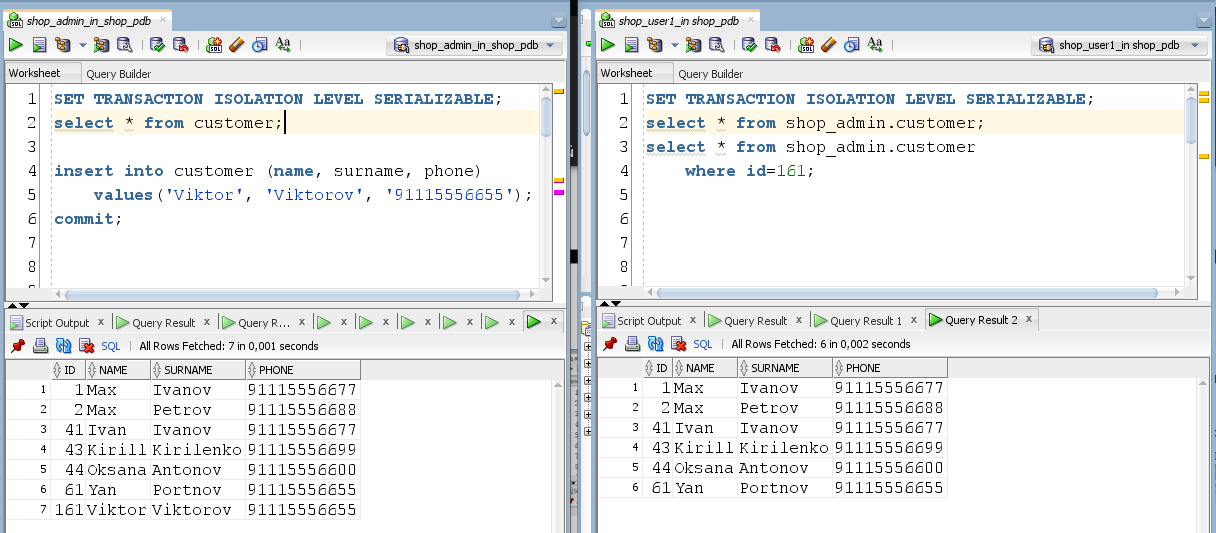
Изображение выглядит как текст

Автоматически созданное описание

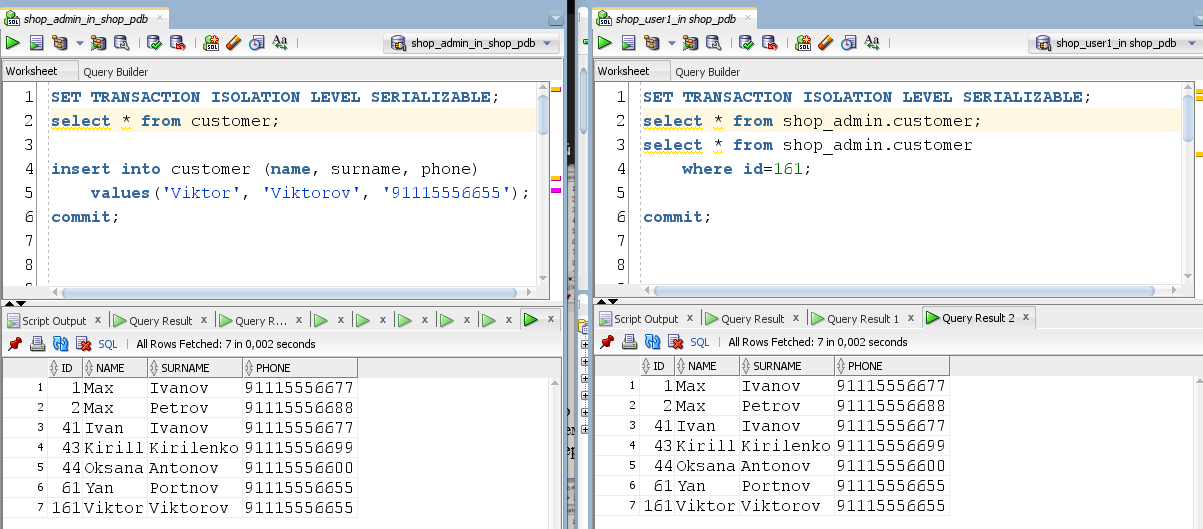
Shop\_user1 пытается обратиться к добавленной, но еще не закоммиченной строке, результат – она не видна.



