Katedra
Informatyki i Automatyki
Politechniki Rzeszowskiej

Bazy Danych

3. Transakcje / Programowanie Baz Danych

Opracował: Maciej Penar

Spis treści

1. Zanim zaczniemy

Zrelaksować się i przyswoić sobie teorię dot. transkacji – w szczególności własności ACID, poziomów izolacji oraz SQL'a do operowania na transakcjach.

Materiały:

- SQL: https://pl.wikipedia.org/wiki/SQL
- Podstawowy kurs systemów baz danych, rozdział … o SQL'ach (nie mam książki przy sobie), J. Ullman, J. Widom
- O transakcjach: link
- Poziomy izolacji w ANSI/SQL: link
- SQL transakcji dla Oracle: link
- Funkcje w Oracle: link
- Procedury w Oracle: link

Oprogramowanie:

- ORACLE Database c12
    - SQL Developer

Fragmenty dokumentacji Oracle 12c:

- CREATE TABLE: link
- Indeksy: link
- Dziedziczenie: link
- Przydatne funkcje dla obiektów: link
- Typy danych: link

2. Trochę o transakcjach

SŁOWO WSTĘPU / ACID

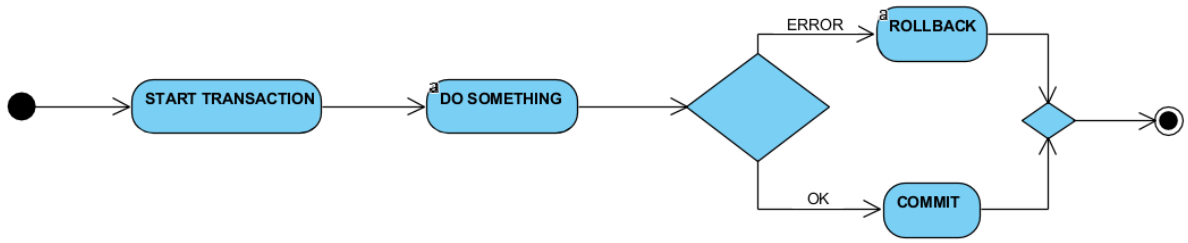
Pojęcie transakcji wywodzi się z Systemów Operacyjnych. Transakcja stanowi jednostkę przetwarzania w ramach jakiegoś systemu (my rozważamy Bazy Danych). Operacja transakcji jest scharakteryzowana czterema cechami:

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:
    a. Spójność jako zachowanie ograniczeń (więzy CHECK / ograniczenia klucza obcego)
    b. * 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 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:

a) SELECT * FROM TR WHERE ID = 1;
b) 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:

a) Są wykonywane współbieżnie
b) 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 rozwiązują one najgorszy możliwy scenariusz – nadpisanie danych jednej transakcji przez drugą.

Techniki pesymistyczne:

a) Oparte są o blokady
b) 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:

a) **Dirty Read** - Brudny odczyt – kiedy transakcja odczytuje niezatwierdzone dane
b) **Non-Repetable Read** - Niepowtarzalny odczyt – transakcja **dwukrotnie** odczytuje zbiór instancji. Instancje krotek ulegają zmianie (na wskutek wyrażeń UPDATE) przy drugim odczycie.
c) **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:

a) Read uncommited – odczyt niezatwierdzony
b) Read commited – odczyt zatwierdzony
c) Repetable Read – odczyt powtarzalny
d) Serializable – szeregowalny

Uwaga: poziom „odczyt powtarzalny" **nie** oznacza że za każdym razem jak powtórzymy odczyt to otrzymamy ten sam zbiór krotek. Link (0:03-0:08)

Tabela poziomów izolacji i możliwych anomalii wygląda tak:

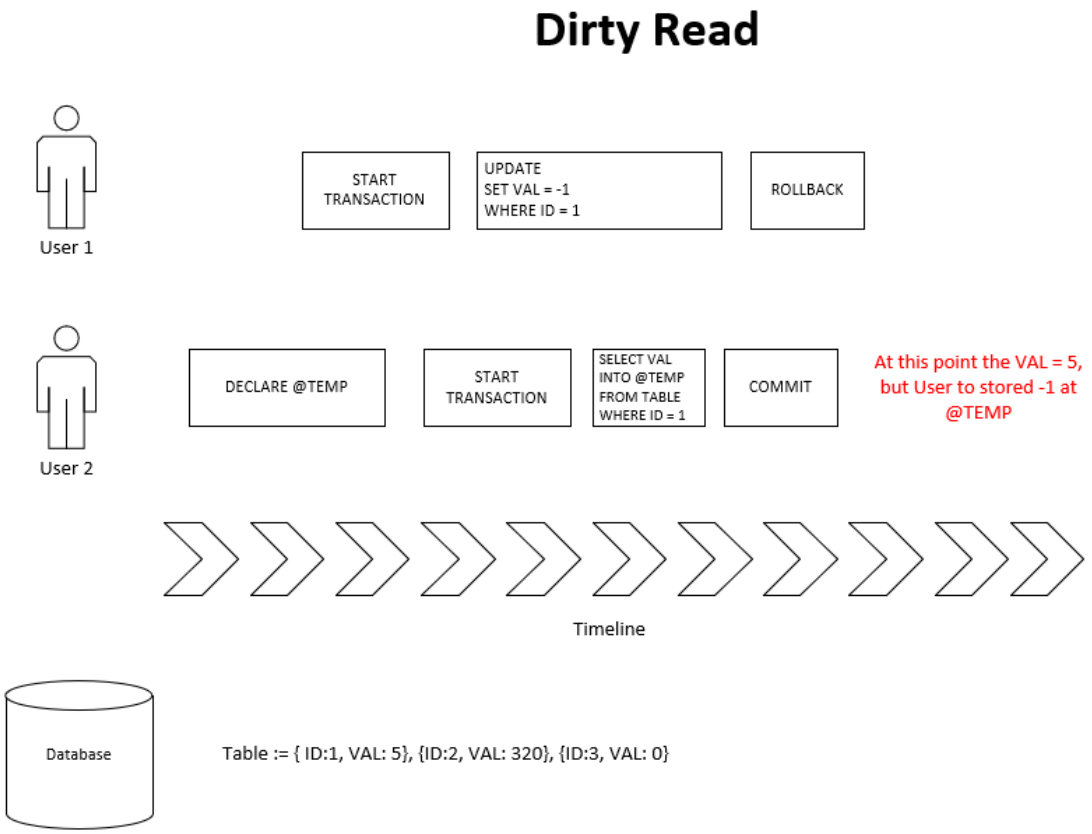
Poziom Izolacji \Anomalia	Dirty Read	Non-Repetable Read	Phantoms
Read Uncommited	występuje	występuje	występuje
Read Commited		występuje	występuje
Repetable Read			występuje
Serializable			

Spostrzeżenie:

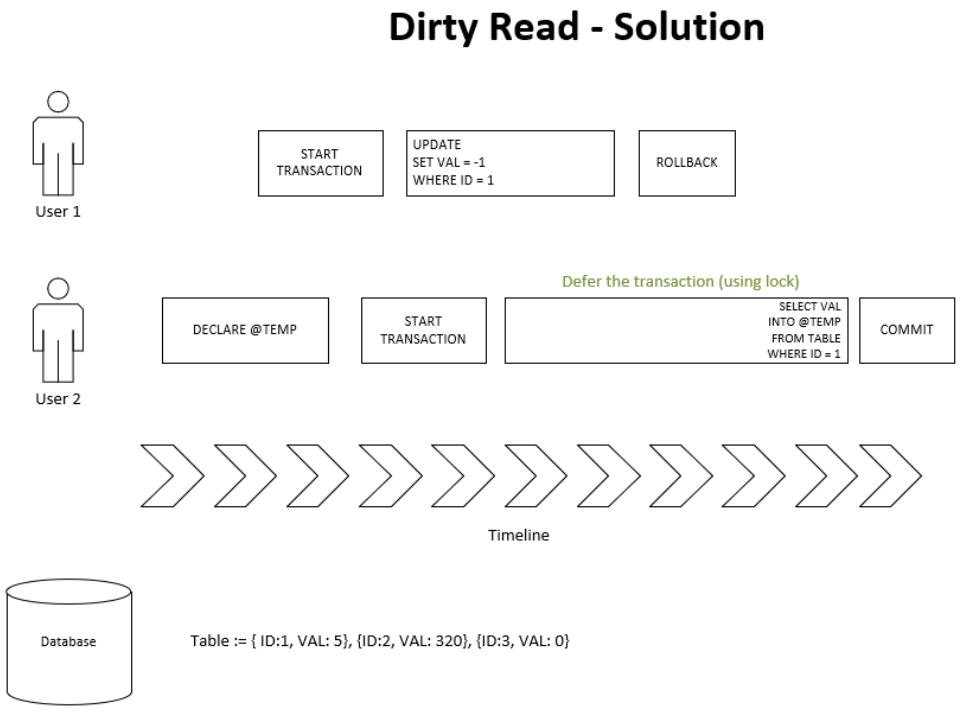
Poziom izolacji **READ COMMITED** jest **optymalny** dopóki Wasze transakcje składają się z **pojedynczych** wyrażeń SELECT / INSERT/ UPDATE / DELETE. **Nie** zaobserwujecie anomalii niepowtarzalnego odczytu, a wpływ fantomowych krotek **zazwyczaj** jest znikomy.

