Sprawdzenie jak działa polecenie **SET TRANSACTION READ ONLY;**

* **Czy** możemy używać kwerend typu **insert, update, delete** razem z transakcja read only

|  |  |
| --- | --- |
| polecenia | wyniki |
| 1. COMMIT; | committed. |
| 1. SELECT \* FROM ACCOUNTS; | ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 1  11110003 1  11110004 1  11110005 1  11110006 1 |
| 1. UPDATE ACCOUNTS SET ACCOUNT\_BALANCE = 2; | 5 rows updated. |
| 1. SELECT \* FROM ACCOUNTS; | ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 2  11110003 2  11110004 2  11110005 2  11110006 2 |
| 1. COMMIT; 2. SET TRANSACTION READ ONLY; | committed.  transaction READ succeeded. |
| 1. SELECT \* FROM ACCOUNTS; | ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 2  11110003 2  11110004 2  11110005 2  11110006 2 |
| 1. UPDATE ACCOUNTS   SET ACCOUNT\_BALANCE = 3;   1. SELECT \* FROM ACCOUNTS; | Error starting at line : 8 in command -UPDATE ACCOUNTS  SET ACCOUNT\_BALANCE = 3 Error report -SQL Error: ORA-01456: nie można przeprowadzać operacji insert/delete/update w ramach transakcji READ ONLY 01456. 00000 - "may not perform insert/delete/update operation inside a READ ONLY transaction"  \*Cause: A non-DDL insert/delete/update or select for update operation was attempted\*Action: commit (or rollback) transaction, and re-execute  ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 2  11110003 2  11110004 2  11110005 2  11110006 2 |

**Wniosek:** kwerenda” SET TRANSACTION READ ONLY” uniemożliwia na zmianę rekordów w bazie danych mamy możliwość jedynie wyświetlać. Na początku transakcja była ustawiona na możliwość pisania i czytania co pozwoliło nam wyświetlić dane przed i po aktualizacji wiersze 2,3,4. To samo próbowałem zrobić po zmianie transakcji na **READ ONLY** wiersz 6 i powtórzyć kwerendy 7,8,9. Przy próbie aktualizacji wyskoczył błąd i dane się nie zaktualizowały. Podsumowując **READ ONLY** działa jak izolacja **serializable**, więc anomalie będą jak w tej izolacji.

Sprawdzenie jak działa domyślna izolacja **READ\_COMMITED;**

* czy mamy doczynienia z anomalia niezatwierdzony odczyt
* czy bieżące kwerendy widza tylko zmiany przed zatwierdzeniem

|  |  |  |  |
| --- | --- | --- | --- |
| Transakcja 1 | Transakcja 2 | Wynik transakcji 1 | Wynik transakcji 2 |
| 1. COMMIT 2. select \* from ACCOUNTS; |  | Committed  ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 2  11110003 2  11110004 2  11110005 2  11110006 2 |  |
| 1. UPDATE ACCOUNTS 2. SET ACCOUNT\_BALANCE =5 |  | 5 rows updated. |  |
|  | 1. COMMIT; 2. select \* from ACCOUNTS; |  | Committed  ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 2  11110003 2  11110004 2  11110005 2  11110006 2 |
| 1. select \* from ACCOUNTS; |  | ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 5  11110003 5  11110004 5  11110005 5  11110006 5 |  |
| 1. ROLLBACK 2. select \* from ACCOUNTS; |  | rollback complete.  ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 2  11110003 2  11110004 2  11110005 2  11110006 2 |  |
| 1. COMMIT 2. select \* from ACCOUNTS; 3. UPDATE ACCOUNTS 4. SET ACCOUNT\_BALANCE =5 |  | Committed  ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 2  11110003 2  11110004 2  11110005 2  11110006 2  5 rows updated. |  |
|  | 1. COMMIT 2. select \* from ACCOUNTS; |  | committed.  ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 2  11110003 2  11110004 2  11110005 2  11110006 2 |
| 1. select \* from ACCOUNTS; |  | ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 5  11110003 5  11110004 5  11110005 5  11110006 5  -------------- --------------- |  |
|  | 1. select \* from ACCOUNTS; |  | ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 5  11110003 5  11110004 5  11110005 5  11110006 5 |
| Transakcja 3 |  | Committed  ACCOUNT\_NUMBER ACCOUNT\_BALANCE  11110002 5  11110003 5  11110004 5  11110005 5  11110006 5 |  |
| 1. COMMIT; 2. select \* from ACCOUNTS; |

**Wniosek:** Rozpoczynając pierwszą transakcje i aktualizując dane widać różnice między wierszem 2 i 7 dane zostały zaktualizowane natomiast w drugiej transakcji, próbując sprawdzić dane po aktualizacji transakcji pierwszej ( wiersz 3,4) w wierszu 6 transakcji drugiej nie ma zmienionych danych. Kolejne polecenia transakcji pierwszej wiersz 8,9 przywracają transakcje do stanu początkowego.

