Komentarze

Bazy danych: Procedury i funkcje

Spis treści

[Refleksje 3](#_Toc40081826)

[One upmanship 3](#_Toc40081827)

# Refleksje

Tu macie zbiorcze refleksje

## One upmanship

*If you’re not one up, you’re one down*

Nie w tym przypadku:

CREATE FUNCTION dbo.SredniaOcena (@tytul nvarchar(50))

…

m.title LIKE @tytul

Wywołanie dbo.SredniaOcena(‘ToyStory%’)… ups

O ile nie znam dużo filmów które mają % w tytule (tu raczej potencjalny side effect się liczy), to jest jeszcze kwestia: [link](https://www.imdb.com/list/ls004630614/)

## One upmanship (2)

*If you’re not one up, you’re one down*

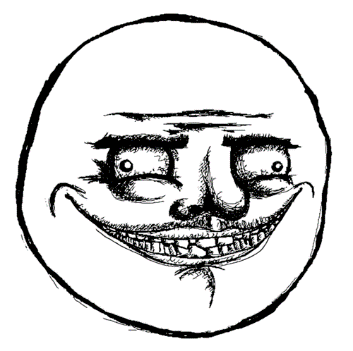
Nie w tym przypadku:

CREATE FUNCTION dbo.F(@Name VARCHAR(500)) RETURNS **TABLE**

Zwrócić **średnią** to nie to samo co zwrócić **tabelę z średnią.**

To co najlepsze w funkcjach-zdefiniowanych-przez-użytkownika (User Defined Functions - UDF), to to że można je wykorzystać do indeksowania: funkcje tabelaryczne mają zgoła inne zastosowanie (exploding)

## Me gusta





## What happens next?

Przy anomaliach brakuje mi omówienie co się dzieje w danej sytuacji jeśli wybierzemy poziom izolacji na którym anomalie znikają.

Najbardziej interesującą mnie anomalią jest oczywiście ta najbardziej niejasna: **niepowtarzalny odczyt**

## do 160769

Zadanie 2 mi się nie kompiluje

## Fun fact

|  |
| --- |
|  |

A to dlatego, że SQL Server odświeża obiekty bazodanowe tylko on-demand/przy połączeniu. Jak się odłączysz i przyłączysz do bazy z powrotem to będzie git.

## do 160761

This is how it should be done

## do 160773

Nie wiem po co te flagi w zad 2. Jakby wywalić SET @flag = 1; IF (@flag =0 ) to działałby tak samo.

## Fun fact (2)

Pokochacie mnie za to co tu napiszę: ogólnie to widzę, że zadanie 2 zostawiłem zbyt dowolne i nikt nie pokusił się o idempotentność działania – bo formułowałem zadanie wychodząc od 1-krotnego wywołania.

**Złotym standardem pisania procesów ETL jest osiąganie takiej implementacji której kilkukrotne wywołanie nie wpływa na stan BD – niezależnie jaki by on nie był.**

Najlepsza byłaby następująca realizacja:

1. W pierwszym kroku sprawdzenie czy Users\_Info jest
   1. Jeśli jest to sprawdzić czy FK z users do ratings jest – jak tak to usunąć
   2. Jeśli nie to utworzyć Users\_Info
2. Usunięcie danych z Users\_Info
3. Załadować tabelę Users\_Info na podstawie ratings
4. Utworzenie FK z users do ratings

…. Ale i tak akceptuje co napisaliście, bo jest to zgodne ze specką zadania.

## **NIEPRAWDOPODOBNIE NIEPOWTARZALNY ODCZYT NA POZIOMIE IZOLACJI POWTARZALNEGO ODCZYTU**

Załóżmy taką tabelę:

CREATE TABLE [dbo].[X](

[X] [int]

) ON [PRIMARY];

GO

Ustalmy na niej stan:

TRUNCATE TABLE X;

INSERT INTO X(X) VALUES(1),(2),(3);

Wy na ogół wykazywaliście taki case:

|  |  |
| --- | --- |
| Przypadek: Powtarzalny odczyt na poziomie izolacji powtarzalnego odczytu | |
| Transakcja 1 | Transakcja 2 |
| BEGIN TRANSACTION  SET TRANSACTION ISOLATION LEVEL REPEATABLE READ  SELECT X FROM X WHERE X = 1;  /\* Czekamy na start T2  Po czym wykonujemy:  \*/  SELECT X FROM X WHERE X = 1;  /\* Powinno się wykonać, a T2 powinna czekać \*/  COMMIT | BEGIN TRANSACTION  SET TRANSACTION ISOLATION LEVEL REPEATABLE READ  UPDATE X  SET X = 4  WHERE X = 1;  COMMIT |