ANOMALIA – BRUDNY ODCZYT

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

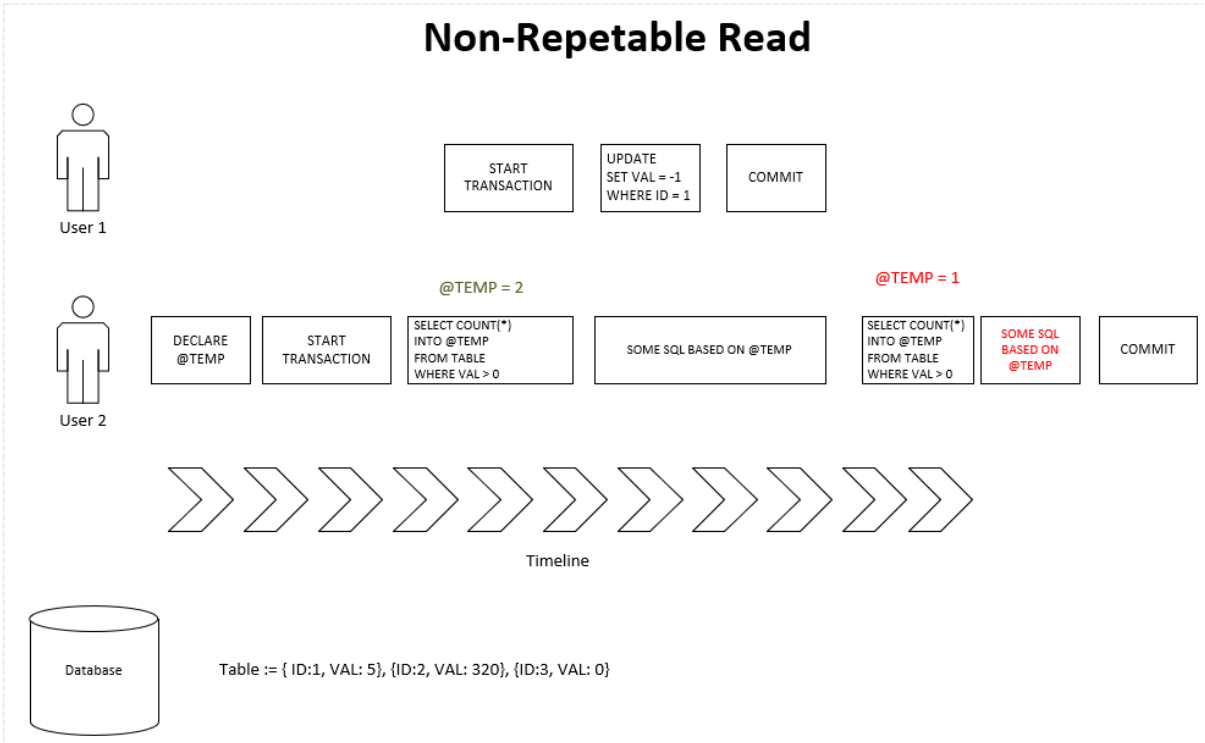


Na poziomie izolacji **zatwierdzonego odczytu** możemy naprawić tę sytuację:

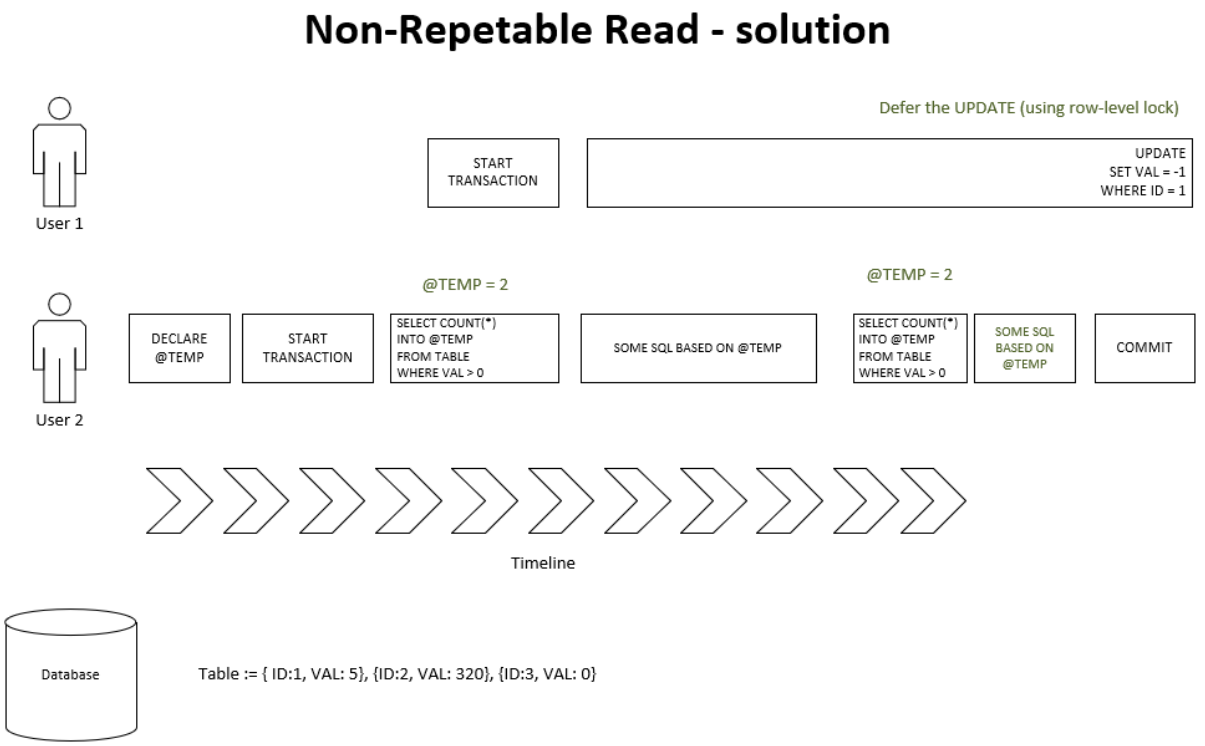


ANOMALIA – NIEPOWTARZALNY ODCZYT

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

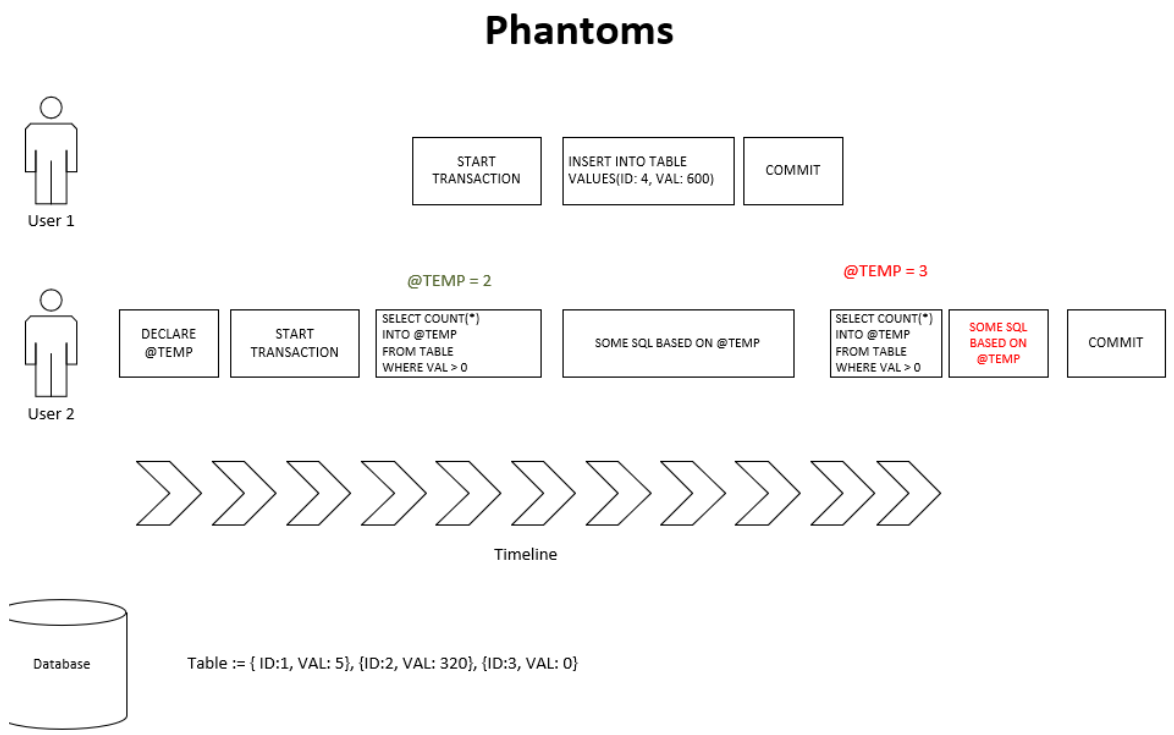


Na poziomie **powtarzalnego odczytu** możemy znaleźć remedium na tę **wydumaną** sytuację:

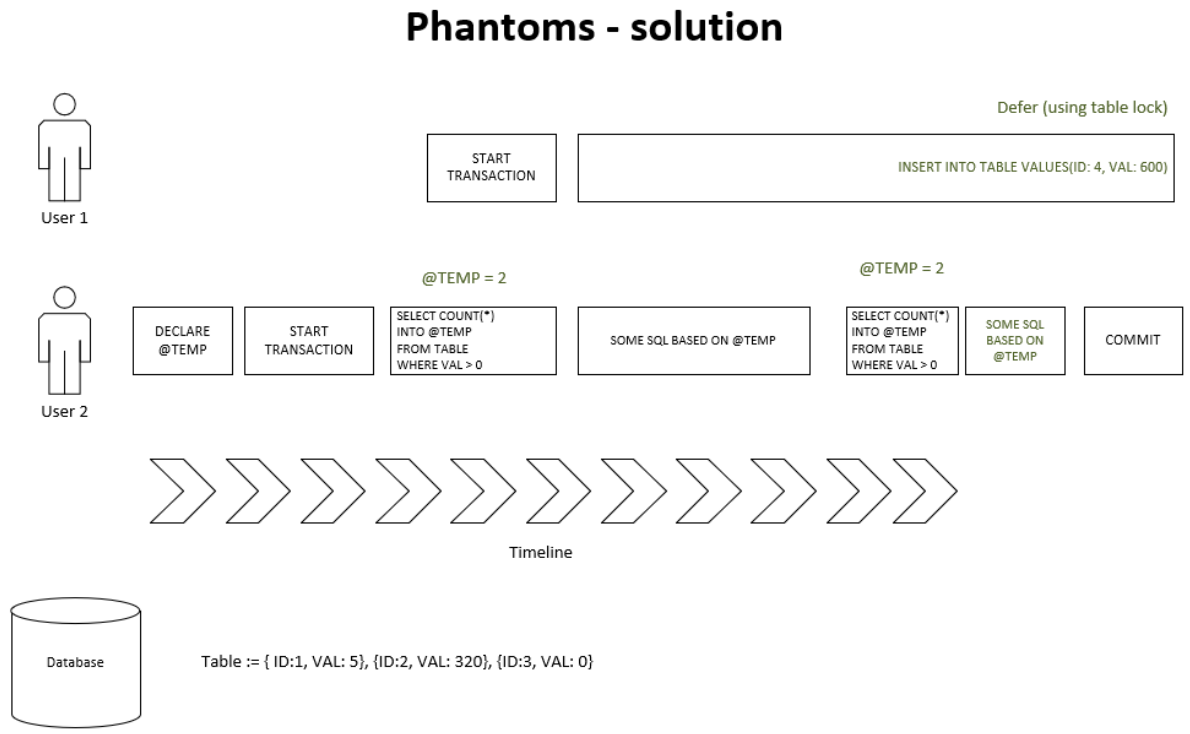


ANOMALIA – FANTOMY

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:

a) Na poziomie transakcji
    a. SET TRANSACTION **ISOLATION LEVEL SERIALIZABLE**;
    b. SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
b) Na poziomie sesji:
    a. ALTER SESSION SET isolation_level=serializable;

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

PODSUMOWANIE

- Transakcja jako jednostka przetwarzania
- Transakcja cechuje się ACID
- Słowa kluczowe **ROLLBACK** i **COMMIT** służą do realizacji właściwości **Atomowości**
- Bazy Danych działają domyślnie na poziomie izolacji **READ COMMITED**
- 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.