Robiąc tą samą czynność tylko tym razem rozpocząć nową transakcje po aktualizacji danych co sprawi zatwierdzenie zmian widzimy ze wywołując ciągle tą samą czynność w transakcji 2 wiersz 15,19 zauważyć warto że zwraca nam inne wartości.

Mamy brak zjawiska brudnego odczytu ale niestety jest anomalia niepowtarzalnego odczytu.

**READ\_COMMITED –**

**anomalia fantomowa**

* sprawdzenie występowania anomalii odczytu

|  |  |  |  |
| --- | --- | --- | --- |
| Transakcja 1 | Transakcja 2 | Wynik transakcji 1 | Wynik transakcji 2 |
| 1. COMMIT; 2. select count(\*) FROM ACCOUNTS; | 1. COMMIT; 2. select count(\*) FROM ACCOUNTS; 3. INSERT INTO ACCOUNTS (ACCOUNT\_NUMBER, ACCOUNT\_BALANCE) VALUES 4. (SEQ\_ACCOUNT\_NUMBER.NEXTVAL, 0); 5. Select count(\*) FROM ACCOUNTS; 6. COMMIT | committed.  COUNT(\*)  ----------  5  committed. | committed.  COUNT(\*)  ----------  5  1 rows inserted.  COUNT(\*)  ----------  6  committed. |
| 1. COMMIT; 2. select count(\*) FROM ACCOUNTS; |  | COUNT(\*)  ----------  6 |  |

**Wniosek:** Dla pierwszej transakcji zmieniła się wielkość zbioru przez wykonanie transakcji 2, która dodała kolejny element powstał efekt fantomowy. Niestety izolacja domyślna read\_commited nie hamuje występowania tej anomalii .

**READ\_COMMITED –**

**anomalia utraconych aktualizacji**

|  |  |  |  |
| --- | --- | --- | --- |
| Transakcja 1 | Transakcja 2 | Wynik transakcji 1 | 1. Wynik transakcji 2 |
| 1. COMMIT; 2. SELECT \* FROM ACCOUNTS; |  | committed.  ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 5  11110003 5  11110004 5  11110005 5  11110006 5  11110007 5  11110008 5 |  |
| 1. UPDATE ACCOUNTS SET ACCOUNT\_BALANCE=3; 2. SELECT \* FROM ACCOUNTS; | 1. COMMIT; 2. SELECT \* FROM ACCOUNTS; 3. UPDATE ACCOUNTS SET ACCOUNT\_BALANCE=5; | 7 rows selected  ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 3  11110003 3  11110004 3  11110005 3  11110006 3  11110007 3  11110008 3 | Committed  Nic się nie wyswietla ładowanie  Czeka |
| 1. COMMIT; 2. SELECT \* FROM ACCOUNTS; | 1. SELECT \* FROM ACCOUNTS; 2. COMMIT; 3. SELECT \* FROM ACCOUNTS; | Committed  ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 3  11110003 3  11110004 3  11110005 3  11110006 3  11110007 3  11110008 3 | 7 rows selected  ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 5  11110003 5  11110004 5  11110005 5  11110006 5  11110007 5  11110008 5  Committed  ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 5  11110003 5  11110004 5  11110005 5  11110006 5  11110007 5  11110008 5 |

**Wniosek:** Dane z pierwszej transakcji zostały nadpisane ale co zauważyłem to, że w drugiej transakcji nastąpiło ładownie „ScriptRunner Task” , które zakończyło się po wywołaniu commit() w pierwszej transakcji czyli ewidentnie pod wpływem pierwszej transakcji kwerenda update nie mogła się wykonać po tym wszystkim na koniec dane zostały nadpisane.

**READ\_COMMITED –**

**anomalia utraconych aktualizacji i for update**

|  |  |  |  |
| --- | --- | --- | --- |
| Transakcja 1 | Transakcja 2 | Wynik transakcji 1 | Wynik transakcji 2 |
| 1. `COMMIT 2. SELECT \* FROM ACCOUNTS; 3. FOR UPDATE | 1. COMMIT | Committed  ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 5  11110003 5  11110004 5  11110005 5  11110006 5  11110007 5  11110008 5 | Committed |
|  | 1. SELECT \* FROM ACCOUNTS |  | Nic się nie wyswietla ładowanie |
| 1. UPDATE ACCOUNTS SET ACCOUNT\_BALANCE=3; 2. SELECT \* FROM ACCOUNTS |  | 7 rows updated.  ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 3  11110003 3  11110004 3  11110005 3  11110006 3  11110007 3  11110008 3 |  |
| 1. commit |  | Committed | Dopiero po commit w pierwszej trasakcji wyświetliła mi się zawartość  ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 3  11110003 3  11110004 3  11110005 3  11110006 3  11110007 3  11110008 3 |
|  | 1. UPDATE ACCOUNTS SET ACCOUNT\_BALANCE=0 2. commit; 3. SELECT \* FROM ACCOUNTS |  | 7 rows updated.  Committed  ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 0  11110003 0  11110004 0  11110005 0  11110006 0  11110007 0  11110008 0 |

