

RĪGAS TEHNISKĀ UNIVERSITĀTE

Datorzinātnes, informācijas tehnoloģijas un enerģētikas fakultāte

4.praktiskais darbs

**Tranzakciju vadība**

Izstrādāja: Natans Šalamberidze

171RMC203

Pārbaudīja: lektors Jānis Eiduks

2023./2024. māc. Gads

Darba mērķis

Izpildot tranzakcijas koplietošanas režīmā ar dažādiem izolācijas līmeņiem, pierādīt, ka tiek novērstas nevēlamas situācijas: netīrā lasīšana, neatkārtojamā lasīšana, fantoma lasīšana un zaudētā atjaunojuma problēma.

Darba uzdevums

Lai šādas situācijas novērstu, tranzakciju izpildei tiek definēti vairāki to izolācijas līmeņi. Praktiskajā darbā tiek pētīti apstiprinātās lasīšanas (READ COMMITED) un serializējamais (SERIALIZABLE) tranzakciju izolācijas līmeņi un to vadība.

1. Jāizprojektē piemērs apstiprinātās lasīšanas izolācijas līmeņa darbības ilustrēšanai ar divām transakcijām. Piemēra attēlojums jādod tabulas formā. Pirmājā kolonā ir secīgi 1. tranzakcijas darbības, otrā kolona parāda datu bāzes datu tekošo stāvokli, trešajā kolonā ir secīgi 2. tranzakcijas darbības, ceturtajā kolonā - paskaidrojumi par atsevišķo komandu darbību. Tabulā jānorāda komandas, kas demonstrē apstiprinātās lasīšanas izolācijas līmeņa realizēšanu.

2. Jāizprojektē piemērs serializējamā izolācijas līmeņa darbības ilustrēšanai ar divām transakcijām. Piemēra attēlojums jādod tabulas formā. Pirmājā kolonā ir secīgi 1. tranzakcijas darbības, otrā kolona parāda datu bāzes datu tekošo stāvokli, trešajā kolonā ir secīgi 2. tranzakcijas darbības, ceturtajā kolonā - paskaidrojumi par atsevišķo komandu darbību. Tabulā jānorāda komandas, kas demonstrē serializējamā izolācijas līmeņa realizēšanu.

3. Jāveic transakciju kopdarbību analīze un skaidrojumi.

4. Jādefinē tabula transakciju izpildei un jāaizpilda tā ar datiem. Var izmantot tabulas no iepriekšējiem darbiem.

4. Jāizprojektē tranzakciju kopdarbības realizēšanu:

1) izveidojot vairākus lietotājus un no katra vienlaicīgi izpildot savu transakciju;

2) vienam lietotājam izveidojot vairākus savienojumus un no katra vienlaicīgi izpildot savu transakciju;

3) izmantojot aktīvās datu bāzes trigerus;

4) izmantojot redaktorus, kuros ērtāk realizēt un apskatīt paralēlās izpildes.

**1. Jāizprojektē piemērs apstiprinātās lasīšanas izolācijas līmeņa darbības ilustrēšanai ar divām transakcijām. Piemēra attēlojums jādod tabulas formā. Pirmājā kolonā ir secīgi 1. tranzakcijas darbības, otrā kolona parāda datu bāzes datu tekošo stāvokli, trešajā kolonā ir secīgi 2. tranzakcijas darbības, ceturtajā kolonā - paskaidrojumi par atsevišķo komandu darbību. Tabulā jānorāda komandas, kas demonstrē apstiprinātās lasīšanas izolācijas līmeņa realizēšanu.**

**Scenārijs:**

1. Transakcija 1 paaugstina T-kreklu daudzumu.
2. Transakcija 2 reizinā paaugstināto T-kreklu daudzumu un apstiprina izmaiņas.
3. Transakcija 1 lasa jauno, sareizināto T-kreklu daudzumu pēc Transakcijas 2 apstiprināšanas un pēc tam arī apstiprina savas izmaiņas.

**Transakcijas tabulas formātā:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Transakcijas 1 darbības** | **Datu bāzes stāvoklis** | **Transakcijas 2 darbības** | **Skaidrojumi** |
| BEGIN; |  |  | Transakcija 1 sākas. |
| UPDATE Inventory SET Quantity = Quantity + 50 WHERE ProductID = 1; | Daudzums pirms: 100, pēc: 150 |  | Transakcija 1 paaugstina T-kreklu daudzumu par 50. |
|  |  | BEGIN; | Transakcija 2 sākas. |
|  |  | UPDATE Inventory SET Quantity = Quantity \* 2 WHERE ProductID = 1; | Transakcija 2 reizinā T-kreklu daudzumu ar 2. |
|  |  | COMMIT; | Transakcija 2 apstiprina izmaiņas. |
| SELECT Quantity FROM Inventory WHERE ProductID = 1; | Jaunais daudzums: 300 |  | Transakcija 1 lasa jauno, sareizināto T-kreklu daudzumu: 300. |
| COMMIT; |  |  | Transakcija 1 apstiprina savas sākotnējās izmaiņas. |