3. Trochę o programowaniu bazy danych

LOGIKA APLIKACJI W BAZIE DANYCH

Dużym osiągnięciem Inżynierii Oprogramowania jest możliwość wykonywania logiki aplikacji po stronie bazy danych. Wady i zalety takiej możliwości są następujące:

Zalety	Wady
1. Zwracanie do programu minimalnego zestawy danych 2. Zmniejszenie ruchu sieciowego (wiąże się z 1) 3. Wykonywanie operacji bezpośrednio na danych (wiąże się z 2)	1. Różnie jest z wydajnością kodu logiki po stronie Baz Danych – **Oracle uchodzi za wydajny** 2. Mała wygoda / Duże ograniczenia kodu proceduralnego 3. Łączenie stylu deklaratywnego (SQL) i proceduralnego (PL/SQL) powoduje że optymalizatory mają „magiczne" reguły (patrz. parameter sniffing) 4. Za dużo kodu po stronie bazy danych wiąże aplikacje – ciężko przeprowadzić migrację 5. Radyklane podejście (tj. dużo logiki po stronie Bazy Danych) może powodować zmniejszenie wydajności na wskutek: o Wyzwalania zdarzeń o Blokowania się tabel (poziomy izolacji zaczynają odgrywać istotną rolę)

Spostrzeżenie:

Nikt nie programuje całej aplikacji w Bazie Danych (patrz. Popularność Oracle APEX / Architektury dwuwarstwowej). I dobrze.

Co można oprogramować w Bazie Danych? Zasadniczo trzy elementy:

1. Wyzwalacze – fragmenty kodu uruchamiane podczas zdarzeń INSERT/UPDATE/DELETE
2. Procedury – fragmenty kodu wykonujące logikę
3. Funkcje – uszczegółowienie procedur, fragmenty kodu wykonujące logikę i zwracające wartości, a nawet i tabele (lub jej zamiennik).

Zajmiemy się **funkcjami** (niedaleko im do procedur) – wyzwalacze na razie sobie podarujemy.

FUNKCJE

Omówimy funkcje na podstawie dwóch przykładów.

Przykład 1: funkcja w Oracle zwracająca wartość bezwzględną:

```
CREATE OR REPLACE FUNCTION MyAbs(x IN NUMBER)
RETURN NUMBER IS
  output NUMBER;
BEGIN
  IF x > 0 THEN output :=x; ELSE output := -x; END IF;
  RETURN output;
END;
```

Aby przetestować można skorzystać z pustej tabeli (DUAL)

```
SELECT MyAbs(-6) FROM dual;
```

W deklaracji funkcji istotne fragmenty to:

`CREATE OR REPLACE FUNCTION [nazwa]` Np. `CREATE OR REPLACE FUNCTION MyAbs`	Gdzie nazywamy funkcję
`[nazwa]([nazwa parametru] IN|OUT [typ])` Np. `MyAbs(x IN NUMBER)`	Gdzie specyfikujemy nazwy parametrów oraz typ oraz czy jest to parametr wejściowy/wyjściowy
`RETURN [typ] IS` Np. `RETURN NUMBER IS`	Gdzie specyfikujemy jaki jest typ zwracany. Słowo IS zaczyna ciało funkcji
`[nazwa zmiennej] [typ zmiennej];` Np. `output NUMBER;`	Pomiędzy IS a BEGIN znajduje się sekcja deklaracji zmiennych

`BEGIN`
`  …`
`END;`
	Konstrukcja Begin/End grupuje kilka operacji

`IF x > 0 THEN output :=x; ELSE output := -x;`
`END IF;`
`RETURN output;`
	Logika funkcji

