Bazy Danych

3. Procedury i funkcje

Opracował: Maciej Penar

Spis treści

[1. (Mniej niż) garść informacji 3](#_Toc35430278)

[2. Zadanie **Błąd! Nie zdefiniowano zakładki.**](#_Toc35430279)

[Info **Błąd! Nie zdefiniowano zakładki.**](#_Toc35430280)

[Info dot. baz danych **Błąd! Nie zdefiniowano zakładki.**](#_Toc35430281)

# 1. Garść informacji

Procedury składowane (ang. Stored Procedures) i funkcje (ang. Functions) stanowią obiekty BD które umożliwiają jej programowanie. Rozszerzają one umiejętności SQL-a: mamy dostępny nie tylko deklaratywny model języka, ale także instrukcje znane z proceduralnych języków (IF, WHILE, itp.).

*Drzewiej istniała bardzo szlachetna koncepcja tzw. architektury 2-warstwowej: Frontu oraz Bazy Danych (wraz z backendem) – koncepcja ta nigdy nie mogła się udać. Zainteresowanych odsyłam do technologii Oracle Apex.*

Różnica pomiędzy procedurami, a funkcjami jest taka, że Funkcje mogą zwrócić co najwyżej 1 wartość – procedury nie mają ograniczeń. Bardzo często funkcje i procedury składowane wrzucamy do jednego worka zbiorczo określanego „procedury”.

Procedury cieszą się dużą popularnością wśród programistów, ze względu na możliwość cache-owania planów zapytania. Stąd dość często wymagana jest podstawowa znajomość ich składni – i tu się zaczynają schody, bo każdy dostawca wymyśla sobie swoją własną składnię. **A to rodzi poważne problemy w kontekście migracji BD – w praktyce wybór dostawcy BD nie pozwala na prostą migrację z powodu procedur**. Ergo: jeśli procedur w BD nie ma, to BD powinna dać się łatwo zmigrować.

Żeby ten problem unaocznić. Załóżmy funkcje dodającą dwie liczby, coś co w zwykłym języku napisalibyśmy jak;

|  |
| --- |
| fun addMe(x : Int, y : Int) = x + y |

W proceduralnym SQL – w MSSQL Serverze napisalibyśmy:

|  |
| --- |
| CREATE FUNCTION AddMe (@x INT, @y INT) RETURNS INT  BEGIN  RETURN @x + @y  END |

Analogiczna funkcja w DB2:

|  |
| --- |
| CREATE FUNCTION AddMe (x INT, y INT) RETURNS INT  DETERMINISTIC  NO EXTERNAL ACTION  CONTAINS SQL  BEGIN  RETURN x + y;  END# |

Analogiczna funkcja w Oracle (nie jestem pewny wersji Oraclowej):

|  |
| --- |
| CREATE FUNCTION AddMe (x IN NUMBER, y IN NUMBER) RETURN NUMBER IS  BEGIN  RETURN (x + y);  END |

Ze względu, że mam pod ręką BD MSSQL Server, to opiszę tu składnie z narzecza tej BD.

## Nowy paradygmat, nowe problemy

Pomimo, że procedury składowane istnieją od dawien-dawna w BD, to obiekty te dalej są obsługiwane problematycznie. Na przykład :

* łatwiej jest zauważyć zjawisko rekursji (w DQL rekursja też jest możliwa)
* **! optymalizatory zapytań nie potrafią szacować kosztu funkcji i często przyjmują jakąś stałą**
* **!! optymalizatory zapytań nie radzą sobie z instrukcjami warunkowymi i optymalizują tylko jedną gałąź (Tzw. parameter sniffing)**