Shop\_admin коммитит транзакцию, но shop\_user1 по-прежнему не видит изменений в таблице. Это связано с тем, что запрос select – тоже транзакция, которую необходимо завершить (commit/roollback), чтобы увидеть чужие коммиты.

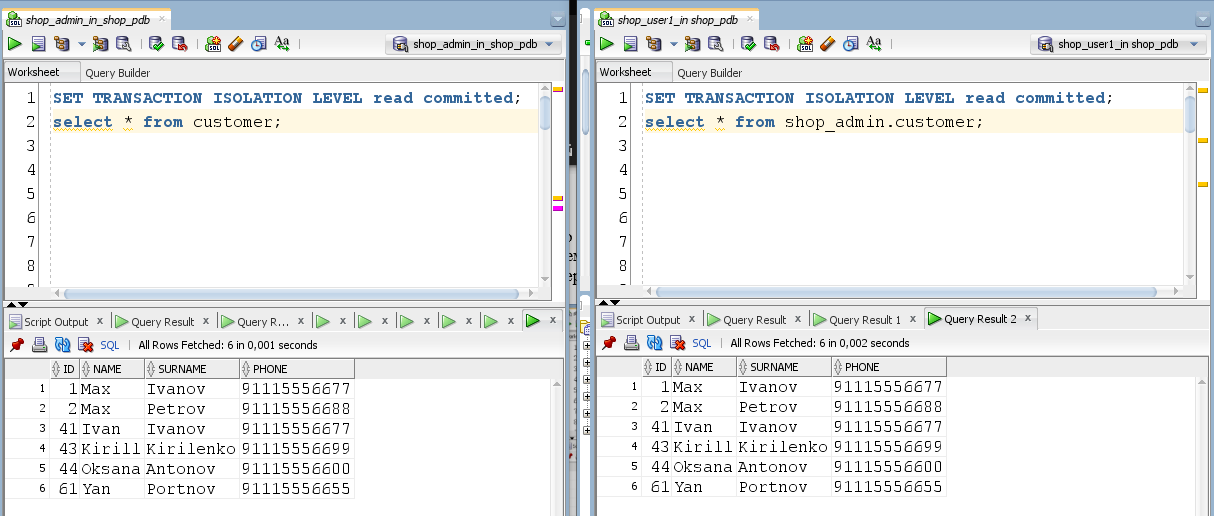


После того, как транзакции справа и слева завершены, оба пользователя видят добавленную строку.

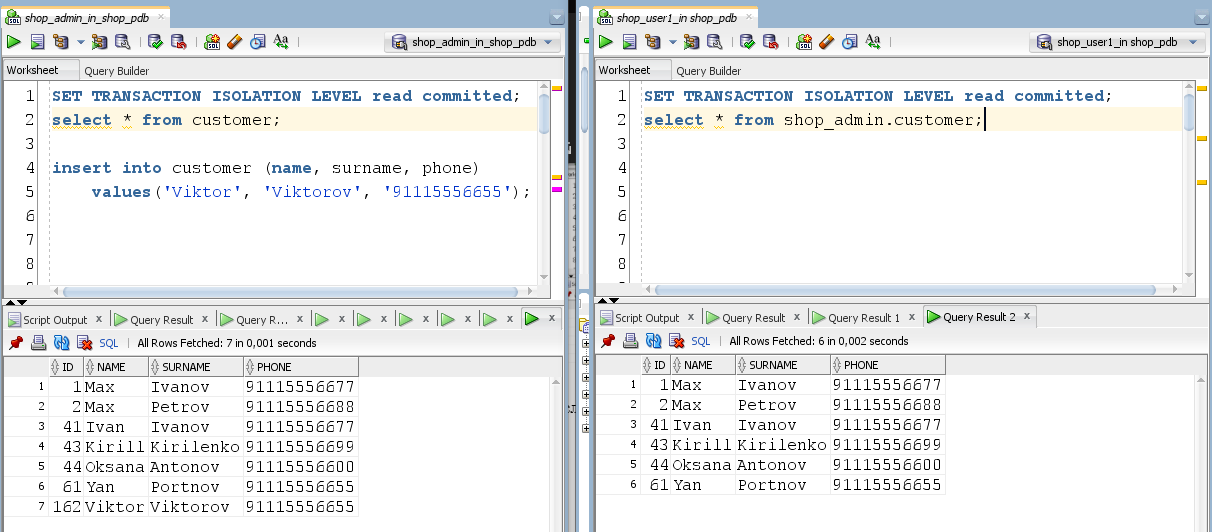


1. *ISOLATION LEVEL READ COMMITTED*

Состояние таблицы до начала транзакций.