**2. Jāizprojektē piemērs serializējamā izolācijas līmeņa darbības ilustrēšanai ar divām transakcijām. Piemēra attēlojums jādod tabulas formā. Pirmājā kolonā ir secīgi 1. tranzakcijas darbības, otrā kolona parāda datu bāzes datu tekošo stāvokli, trešajā kolonā ir secīgi 2. tranzakcijas darbības, ceturtajā kolonā - paskaidrojumi par atsevišķo komandu darbību. Tabulā jānorāda komandas, kas demonstrē serializējamā izolācijas līmeņa realizēšanu.**

**Scenārijs:**

1. Transakcija 1 (T1) rediģē daudzumu produktam ar ID = 1.
2. Transakcija 2 (T2) mēģina rediģēt to pašu produktu.

**Transakcijas tabulas formātā:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Transakcija 1 Darbības** | **Datu bāzes stāvoklis** | **Transakcija 2 Darbības** | **Skaidrojums** |
| BEGIN; SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; |  |  | Transakcija 1 sākas, iestatot serializējamo izolācijas līmeni. |
|  | Noliktava: Daudzums 30 (Dators) |  |  |
| UPDATE Noliktava SET Daudzums = Daudzums - 5 WHERE ProduktaID = 1; | Daudzums samazināts: 25 (Dators) |  | T1 palielina daudzums produktam Dators. |
|  |  | BEGIN; SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; | T2 sākas, iestatot serializējamo izolācijas līmeni. |
|  |  | UPDATE NOLIKTAVA SET Daudzums = Daudzums - 5 WHERE ProduktaID = 1; | T2 grib piešķirt jaunu daudzumu. |
| COMMIT; |  |  | T1 apstiprina izmaiņas. |
|  |  |  | T2 nevar piešķirt jaunu daudzumu. |
|  |  | COMMIT; | T2 apstiprina izmaiņas. |

**3. Jāveic transakciju kopdarbību analīze un skaidrojumi.**

### 1. "Read Committed" izolācijas līmeņa analīze

**Scenārijs:**

* **Transakcija 1** paaugstina T-kreklu daudzumu.
* **Transakcija 2** reizinā paaugstināto T-kreklu daudzumu un apstiprina izmaiņas.
* **Transakcija 1** lasa jauno, sareizināto T-kreklu daudzumu pēc Transakcijas 2 apstiprināšanas un pēc tam arī apstiprina savas izmaiņas.

**Analīze:**

* **Transakcija 1** sākumā veic T-kreklu daudzuma palielināšanu, kas nozīmē, ka daudzums tiek mainīts no 100 uz 150. Šajā laikā izmaiņas vēl nav apstiprinātas.
* **Transakcija 2** sākas un pēc Transakcijas 1 izmaiņu apstiprināšanas reizinā šo daudzumu ar 2, tādējādi gala rezultāts ir 300 T-krekli. Šīs izmaiņas tālāk tiek apstiprinātas.
* Kad **Transakcija 1** turpina, tā redz Transakcijas 2 apstiprinātās izmaiņas, kas atbilst "Read Committed" izolācijas līmeņa prasībām: redzamas tikai citas tranzakcijas apstiprinātās izmaiņas.
* Šis scenārijs efektīvi demonstrē, kā "Read Committed" izolācijas līmenis var novērst "dirty reads", bet nevar novērst citas fenomenus kā "non-repeatable reads" vai "phantom reads".

### 2. "Serializable" izolācijas līmeņa analīze

**Scenārijs:**

* **Transakcija 1 (T1)** rediģē daudzumu produktam ar ID = 1.
* **Transakcija 2 (T2)** mēģina rediģēt to pašu produktu.

**Analīze:**

* **Transakcija 1 (T1)** sāk darbu, izmantojot **SERIALIZABLE** izolācijas līmeni, kas nodrošina, ka visas izmaiņas, ko tā veic, būs redzamas tikai pēc to apstiprināšanas un neietekmēs citas paralēli notiekošas transakcijas. T1 samazina produktu daudzumu no 30 uz 25.
* **Transakcija 2 (T2)** sākas arī ar **SERIALIZABLE** izolācijas līmeni. Tā mēģina samazināt produktu daudzumu, bet, tā kā tā sākas pēc T1 un izmanto to pašu izolācijas līmeni, tai tiek parādīta datu bāzes versija pirms T1 izmaiņām. Tādējādi, T2 redz produktu daudzumu kā 30, nevis 25.
* Kad **T1 apstiprina savas izmaiņas**, T2 joprojām strādā ar sākotnējo datu kopiju, kas tika ielādēta tās darbības sākumā. Jebkuras mēģinājumi piekļūt vai mainīt produktu daudzumu, kas jau tika modificēts T1, rezultēsies ar kļūdu vai bloķēšanu, jo SERIALIZABLE izolācijas līmenis neļauj "dirty" un "non-repeatable" lasījumiem.
* **T2 darbība** tiek pabeigta vai pārtraukta, atkarībā no sistēmas uzstādījumiem un konflikta risināšanas politikas. Ja T2 mēģina veikt savu atjaunināšanu pēc T1 apstiprināšanas, tā saņem kļūdu par nesakritīgu nolasīšanu, kas prasa transakcijas atkārtotu iesākšanu vai izmaiņu noraidīšanu.