A kto zwrócił uwagę na taki flow?

|  |  |
| --- | --- |
| Przypadek: Powtarzalny odczyt na poziomie izolacji powtarzalnego odczytu 2 | |
| Transakcja 1 | Transakcja 2 |
| BEGIN TRANSACTION  SET TRANSACTION ISOLATION LEVEL REPEATABLE READ  SELECT X FROM X WHERE X = 1;  /\* Czekamy na start T2  Po czym wykonujemy:  \*/  SELECT X FROM X WHERE X = 1;  COMMIT | BEGIN TRANSACTION  SET TRANSACTION ISOLATION LEVEL REPEATABLE READ  UPDATE X  SET X = 1  WHERE X = 3;  /\* Nie commitujemy \*/  COMMIT |
| To jest bardziej kwestia READ COMMITTED | |

A w zasadzie taki::

|  |  |
| --- | --- |
| Przypadek: Powtarzalny odczyt na poziomie izolacji powtarzalnego odczytu 2 | |
| Transakcja 1 | Transakcja 2 |
| BEGIN TRANSACTION  SET TRANSACTION ISOLATION LEVEL REPEATABLE READ  SELECT X FROM X WHERE X = 4;  /\* Czekamy na start T2  Po czym wykonujemy:  \*/  SELECT X FROM X WHERE X = 4;  COMMIT | BEGIN TRANSACTION  SET TRANSACTION ISOLATION LEVEL REPEATABLE READ  UPDATE X  SET X = 4  WHERE X = 1;  COMMIT |

Co prowadzi do pytania: **co w zasadzie podlega blokowaniu?**

Co prowadzi do kontrpytania: [link](https://dba.stackexchange.com/questions/12864/what-is-lock-escalation)

Ofc: „behaviour may vary” dla każdej BD

## do 158369

Na pierwszy rzut oka zadanie 2 nie powinno się nigdy wykonać, bo kładziesz FK przed załadowaniem danych.

## do 160759

Dorzucam w bonusie 0,5 pkt za tę piękną funkcję losującą.

## A propos H2

### U 160759:

Zaciekawiło mnie to stwierdzenie: Na rysunku wyżej możemy zauważyć poprawne działanie kodu. Jest to spowodowane, że baza danych H2 **ma już wbudowaną izolacje anomalii phantoms na poziomie izolacji Repetable Read.**

Nie mogę znaleźć tej informacji: [link](https://h2database.com/html/advanced.html) -tu jest sugestia, że na domyślnym silniku jest Repeatable read który: „Dirty reads and non-repeatable reads aren't possible, **phantom reads are possible.”.**

Z kolei na silniku PageStore nie ma repeatable read i jest adnotacja: „If repeatable read isolation level is requested when using a PageStore engine it is replaced with serializable isolation level.”.

### U 160794

Mając na względzie powyższy cytat + link nie wiem czy przychylam się do stwierdzenia: „*Co ciekawe - Anomalia* ***nie występuje****. A powinna. Dlaczego nie?* Odpowiedź jest jedna”.

## do 160823

W procedurze próbowałeś zrobić postawić tabelę tymczasową: [link](https://codingsight.com/introduction-to-temporary-tables-in-sql-server/)

Wystarczyło by zmodyfikować:

|  |
| --- |
| DROP TABLE IF EXISTS temp  SELECT  Id as userId,  movieId,  rating,  timestamp  INTO temp  FROM  dbo.ratings  INNER JOIN dbo.Users\_Info ON dbo.ratings.userId = dbo.Users\_Info.Name; |

Na równoważne:

|  |
| --- |
| SELECT  Id as userId,  movieId,  rating,  timestamp  INTO #temp  FROM  dbo.ratings  INNER JOIN dbo.Users\_Info ON dbo.ratings.userId = dbo.Users\_Info.Name; |

## do 160781

Nie rób starej składni złączeń (przy funkcji).

FROM dbo.ratings r, dbo.movies m WHERE ………. AND m.movieId = r.movieId

+

„Procedura została utworzona bez żadnych błędów.”