# Procedury

W ramach proceduralnego SQL-a można (składnia SQL Servera):

|  |  |  |
| --- | --- | --- |
| **Co** | **Jak** | **Komentarz** |
| Deklarować zmienną | DECLARE @x INT; | Dowolna nazwa zmiennej, dowolny typ zgodny z ANSI SQL |
| Ustawić wartość zmiennej | SET @x = 1; |  |
| Blok SQL-a | BEGIN  ……  END | Blok jest traktowany jako jeden element |
| Instrukcja warunkowa IF | IF @x = 1 [SQL lub blok]  ELSE [SQL lub blok]; | Dowolny warunek – wygląda podobnie jak przypisanie |
| Instrukcja WHILE | WHILE @x=1 [SQL lub blok] | Dowolny warunek – wygląda podobnie jak przypisanie |
| Deklaracja kursora | DECLARE my\_cursor CURSOR FOR [SQL] | Deklaracja dla kursora |
| Czytanie kursora | OPEN my\_cursor  FETCH NEXT FROM my\_cursor INTO @x    WHILE @@FETCH\_STATUS = 0  BEGIN  [SQL using @x]  FETCH NEXT FROM my\_cursor INTO @x  END    CLOSE my\_cursor  DEALLOCATE my\_cursor | |

Więcej informacji tu: [link](https://docs.microsoft.com/en-us/sql/t-sql/language-elements/control-of-flow?view=sql-server-ver15)

## Uruchamianie procedur i funkcji

Zdefiniowaną wcześniej funkcję:

|  |
| --- |
| CREATE FUNCTION AddMe (@x INT, @y INT) RETURNS INT  BEGIN  RETURN @x + @y  END |

W SQL Server możemy wykorzystać w dowolnym zapytaniu SELECT:

|  |
| --- |
| SELECT dbo.AddMe(11,5); |
| SELECT dbo.AddMe(t.x,t.y) FROM Test t; |

Ze względu na ogólniejszy charakter procedur, sposób ich wywołania różni się i zazwyczaj wykorzystywane jest słowo kluczowe EXEC/EXECUTE.

|  |
| --- |
| EXECUTE SomeProcedure @input1 = 1, @input2 = 2; |

# 3. Trochę o transakcjach

## Słowo wstępu / ACID

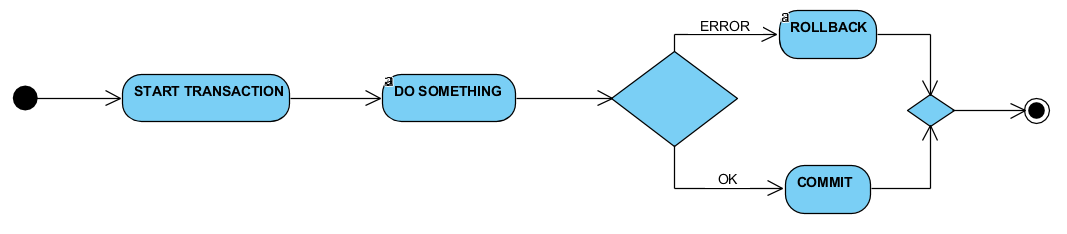
Pojęcie transakcji wywodzi się z Systemów Operacyjnych. Transakcja stanowi jednostkę przetwarzania w ramach jakiegoś systemu (my rozważamy Bazy Danych). Operacja transakcji jest scharakteryzowana czterema cechami:

1. **Atomicity** - Atomowością – czyli operacja wykonuje się w całości albo w ogóle
2. **Consistency** – Spójnością – czyli operacja przeprowadza system ze stanu spójnego w inny spójny stan. W przypadku baz danych spójność jest rozumiana dwojako:
   1. Spójność jako zachowanie ograniczeń (więzy CHECK / ograniczenia klucza obcego)
   2. \* Spójność jako identyczny stan replik (dla rozproszonych baz danych)
3. **Isolation** - Izolacja – jeśli system przetwarza operacje współbieżnie (a przetwarza), to operacje przeciwdziałają negatywnym skutkom konkurencji
4. **Durability** – trwałość – czyli wynik transakcji ulega trwałemu zatwierdzeniu odpowiada za to na ogół tzw. „dziennik transakcji’. Trwałość można rozumieć jako „dziennikowanie” systemu.