Abi scenāriji apliecina, kā izolācijas līmeņi ietekmē datu konsekvenci un tranzakciju izpildes paradumus, garantējot, ka datu bāzes sistēmas atbilst ACID prasībām atkarībā no izvēlētā izolācijas līmeņa.

**4. Jādefinē tabula transakciju izpildei un jāaizpilda tā ar datiem. Var izmantot tabulas no iepriekšējiem darbiem.**

### 1. "Read Committed" izolācijas līmeņa tabulas un dati

CREATE TABLE Inventory (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Quantity INT

);

INSERT INTO Inventory (ProductID, ProductName, Quantity) VALUES(1, 'T-krekls', 100);

INSERT INTO Inventory (ProductID, ProductName, Quantity) VALUES(2, 'Džinsi', 50);

### 2. "Serializable" izolācijas līmeņa tabulas un dati

CREATE TABLE Noliktava (

ProduktaID INT PRIMARY KEY,

ProduktaNosaukums VARCHAR(100),

Daudzums INT

);

INSERT INTO Noliktava VALUES (1, 'Dators', 30);

INSERT INTO Noliktava VALUES (2, 'Printeri', 15);

**4. Jāizprojektē transakciju kopdarbības realizēšanu:**

**1) izveidojot vairākus lietotājus un no katra vienlaicīgi izpildot savu transakciju;**

Pirmkārt tika izveidoti divi lietotāji – natans un natans2.

Tika iedoti visas vajadzīgas piekļuves.

CREATE USER natans IDENTIFIED BY natans;

GRANT CREATE SESSION TO new\_user;

GRANT CREATE TABLE TO c##natans;

GRANT ALTER TABLE TO c##natans;

ALTER SESSION SET CONTAINER = CDB$ROOT;

CREATE USER c##natans IDENTIFIED BY natans;

GRANT CONNECT, RESOURCE TO c##natans;

ALTER USER c##natans QUOTA 100M ON USERS;

GRANT EXECUTE ON DBMS\_LOCK TO c##natans;

/

CREATE USER c##natans2 IDENTIFIED BY natans2;

GRANT CONNECT, RESOURCE TO c##natans2;

GRANT CREATE TABLE TO c##natans2;

GRANT ALTER TABLE TO c##natans2;

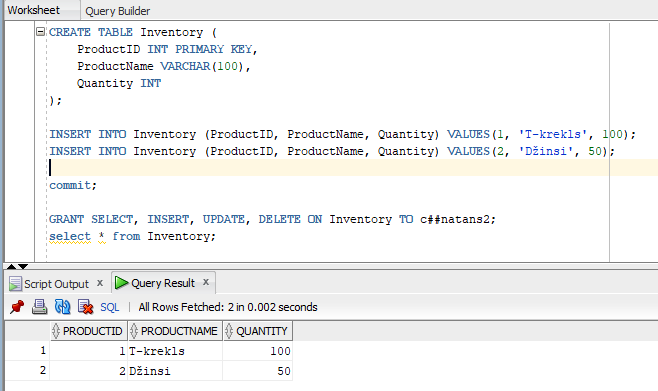
ALTER SESSION SET CONTAINER = CDB$ROOT;

ALTER USER c##natans2 QUOTA 100M ON USERS;

GRANT EXECUTE ON DBMS\_LOCK TO c##natans2;

**1. "Read Committed"**

Tabulu veidošana un aizpildīšana.



Pirmajā transakcijai tika pievienots 5 sekunžu taimeris, lai paspētu palaist transakciju otram lietotājam.

**Transakcija 1**

BEGIN

UPDATE Inventory SET Quantity = Quantity + 50 WHERE ProductID = 1;

DBMS\_LOCK.SLEEP(5);

COMMIT;

END;