Przykład 2: funkcja w Oracle wykonująca SELECT (baza danych TPC-H):

```
CREATE OR REPLACE FUNCTION TESTF(brand IN CHAR)
RETURN NUMBER IS
  output NUMBER;
BEGIN
  SELECT COUNT(*) AS x INTO output FROM PART WHERE P_BRAND = brand;
  RETURN output;
END;
```

Aby przetestować można skorzystać tak:

```
SELECT P.*, TESTF(P.P_BRAND) FROM PART P;
```

W deklaracji funkcji istotne nowe fragmenty to:

`SELECT COUNT(*) AS x INTO output`	Umożliwia umieszczenie wartości w zmiennej

PODSUMOWANIE

- Nie programować całej logiki aplikacji w Bazie Danych. Można, ale to nie działa.
- Zagadnienie programowania Baz Danych jest szersze i wrócimy do niego na następnych listach

4. Trochę o rekursji

Rekursja w SQL'u jest ograniczona i opiera się na CTE (Common Table Expressions) – czyli WITH. Ogólny schemat jest następujący:

```
WITH TEMP(...) AS(
  SELECT ... FROM TEST
  UNION ALL
  SELECT ... FROM TEST INNER JOIN TEMP ON [warunek]
)
SELECT * FROM TEMP;
```

Zwrócić uwagę na dwa istotne fragmenty:

`SELECT ... FROM TEST`
Dzięki któremu otwieramy rekursję (zbiór początkowy)

Oraz całą istotę rekursji:

`SELECT ... FROM TEST INNER JOIN TEMP ON [warunek]`
Dzięki czemu dodajemy krotki do zbioru TEST, w definicji odwołując się do niego samego.

Ograniczenia:

- Nie można stosować funkcji agregujących
- Rekursja może tylko dodawać krotki (monotoniczność)
- Rekursja musi się skończyć

5. (6 pkt) Transakcje

Zaprojektować eksperyment w formie skryptów SQL (w ramach eksperymentu może być ich kilka).

1. (1 pkt) Wykonać jawnie zatwierdzony INSERT
2. (1 pkt) Wykonać jawnie cofnięty INSERT
3. (2 pkt) Dobrać poziom izolacji. Pokazać anomalię niepowtarzalnego odczytu / fantomowych krotek
4. (2 pkt) Dobrać poziom izolacji. Pokazać sytuację gdy niepowtarzalny odczyt / fantomowe krotki mogły się pojawić, ale się nie pojawiają

6. (3 pkt) Wstęp do programowania baz danych

Napisać funkcje:

1. (1 pkt) Napisać funkcję power2 która wykonuje potęgowanie za pomocą pętli
2. (1 pkt) Dla bazy danych TPC-H napisać funkcję która zwraca identyfikator klienta (CUSTOMER) na podstawie nazwiska (C_NAME)
3. (1 pkt) Dla bazy danych TPC-H napisać funkcję która zwraca liczbę klientów (CUSTOMER) w ramach danego segmentu (C_MKTSEGMENT)

7. (3 pkt) Zaawansowany SQL - rekursja

Wykonać SQL:

```
CREATE TABLE ENTITY(ID INT PRIMARY KEY, PARENT_ID INT);
INSERT INTO ENTITY VALUES(0,NULL);
INSERT INTO ENTITY VALUES(1,NULL);
INSERT INTO ENTITY VALUES(2,1);
INSERT INTO ENTITY VALUES(3,1);
INSERT INTO ENTITY VALUES(4,2);
INSERT INTO ENTITY VALUES(5,3);
INSERT INTO ENTITY VALUES(6,5);
INSERT INTO ENTITY VALUES(7,5);
INSERT INTO ENTITY VALUES(8,7);
INSERT INTO ENTITY VALUES(9, NULL);
INSERT INTO ENTITY VALUES(10,9);
```

1. (1.5 pkt) Dla każdego bytu znaleźć wszystkich bezpośrednich oraz niebezpośrednich rodziców
2. (1.5 pkt) Dla każdego bytu znaleźć ilu jest bezpośrednich oraz niebezpośrednich rodziców

Uwaga: Jeśli byt rodziców ma, jeśli nie ma to i tak wyświetlić dany byt. Tj. do wyniku wliczamy (1, NULL) bo stoi na czele, wliczamy też (2,1), (4, 1) ale już (2, NULL) i (4, NULL) nie