## do 160777

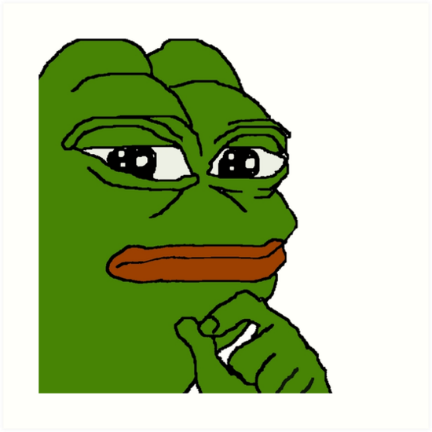
Ad 1) Do podanego **tytułu filmu –** nie Id

Ad 2) A co z FK gdy tabeli nie ma?

## do 160766 – You livin’ da past man

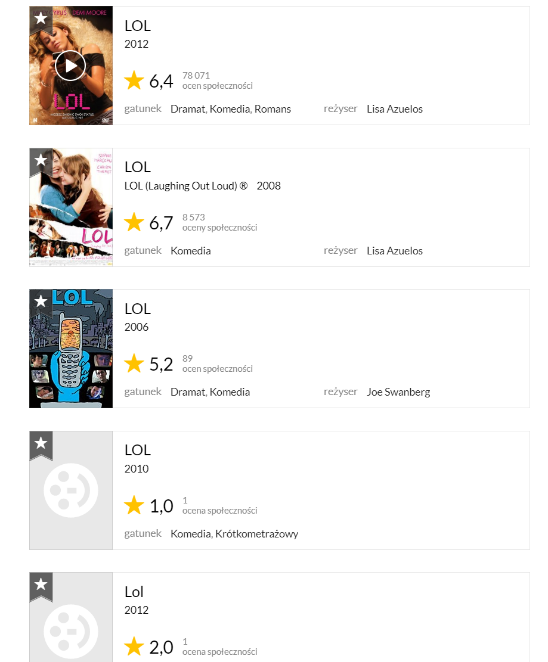
|  |
| --- |
|  |

## Do 160766



**WHERE LOWER**(m.title) = **LOWER**(@title)

Po 15 minutach myślenia czy lowercase-owany tytuł powoduje wejście rekordów na dane innego filmu …. Odpuszczam. Nie znalazłem kontraprzykładu…….. I TERAZ DOSTAŁEM PRZEBŁYSKU JAK TO PISZĘ: JEST! JEST!:



2 razy LOL i Lol z 2012 roku….

*it would be very bad if this one 2 star grade would influence Your movie… wouldn’t it?*

## Niby tak można, ale proszę: nie



W klauzuli JOIN … ON – warunki powinny rekonstruować to co „napsuła” normalizacja tj. w 99% przypadków to powinny być warunki FK = FK. Bardzo rzadko pojawia się potrzeba łączenia po innych kolumnach niż te oznaczone FK…. (pozostały 1% przypadków to gdy kolumny nie zostały oznaczone jako FK, ale znaczeniowo **są** one FK)

Warunek m.title = @title nie wpływa na złączenie i nie pracuje na kolumnie typu FK – więc powinien znaleźć się w klauzuli WHERE.

## Do 157760

Trzeba było zapytać – to nie kosztuje :P

„**Czyli jak wynaleźć koło na nowo i dotrzeć do najdalszych czeluści Internet. Plan był prosty – do tabeli *Users\_Info* w pole name wpisywać nazwy wygenerowane** ~~przez funkcję rekurencyjną. (wykonanie wręcz przeciwnie) Jak każdy generator trzeba zacząć od funkcji zwracającej losowe wartości – takiej jak~~ *~~NEWID()~~*~~.~~ *~~RAND()~~* ~~przy wielokrotnym wywołaniu zwraca tą samą wartość. Niestety, ale żadnej z tych funkcji nie można użyć w funkcji użytkownika. Dlatego należy stworzyć widok który będzie zwracał pierwszą wartość~~ **z*NEWID().*”**

I takie coś akceptowałem – zresztą tak samo jak HASHBYTES/CAST-y.

Clu zadania polegało na odwołaniach do metadanych BD (sys.\*)

## NA marginesie – zadanie 2

Na marginesie: szczerze powiedziawszy wydawało mi się, że ALTER TABLE + ADD CONSTRAINT musi iść jako osobny wsad i trzeba wykonać haczyk z **dynamicznym SQL-em.** Na ogół jak w dyskusji się pojawia „dynamiczny SQL” to niezorientowani podnoszą larum. Kwestia z dynamicznym SQL jest taka jak zawsze:

**żeby używać to trzeba wiedzieć „kiedy”**

Oczywiście jak teraz sprawdzam to tylko TRIGGERY oraz VIEW muszą iść jako wsady… więc zadanie sprowadzało się do sprawdzenia tabel systemowych.