/

**Transakcija 2**

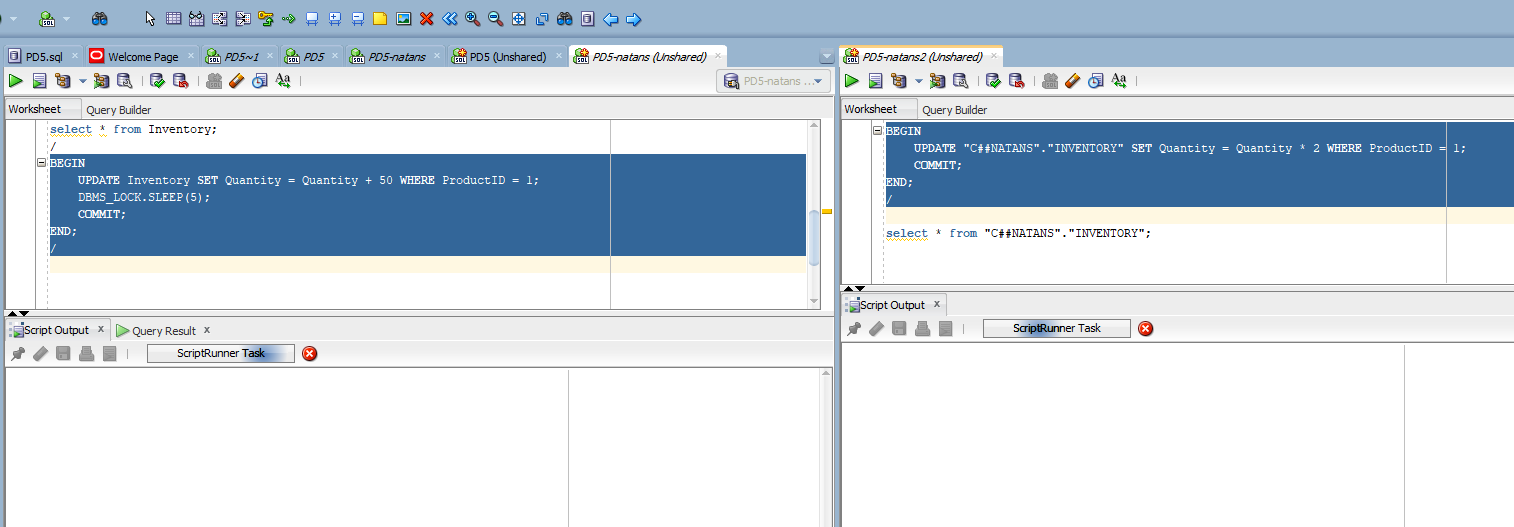
BEGIN

UPDATE "C##NATANS"."INVENTORY" SET Quantity = Quantity \* 2 WHERE ProductID = 1;

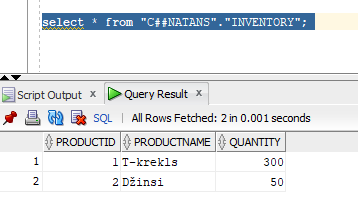
COMMIT;

END;

/



Rezultāts:



**2. "Serializable"**

Tabulu veidošana un aizpildīšana.

CREATE TABLE Noliktava (

ProduktaID INT PRIMARY KEY,

ProduktaNosaukums VARCHAR(100),

Daudzums INT

);

INSERT INTO Noliktava VALUES (1, 'Dators', 30);

INSERT INTO Noliktava VALUES (2, 'Printeri', 15);

GRANT SELECT, INSERT, UPDATE, DELETE ON Noliktava TO c##natans2;

COMMIT;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN

UPDATE Noliktava SET Daudzums = Daudzums - 5 WHERE ProduktaID = 1;

DBMS\_LOCK.SLEEP(10);

COMMIT;