Пользователь shop\_admin открыл транзакцию – ему видны его изменения. При этом пользователь shop\_user1 видит состояние таблицы до начала этой транзакции.



При чтении всей таблицы пользователем справа незакоммиченные изменения не видны. Однако при обращении к добавляемой строке по id, а не через чтение всей таблицы, пользователь справа может увидеть незакоммиченную строку (так называемое *чтение фантома*). Ошибка может возникать из-за сохранения изменений в кеше, но она возникает не каждый раз (эксперимент проводился многократно), так что скриншоты отсутствуют.

В большинстве случаев в данном эксперимента чтения фантома не возникало, и пользователь справа мог увидеть изменения только после коммита слева.

**5. Заключение**

В данной работе были изучены механизмы базы данных, обеспечивающие целостность данных в условиях многопользовательского доступа. Был создан новый пользователь для моделирования работы нескольких пользователей в базе данных. Были изучены основы транзакций, их подтверждение и отмена.

В работе рассмотрены два уровня изоляции: READ COMMITTED и SERIALIZABLE. Проведен эксперимент для двух данных уровней и выявлены различия.

**Приложение A. Создание базы данных и пользователей.**

/ as sysdba

create pluggable database SHOP\_PDB admin user shop\_admin identified by shop\_admin\_pas

default tablespace SHOP\_PDB\_USERS

datafile 'C:\APP\ALEXM\PRODUCT\18.0.0\ORADATA\XE\SHOP\_PDB\SHOP\_PDB\_USERS01.dbf' size 250m autoextend on

file\_name\_convert=('C:\APP\ALEXM\PRODUCT\18.0.0\ORADATA\XE\pdbseed','C:\APP\ALEXM\PRODUCT\18.0.0\ORADATA\XE\SHOP\_PDB');

show pdbs;

alter pluggable database SHOP\_PDB open;

show pdbs;

ALTER SESSION SET CONTAINER=shop\_pdb;

GRANT

any privilege

to shop\_admin

CONTAINER=CURRENT;

**Приложение B. Создание таблиц.**

CREATE TABLE order\_detail (

id number generated always as identity primary key,

good\_id number(9, 0) NOT NULL,

count number(3, 0) NOT NULL,

order\_id number NOT NULL

);

CREATE TABLE orders (

id number generated always as identity primary key,

"date" date NOT NULL,

customer\_id number(6, 0) NOT NULL,

sale\_id number(3, 0)

);

CREATE TABLE good (

id number generated always as identity primary key,

name varchar2(100) NOT NULL,

vendor\_id number(4, 0) NOT NULL,

price number(6, 2) NOT NULL,

category\_id number(3, 0) NOT NULL,

date\_from date NOT NULL,

date\_to date NOT NULL

);

CREATE TABLE vendor (

id number generated always as identity primary key,

name varchar2(100) NOT NULL,

phone varchar2(16) NOT NULL,

address varchar2(200)

);

CREATE TABLE category (

id number generated always as identity primary key,

name varchar2(30) NOT NULL

);

CREATE TABLE customer (

id number generated always as identity primary key,

name varchar2(20) NOT NULL,

surname varchar2(20) NOT NULL,

phone varchar2(16)

);

CREATE TABLE sale (

id number generated always as identity primary key,

min\_sum number(8, 0) NOT NULL,

percent number(3, 1) NOT NULL,

date\_from date NOT NULL,

date\_to date NOT NULL

);

ALTER TABLE order\_detail ADD FOREIGN KEY (good\_id) REFERENCES good (id);

ALTER TABLE order\_detail ADD FOREIGN KEY (order\_id) REFERENCES orders (id);

ALTER TABLE good ADD FOREIGN KEY (vendor\_id) REFERENCES vendor (id);

ALTER TABLE good ADD FOREIGN KEY (category\_id) REFERENCES category (id);

ALTER TABLE orders ADD FOREIGN KEY (customer\_id) REFERENCES customer (id);

ALTER TABLE orders ADD FOREIGN KEY (sale\_id) REFERENCES sale (id);

**Приложение C. Заполнение таблиц.**

-- category

insert into category (name) values ('fruit');

insert into category (name) values ('vegetables');

insert into category (name) values ('chocolate');

insert into category (name) values ('water');