Wszystkie cztery cechy w skrócie możemy zapisać jako **ACID** od pierwszych liter **A**tomicity**, C**onsistency**, I**solation**, D**urability**.**

## Schemat transakcji

Ogólny schemat transakcji jest następujący:



Zasadniczo transakcja posiada trzy etapy:

1. Początek
2. Ciało – czyli właściwe wyrażenia SQL
3. Zatwierdzenie / Odrzucenie

## Jawne Transakcje - SQL

Chyba standard ANSI/SQL nie precyzuje w jaki sposób w SQL’u wyrażać jawne transakcje. Z tego względu w każdej bazie danych składania się różni. Poniżej znajduje się składnia zwyczajowa (DB2 oraz SQL Server się stosują) oraz składnia w Oracle.

|  |  |  |
| --- | --- | --- |
| **Wyrażenie** | **Zwyczajowo** | **Oracle DB** |
| Początek transakcji | BEGIN TRANSACTION; | SET TRANSACTION NAME [nazwa]; |
| Zatwierdzenie transakcji | COMMIT; | COMMIT; |
| Odrzucenie transakcji | ROLLBACK; | ROLLBACK; |

Przykłady jawnych transakcji:

|  |  |
| --- | --- |
| Prosty SELECT | **SET TRANSACTION NAME ‘1';**  SELECT \* FROM TR;  **COMMIT**; |
| Prosty SELECT… wyraża to samo co poprzedni | **SET TRANSACTION NAME '2';**  SELECT \* FROM TR;  **ROLLBACK**; |
| Prosty INSERT – zatwierdzony | **SET TRANSACTION NAME '3';**  INSERT INTO TR VALUES(1000);  **COMMIT**; |
| Prosty INSERT – odrzucony | **SET TRANSACTION NAME '3';**  INSERT INTO TR VALUES(-1000);  **ROLLBACK**; |

Spostrzeżenie: słowa kluczowe **ROLLBACK** i **COMMIT** służą do realizacji właściwości **Atomowości.**

## Niejawne Transakcje - SQL

Twist fabularny. Gdy do bazy danych wpłynie:

1. SELECT \* FROM TR WHERE ID = 1;
2. INSERT INTO TR VALUES(1337);

To baza danych i tak opakuje to w wyrażenia:

|  |  |
| --- | --- |
| a) | **SET TRANSACTION NAME ‘dasdasdadasdsad'; // BD doda niejawnie**  SELECT \* FROM TR WHERE ID = 1;  **COMMIT**; **// BD doda niejawnie** |
| b) | **SET TRANSACTION NAME ‘asddsaasdaas’; // BD doda niejawnie**  INSERT INTO TR VALUES(1337);  **COMMIT**; **// BD doda niejawnie** |

## Poziomy izolacji

Poziomy izolacji służą do ustalenia w jaki sposób wpływają na siebie transakcje które:

1. Są wykonywane współbieżnie
2. Są odrzucone bądź zatwierdzone

Standard ANSI/ISO SQL przewiduje cztery poziomy izolacji w ramach technik zwanych „pesymistycznym” sterowaniem współbieżnością. Na ogół Bazy Danych sterują pesymistycznie. Zakładamy że transakcje działające współbieżnie mogą nadpisać swoje wyniki wzajemnie. **Pesymizm** technik związany jest z tym, że nie dopuszczają one do najgorszego możliwego scenariusza – nadpisania danych jednej transakcji przez drugą.

Techniki pesymistyczne:

1. Oparte są o blokady
2. Gwarantują uporządkowanie (uszeregowanie) transakcji w taki sposób że transakcje nie czytają niezatwierdzonych wyników innej transakcji

|  |
| --- |
| *Ciekawostka* |
| Istnieją też techniki **optymistyczne** – na ogół są bardziej przepustowe (pod względem transakcji-na-minutę), ale wymagają więcej pamięci operacyjnej oraz godzimy się na utratę danych. W technikach optymistycznych wygrywa ostatni piszący. |

No dobra, co może pójść nie tak podczas przetwarzania transakcji? Mamy trzy anomalie:

1. **Dirty Read** - Brudny odczyt – kiedy transakcja odczytuje niezatwierdzone dane
2. **Non-Repetable Read** - Niepowtarzalny odczyt – transakcja **dwukrotnie** odczytuje zbiór instancji. Instancje krotek ulegają zmianie (na wskutek wyrażeń UPDATE) przy drugim odczycie.
3. **Phantoms** - Fantomowe krotki – transakcja **dwukrotnie** odczytuje zbiór instancji. Zbiór krotek zawiera krotki których nie było poprzednio (na wskutek INSERT).