END;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

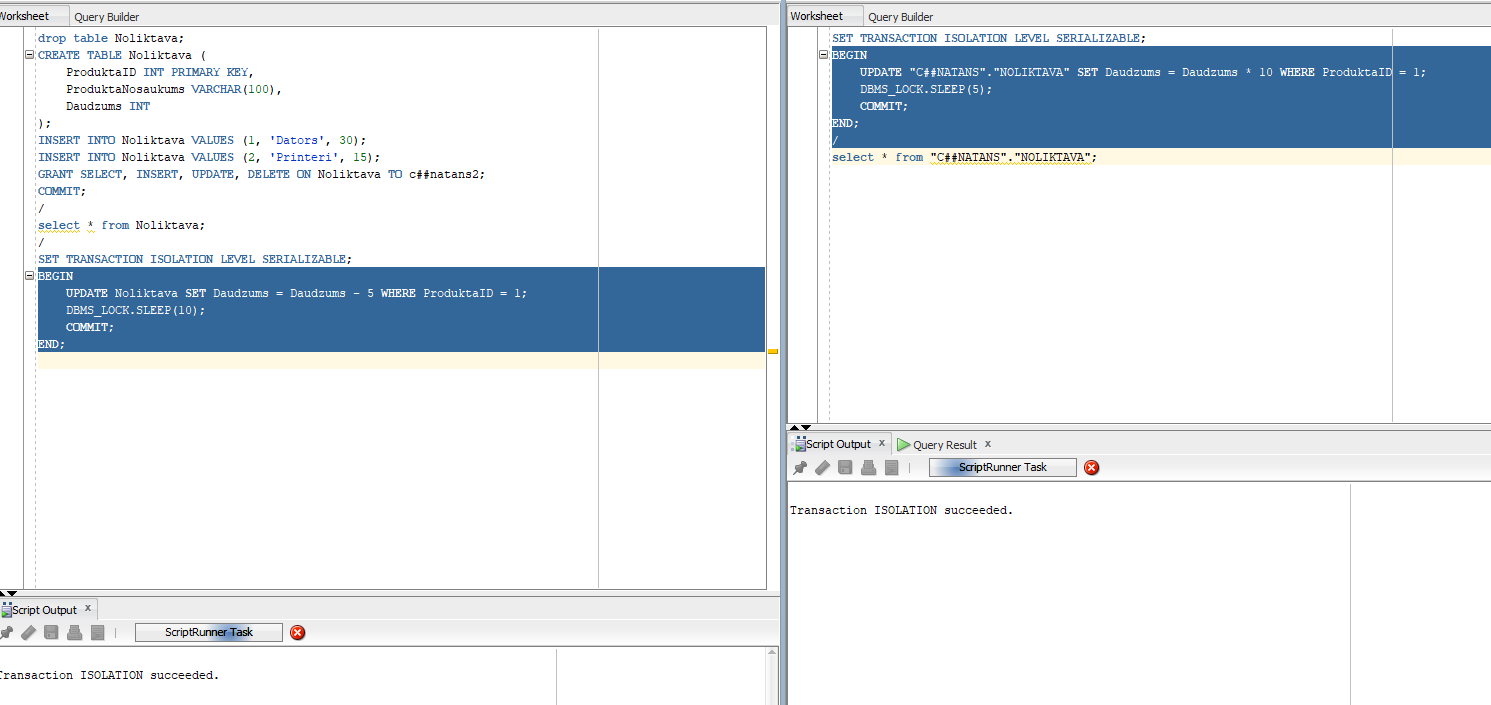
BEGIN

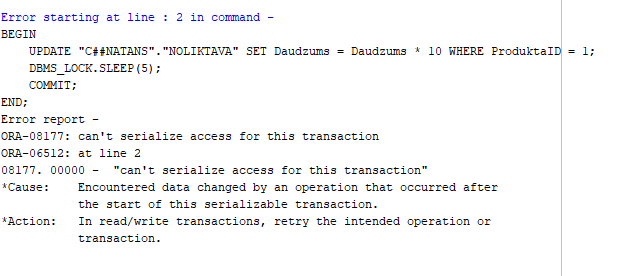
UPDATE "C##NATANS"."NOLIKTAVA" SET Daudzums = Daudzums \* 10 WHERE ProduktaID = 1;

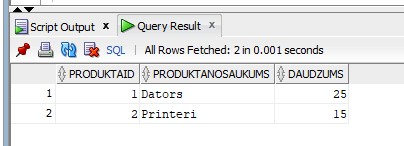
DBMS\_LOCK.SLEEP(5);

COMMIT;

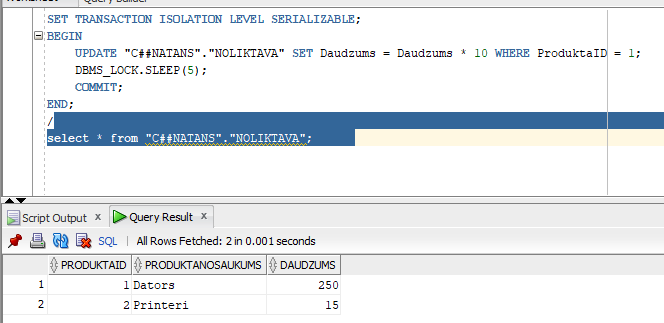
END;