**Wniosek:** Dane z pierwszej transakcji zostały nadpisane ale co zauważyłem to, że w drugiej transakcji nastąpiło ładownie „ScriptRunner Task”, uniemożliwiając wypisanie danych. Zakończyło się po wywołaniu commit() wiersz 6 w pierwszej transakcji i dopiero wtedy można było działać na drugiej transakcji aktualizując je na sam koniec w wierszu 7 i sprawdzając wypisywanie danych. Podsumowując klauzula for update przy poleceniu selekt oczekuje aż zakończy się wprowadzanie danych przez pierwsza rozpoczętą transakcje. Nie mamy zablokowanych możliwości edycji rekordów.

Sprawdzenie jak działa najwyższa izolacja z anomalii Dirty read oraz Non-reapeatable read

**Serializable**

|  |  |  |  |
| --- | --- | --- | --- |
| Transakcja 1 | Transakcja 2 | Wynik transakcji 1 | Wynik transakcji 2 |
| 1. `COMMIT 2. SELECT \* FROM ACCOUNTS |  | Committed  ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 0  11110003 0  11110004 0  11110005 0  11110006 0  11110007 0  11110008 0 |  |
| 1. UPDATE ACCOUNTS SET ACCOUNT\_BALANCE=1 | 1. commit | 7 rows updated. | Committed |
| 1. SELECT \* FROM ACCOUNTS | 1. SELECT \* FROM ACCOUNTS | ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 1  11110003 1  11110004 1  11110005 1  11110006 1  11110007 1  11110008 1 | ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 0  11110003 0  11110004 0  11110005 0  11110006 0  11110007 0  11110008 0 |
| 1. commit |  | Committed |  |
|  | 1. SELECT \* FROM ACCOUNTS |  | ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 0  11110003 0  11110004 0  11110005 0  11110006 0  11110007 0  11110008 0 |
|  | 1. commit |  | Committed |
|  | 1. SELECT \* FROM ACCOUNTS |  | ACCOUNT\_NUMBER ACCOUNT\_BALANCE  -------------- ---------------  11110002 1  11110003 1  11110004 1  11110005 1  11110006 1  11110007 1  11110008 1 |

**Wniosek:** Izolacja serializable jest bezkompromisowa ze wszystkich dostępnych izolacji. Pozwala na zapobieganie anomaliom. W powyższym przykładzie chciałem się upewnić czy jest możliwość wystąpienia anomalii Dirty read oraz Non-reapeatable read. Można zauważyć że mimo zakończenia transakcji jeden wiersz 5 nadal nie widać w równoległej transakcji 2 jej poczynań wiersz 4, 6 jest taki sam dopiero po zakończeniu drugiej transakcji i sprawdzeniu ponownie danych wiersz 8 widać że transakcja 1 zaktualizowała dane.

**Serializable– anomalia fantomowa**

* sprawdzenie występowania anomalii odczytu

|  |  |  |  |
| --- | --- | --- | --- |
| Transakcja 1 | Transakcja 2 | Wynik transakcji 1 | Wynik transakcji 2 |
| 1. COMMIT; 2. select count(\*) FROM ACCOUNTS; | 1. commit | committed.  COUNT(\*)  ----------  7 | committed |
|  | 1. INSERT INTO ACCOUNTS (ACCOUNT\_NUMBER, ACCOUNT\_BALANCE) VALUES (SEQ\_ACCOUNT\_NUMBER.NEXTVAL, 0); |  | 1 rows inserted. |
| 1. UPDATE ACCOUNTS SET ACCOUNT\_BALANCE=1 |  | Występuje problem z aktualizacja danych do mometu zakończenia transakcji 2. |  |
| 1. COMMIT; 2. select count(\*) FROM ACCOUNTS; | 1. commit |  | Committed  COUNT(\*)  ----------  8 |

**Wniosek:** Pod wpływem działania drugiej równoległej transakcji i zmianie przez nią zbioru danych nie możliwe było do wykonania update zbioru wiersz 4. Izolacja serializable poradziło sobie z anomaliom fantomu. Transakcja 2 blokowała zmiany w transakcji 1 przyczyniło się to do zachowania spójności danych ale konsekwencja było opóźnienie. Na sam koniec wierszu 6 udłao się zaobserwować dodatkowy rekord gdyż count wyniósł 8.