Mamy cztery poziomy izolacji do walki z anomaliami. Najniższy poziom – „odczyt niezatwierdzony” dopuszcza wszystkie anomalie. Każdy wyższy poziom izolacji odejmuje 1 anomalię. Poziomy te to:

1. Read uncommited – odczyt niezatwierdzony
2. Read commited – odczyt zatwierdzony
3. Repetable Read – odczyt powtarzalny
4. Serializable – szeregowalny

Uwaga: poziom „odczyt powtarzalny” **nie** oznacza że za każdym razem jak powtórzymy odczyt to otrzymamy ten sam zbiór krotek. [Link](https://www.youtube.com/watch?v=Y7-FQYMiKeQ) (0:03-0:08)

Tabela poziomów izolacji i możliwych anomalii wygląda tak:

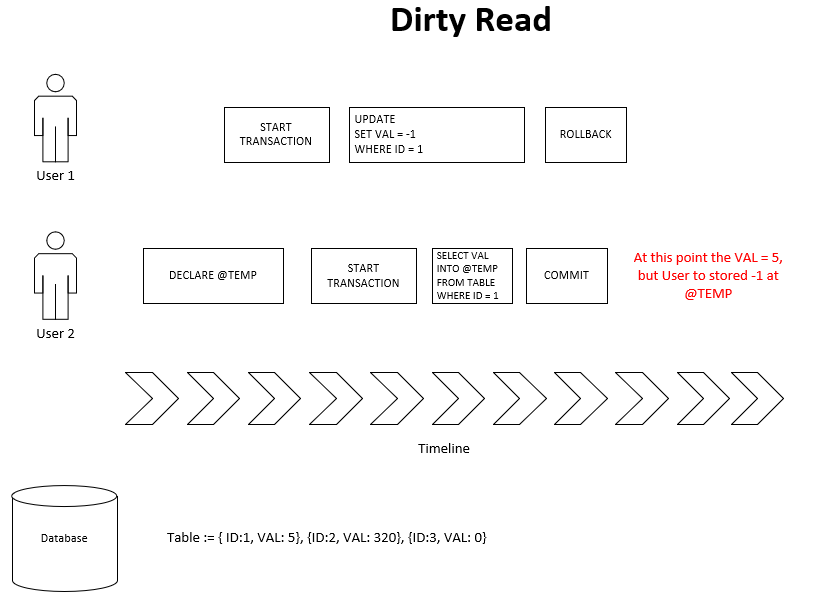
|  |  |  |  |
| --- | --- | --- | --- |
| Poziom Izolacji \Anomalia | Dirty Read | Non-Repetable Read | Phantoms |
| Read Uncommited | występuje | występuje | występuje |
| Read Commited |  | występuje | występuje |
| Repetable Read |  |  | występuje |
| Serializable |  |  |  |

Spostrzeżenie:

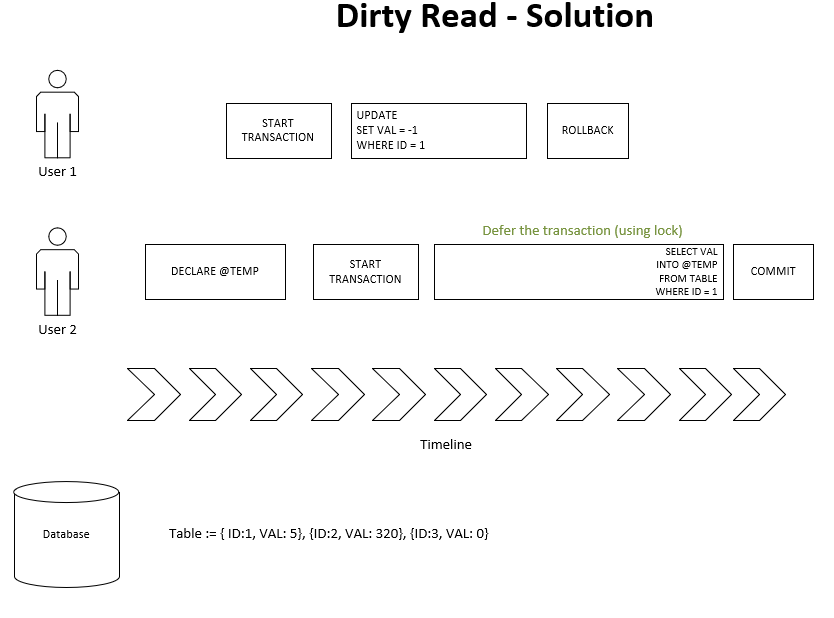
Poziom izolacji **READ COMMITED** jest **optymalny** dopóki Wasze transakcje składają się z **pojedynczych** wyrażeń SELECT / INSERT/ UPDATE / DELETE. **Nie** zaobserwujecie anomalii niepowtarzalnego odczytu, a wpływ fantomowych krotek **zazwyczaj** jest znikomy.

## anomalia – Brudny odczyt

Na poziomie izolacji **brudny odczyt** może wystąpić następująca sytuacja:

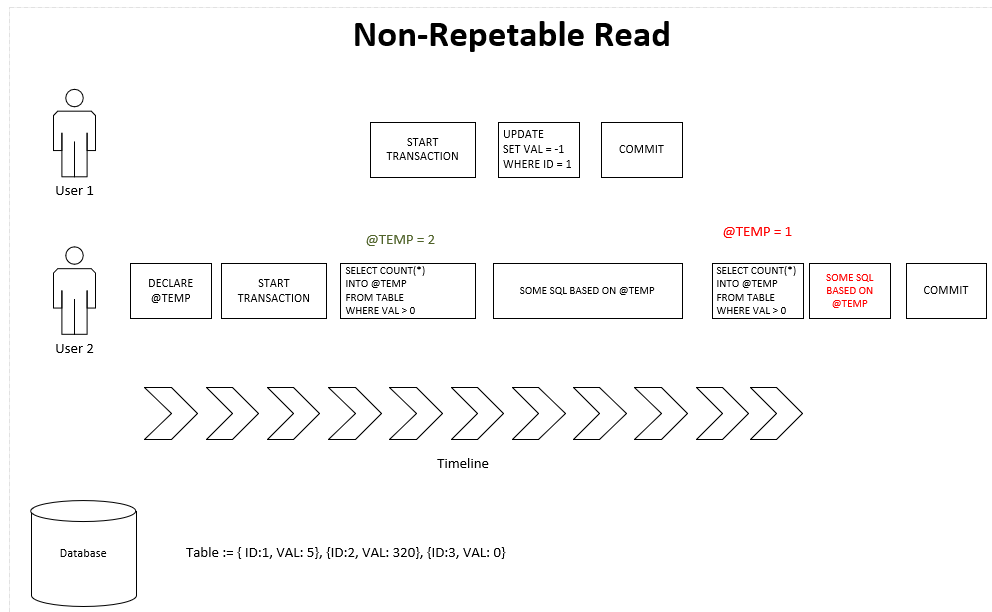


Na poziomie izolacji **zatwierdzonego odczytu** możemy naprawić tę sytuację:

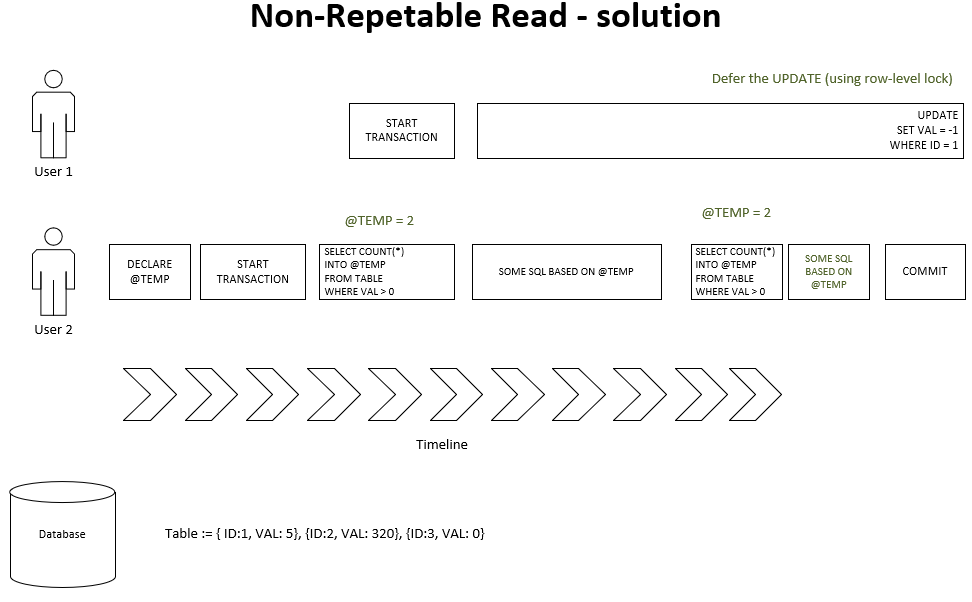


## anomalia – niepowtarzalny odczyt

Na poziomie **zatwierdzonego odczytu** może pojawić się problem niepowtarzalnego odczytu:

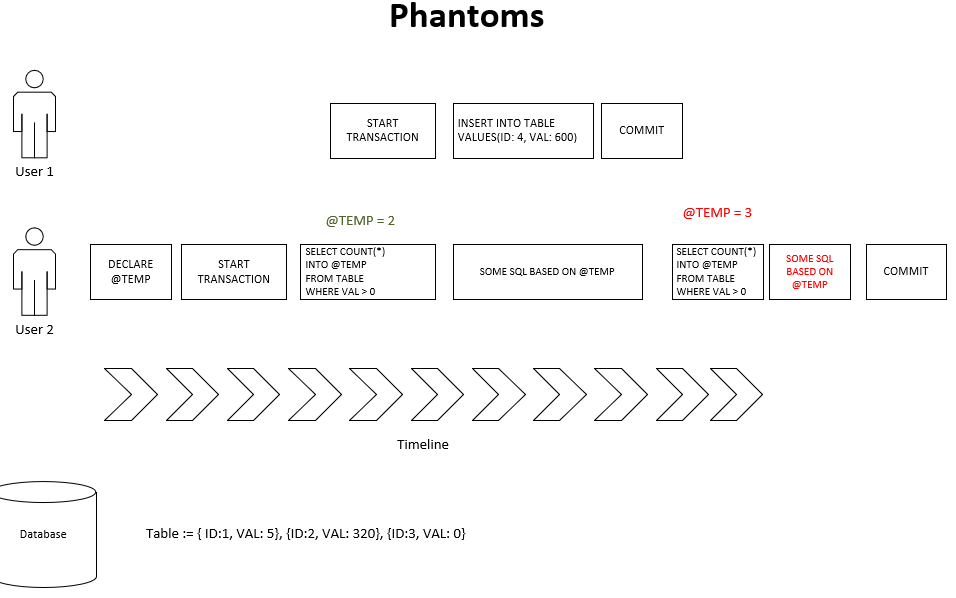


Na poziomie **powtarzalnego odczytu** możemy znaleźć remedium na tę **wydumaną** sytuację:

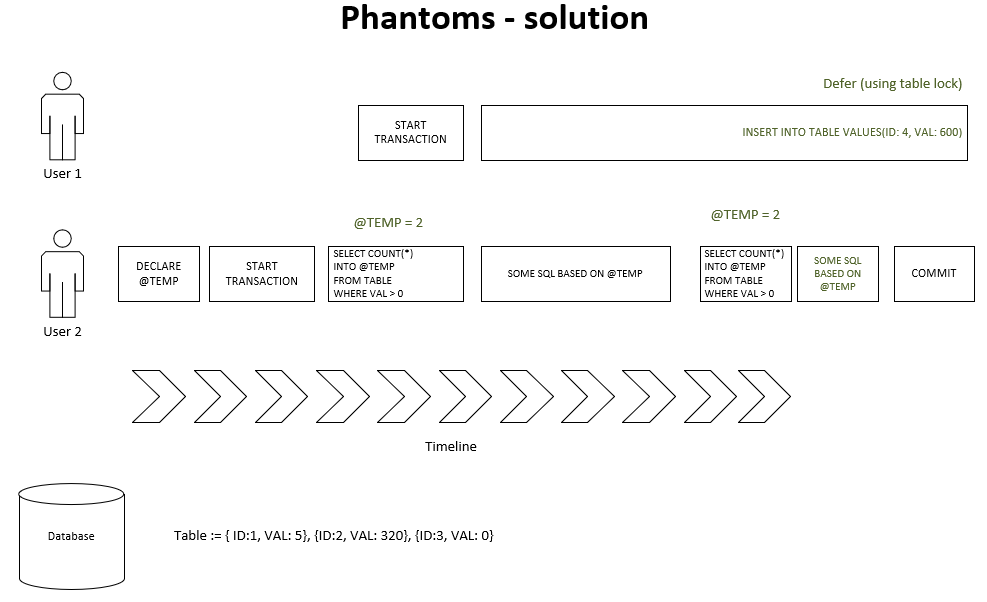


## anomalia – fantomy

Poziom izolacji **powtarzalny odczyt** błędnie sugeruje że zapytanie SQL zwraca ten sam wynik. Obrazuje to poniższa sytuacja (zwrócić uwagę na INSERT Usera 1):



Da się to naprawić na najwyższym poziomie izolacji **szeregowalnym:**



## poziomy izolacji - sql

W Oracle DB mamy dwa poziomy izolacji:

* Read Commited
* Serializable

Możemy wymusić żeby transakcja działała na konkretnym poziomie izolacji za pomocą wyrażeń:

1. Na poziomie transakcji
   1. SET TRANSACTION **ISOLATION LEVEL** **SERIALIZABLE**;
   2. SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
2. Na poziomie sesji:
   1. ALTER SESSION SET isolation\_level=serializable;

|  |
| --- |
| Hint: |
| Jak chcemy nadać nazwę transakcji to użyć:  SET TRANSACTION **ISOLATION LEVEL** **SERIALIZABLE** NAME [nazwa]; |

# Zadania

## Podzadanie 1

Dla bazy z poprzedniej listy napisać funkcję która dla podanego tytułu filmu zwraca jego średnią.

## Podzadanie 2

Dla bazy z poprzedniej listy napisać procedurę która dla podanego gatunku filmowego zmienia oceny każdego filmu z tego gatunku na wartość będącą średnią wszystkich ocen tego filmu.

## Podzadanie 3

Napisać program w dowolnym języku programowania działający w konsoli prezentujący anomalie dla poziomów izolacji dostępnych w wybranej BD – Oracle-owcy są na uprzywilejowanej pozycji 😊. **Pamiętać o tym, że anomalie występują przy współbieżnym dostępie.** Oznacza to, że:

* będzie musieli nawiązać z języka programowania dwa niezależne połączenia
* będziecie musieli wprowadzić sztuczne opóźnienia pomiędzy komendami COMMIT, ROLLBACK oraz SQL-ami

Na przykład by pokazać brudny odczyt (poziom izolacji READ UNCOMMITED):

* Połączenie 1 powinno wykonać start transakcji na poziomie izolacji READ UNCOMMITED, poczekać 10 sekund po czym wykonać SELECT i zakończyć działanie.
* Równolegle połączenie 2 powinno wykonać start transakcji na poziomie izolacji READ UNCOMMITED, natychmiastowo wykonać UPDATE, poczekać 20 sekund i wykonać ROLLBACK.

## Sprawozdanie

Ze wszystkich 3 zadań chciałbym otrzymać sprawozdanie do 8:00 dnia 2020-05-11.