****

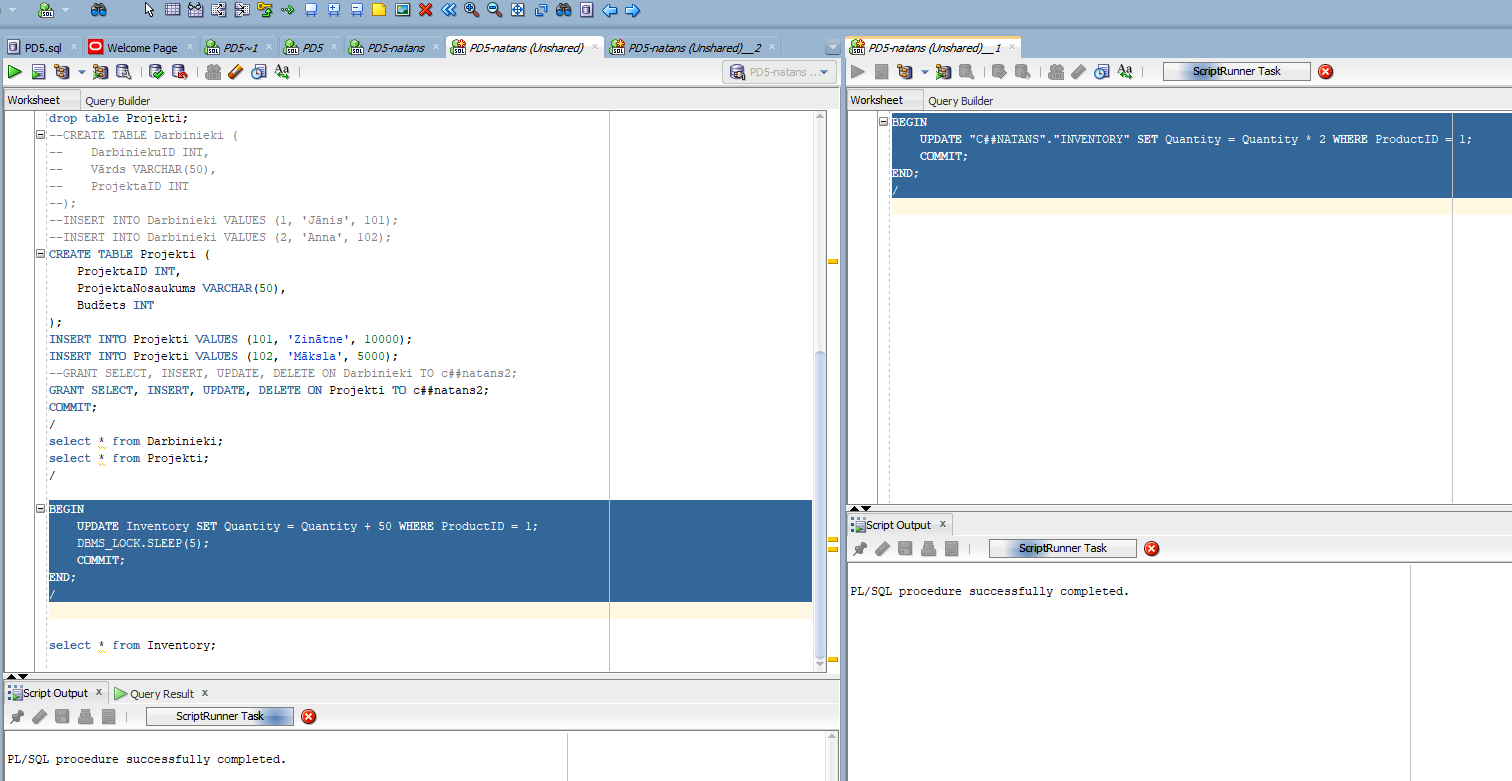
****

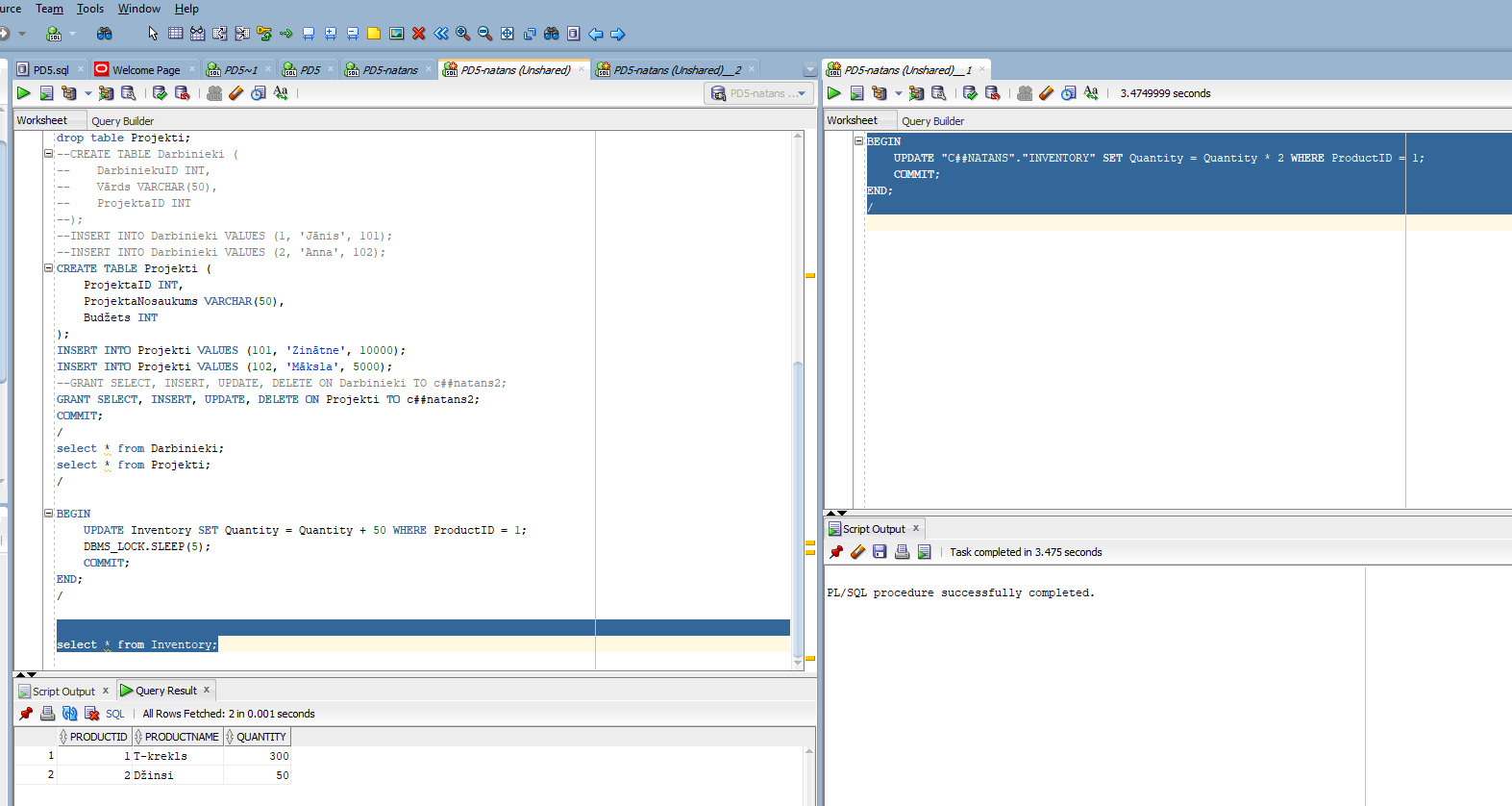
Pēc atkārtotas transakcijas palaišanas dati tiek rediģēti.