-- vendor

insert into vendor (name, phone, address) values ('fruit vendor', '81234567788', 'Moscow');

insert into vendor (name, phone, address) values ('fruit and vegetables vendor', '81234568899', 'Krasnodar');

insert into vendor (name, phone, address) values ('chocolate vendor 1', '81112223344', 'Podolsk');

insert into vendor (name, phone) values ('chocolate vendor 2', '83334445566');

insert into vendor (name, phone) values ('water vendor 1', '89998887766');

-- customer

insert into customer (name, surname, phone)

values ('Ivan', 'Ivanov', '91115556677');

insert into customer (name, surname, phone)

values ('Petr', 'Petrov', '91115556688');

insert into customer (name, surname, phone)

values ('Kirill', 'Kirilenko', '91115556699');

insert into customer (name, surname, phone)

values ('Anton', 'Antonov', '91115556600');

insert into customer (name, surname, phone)

values ('Olga', 'Olegova', '91115556666');

insert into customer (name, surname, phone)

values ('Yan', 'Portnov', '91115556655');

declare

c\_date\_to date := to\_date('01-01-2023', 'dd-mm-yyyy');

c\_date\_from date := to\_date('01-01-2022', 'dd-mm-yyyy');

begin

-- good

insert into good (name, vendor\_id, price, category\_id, date\_from, date\_to)

values ('apple', 1, 30, 1, c\_date\_from, c\_date\_to);

insert into good (name, vendor\_id, price, category\_id, date\_from, date\_to)

values ('banana', 1, 20, 1, c\_date\_from, c\_date\_to);

insert into good (name, vendor\_id, price, category\_id, date\_from, date\_to)

values ('cucumber', 2, 25, 2, c\_date\_from, c\_date\_to);

insert into good (name, vendor\_id, price, category\_id, date\_from, date\_to)

values ('alionka', 3, 70, 3, c\_date\_from, c\_date\_to);

insert into good (name, vendor\_id, price, category\_id, date\_from, date\_to)

values ('alpen gold', 4, 70, 3, c\_date\_from, c\_date\_to);

insert into good (name, vendor\_id, price, category\_id, date\_from, date\_to)

values ('sviatoy istochnik', 5, 50, 4, c\_date\_from, c\_date\_to);

-- sale

insert into sale (min\_sum, percent, date\_from, date\_to)

values (0, 0.5, c\_date\_from, c\_date\_to);

insert into sale (min\_sum, percent, date\_from, date\_to)

values (30000, 1, c\_date\_from, c\_date\_to);

insert into sale (min\_sum, percent, date\_from, date\_to)

values (50000, 2, c\_date\_from, c\_date\_to);

insert into sale (min\_sum, percent, date\_from, date\_to)

values (100000, 5, c\_date\_from, c\_date\_to);

insert into sale (min\_sum, percent, date\_from, date\_to)

values (2000000, 7, c\_date\_from, c\_date\_to);

end;

-- orders

insert into orders ("date", customer\_id, sale\_id)

values (SYSDATE, 1, 1);

insert into orders ("date", customer\_id, sale\_id)

values (SYSDATE, 2, 1);

insert into orders ("date", customer\_id, sale\_id)

values (SYSDATE, 2, 1);

-- order\_detail

insert into order\_detail (good\_id, count, order\_id)

values (1, 5, 1);

insert into order\_detail (good\_id, count, order\_id)

values (2, 10, 1);

insert into order\_detail (good\_id, count, order\_id)

values (3, 2, 1);

insert into order\_detail (good\_id, count, order\_id)

values (4, 7, 2);

insert into order\_detail (good\_id, count, order\_id)

values (5, 1, 2);

insert into order\_detail (good\_id, count, order\_id)

values (6, 2, 2);

insert into order\_detail (good\_id, count, order\_id)

values (1, 3, 2);

insert into order\_detail (good\_id, count, order\_id)

values (1, 20, 3);

**Приложение D. Создание пользователя с ограниченными привилегиями.**

-- / as sysbda

ALTER SESSION SET CONTAINER=shop\_pdb;

CREATE USER shop\_user1

IDENTIFIED BY shop\_user1\_pas

TEMPORARY TABLESPACE temp

CONTAINER=CURRENT;

GRANT

create session,

select any table,

insert any table,

update any table,

delete any table,

to shop\_user1;