****

**2) vienam lietotājam izveidojot vairākus savienojumus un no katra vienlaicīgi izpildot savu transakciju;**

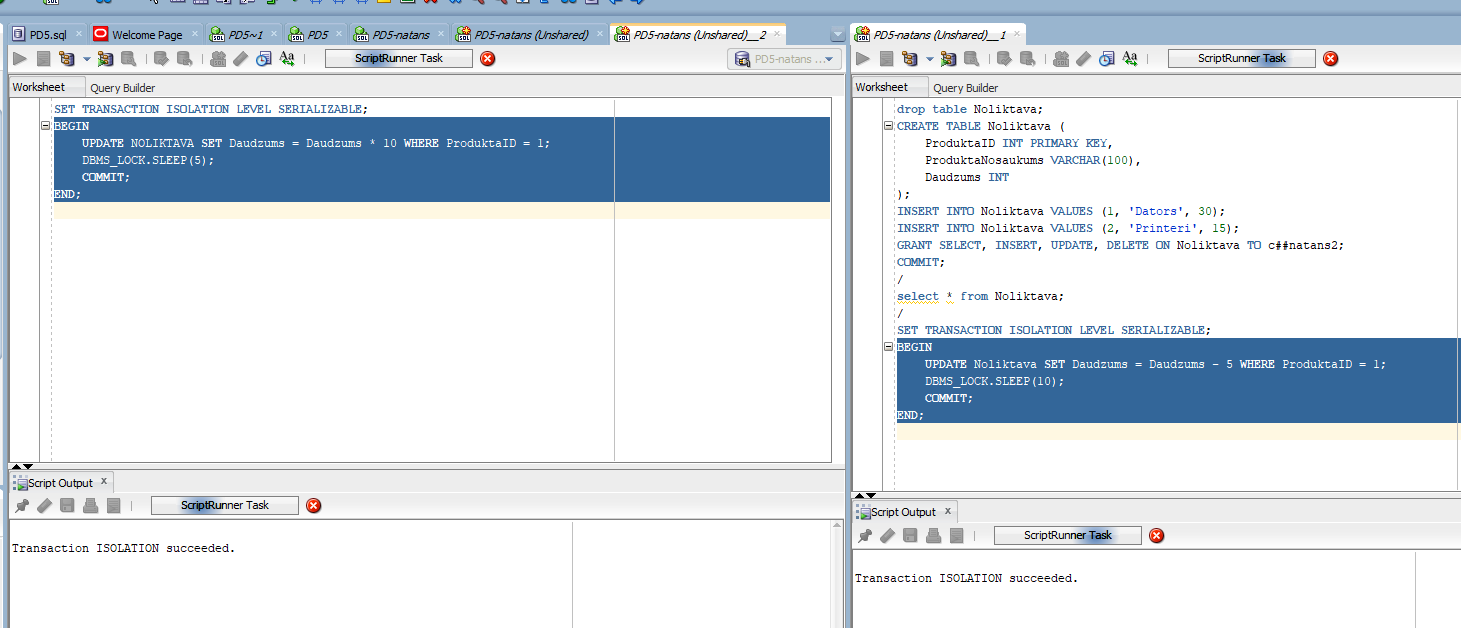
**READ COMMITED**

****

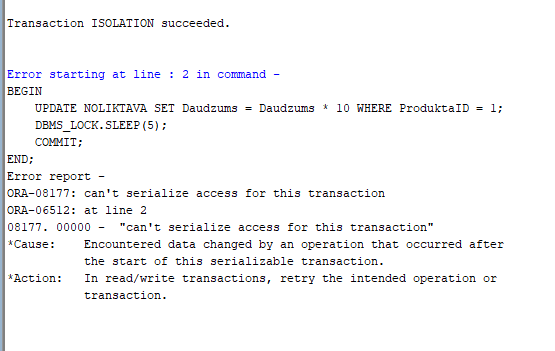
****

Tāds pats rezultāts, kā ar diviem lietotājiem.

**SERIALIZABLE**

****

Tāds pats rezultāts, kā ar diviem lietotājiem:



**3) izmantojot aktīvās datu bāzes trigerus;**

CREATE TABLE InventoryLog (

LogID INT PRIMARY KEY,

ProductID INT,

OldQuantity INT,

NewQuantity INT,

ChangeTimestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE SEQUENCE InventoryLogSeq START WITH 1;

CREATE OR REPLACE TRIGGER InventoryChangeTrigger

AFTER UPDATE ON Inventory

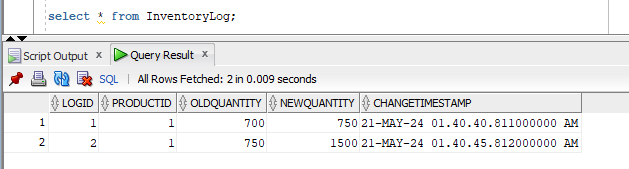
FOR EACH ROW

BEGIN

INSERT INTO InventoryLog (LogID, ProductID, OldQuantity, NewQuantity)

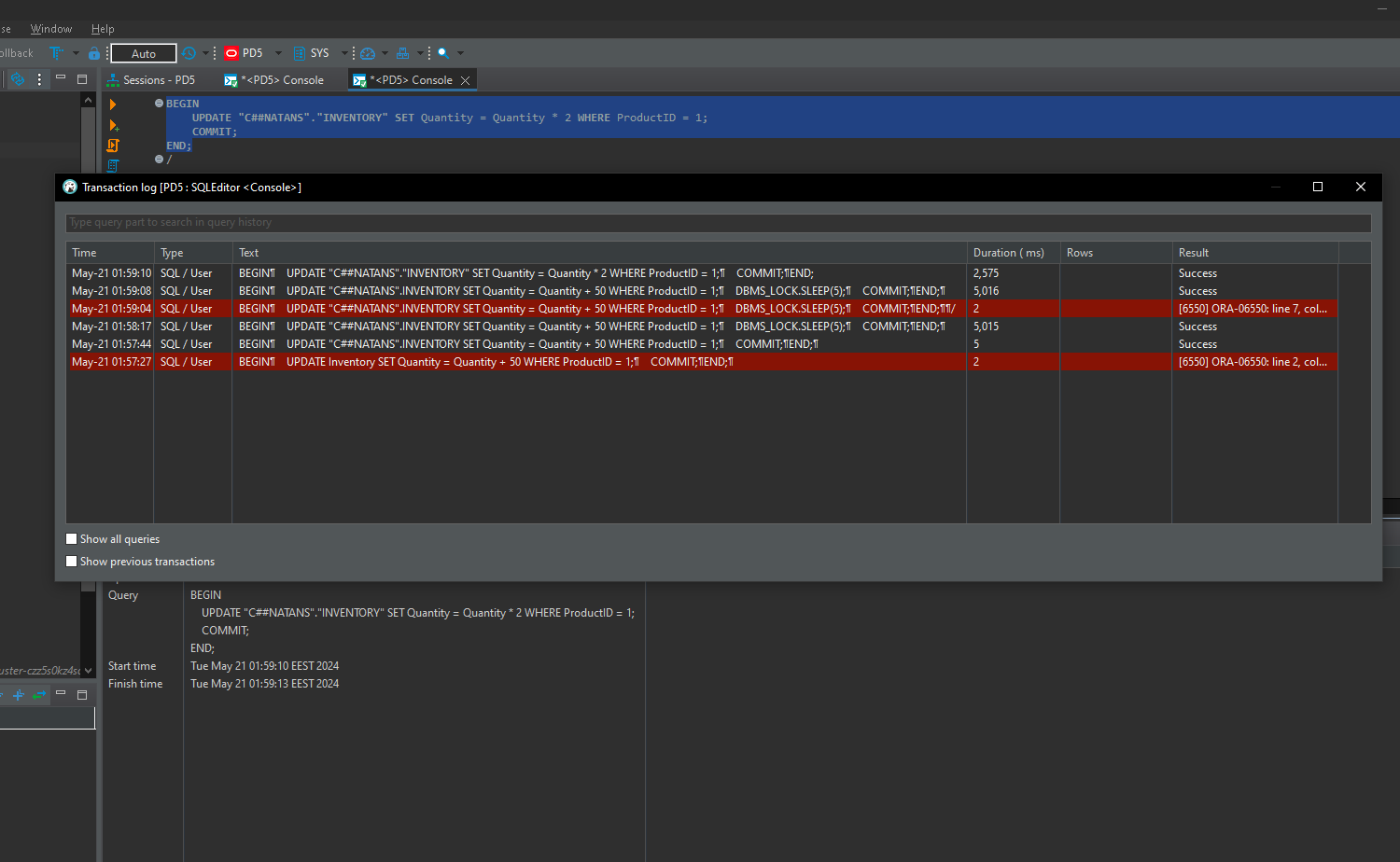
VALUES (InventoryLogSeq.NEXTVAL, :OLD.ProductID, :OLD.Quantity, :NEW.Quantity);

END;

****

**4) izmantojot redaktorus, kuros ērtāk realizēt un apskatīt paralēlās izpildes.**

DBeaver ir universāls datu bāzu pārvaldības rīks, kas atbalsta daudzas datu bāzes, ieskaitot MySQL, PostgreSQL, SQLite, Oracle un citas. Tas piedāvā funkciju "Transaction log", kas parāda transakciju izmaiņas sesijā, kā arī ļauj palaist vaicājumus dažādās sesijās vienlaicīgi.

****