Bazy Danych

4. Gry i zabawy z Triggerami (i nie tylko)

Opracował: Maciej Penar

Spis treści

[1. Elementy aktywne 3](#_Toc39394132)

[2. CHECK 3](#_Toc39394133)

[3. TRIGGER-y jako elementy aktywne 3](#_Toc39394134)

[Zadanie: Gry i zabawy z triggerami (i nie tylko) 4](#_Toc39394135)

[1) Złe zastosowanie triggerów - dziennikowanie 4](#_Toc39394136)

[2) klasyczne zastosowanie triggerów 4](#_Toc39394137)

[3) dziwne zastosowanie triggerów 4](#_Toc39394138)

[4) Pytanie 5](#_Toc39394139)

[5) Dziennikowanie raz jeszcze, ale tym razem lepiej 5](#_Toc39394140)

[Termin listy 6](#_Toc39394141)

# 1. Elementy aktywne

Elementy aktywne odpowiadają za wykonanie pewnej programowalnej logiki w pewnym ustalonym momencie – zależnym od wybranego elementu aktywnego.

I tak do aktywnych komponentów BD możemy zaliczyć:

* Więzy (np. więzy CHECK, FOREIGN KEY z domieszką ON DELETE/ON UPDATE)
* Triggery
* Materialized Query Tables / Widoki zmaterializowane / Widoki indeksowane
* …. Na siłę: indeksy
* Change Data Capture

Podział nie jest taki istotny**: istotne jest, że wykorzystując którykolwiek z komponentów zyskujemy pewną funkcjonalność lub poprawiamy wydajność (najczęściej SELECT-ów)… kosztem wydajności (najczęściej INSERT-ów/UPDATE-ów).**

# 2. CHECK

Więzy typu **check** służą do sprawdzenia **przed** wykonaniem transakcji update/insert, czy spełnione są pewne warunki. Te więzy można podać niejawnie w definicji tabeli np.:

CREATE TABLE People(

Id INT IDENTITY(1,1),

Wiek INT CHECK(Wiek > 0)

);

Co zabroni bazie danych wsadzenia do tabeli People takich rekordów które mają niedodatnią wartość w kolumnie Wiek. Trzeba pamiętać, że w takiej formie nie podajemy nazwy więzów, przez co Baza Danych sama ją wybierze. Gdybyśmy chcieli nadać nazwę lub zapomnieli dodać więzy to możemy zrobić to w dowolnym momencie za pomocą komendy ALTER TABLE:

ALTER TABLE People

ADD CONSTRAINT OnlyYoungPeople CHECK(Wiek < 19);

# 3. TRIGGER-y jako elementy aktywne

Triggery stanowią uzupełnienie procedur składowanych / więzów. O ile utworzone procedury składowane, pozostawione same sobie nie wykonają się nigdy, to logika Trigger-ów parowana jest z momentem ich ‘odpalenia’.

Uproszczona składnia jest następująca:

CREATE [ OR ALTER ] TRIGGER [ schema\_name . ]trigger\_name

ON { table | view }

{ FOR | AFTER | INSTEAD OF }

{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }

AS BEGIN

// SQL

END

Ponownie, istotne elementy triggera to:

* Jego nazwa (CREATE TRIGGER [name])
* Na jakiej tabeli operuje (ON [table/view])
* W ramach jakiej akcji (INSERT/UPDATE/DELETE) oraz kiedy powinna zostać wykonana logika:
  + FOR – przed
  + AFTER – po
  + INSTEAD OF - zamiast
* Ciało triggera: AS, a najczęściej konstrukcja AS BEGIN … END

Triggerów nikt nie lubi i wykorzystywane są na ogół **błędnie**. W ramach tej listy popatrzymy na błędne zastosowania (zastanowimy się jak zrobić coś lepiej) oraz popatrzymy na ‘smaczki’ triggerów.

# Zadanie: Gry i zabawy z triggerami (i nie tylko)

Wykonać zadanie dla bazy danych MovieLensa (lista 2) z tabelą UserInfo (lista 3).

## 1) Złe zastosowanie triggerów - dziennikowanie

Załóżmy że jako projektanci chcemy „śledzić” zmiany w tabeli zawierającej oceny użytkowników. Jeśli użytkownik wykona aktualizację swojego wpisu (!) to chcemy wiedzieć:

* **Jaki był dowolny historyczny wpis dowolnego użytkownika – bez względu na to ile razy go aktualizował** – np. jaki był pierwszy/przedostatni wpis usera o id = 10
* **I chcemy wiedzieć kiedy aktualizował –** np. jaki był wpis usera od id 10 w dniu 2020-05-01 00:42:22

*Zaproponować odpowiednią tabelę, oprogramować stosowny TRIGGER i wykazać działanie mechanizmu – w sprawozdaniu podać zalety i wady takiego rozwiązania – w kontekście przyjętej struktury Bazy Danych.*

## 2) klasyczne zastosowanie triggerów

Załóżmy, że na backendzie chcemy operować bezpośrednio na nazwach użytkowników i ich ocenach (okazało się, że klucze sztuczne i tak są IDENTITY więc BD sama tym zarządza). Czyli jak mamy tabele:

Ratings(UserId,MovieId,Rating), User(UserId, Name), Movie(MovieId, Name)

**To chyba lepiej w kodzie operować na: XXX(UserName, MovieName, Rating)**

*Utworzyć odpowiedni* ***widok*** *i za pomocą triggera INSTEAD OF napisać implementację dla operacji INSERT/UPDATE/DELETE. Wykazać działanie, opisać wady i zalety.*

## 3) dziwne zastosowanie triggerów

Załóżmy, że tabela Users\_Info ma **fundamentalny** błąd: ma za długi typ danych w kolumnie Name np. VARCHAR(30). Co gorsza – nasz backend już korzysta z tej tabeli i zakłada taką długość… choć 99,999% userów ma długość Nazwiska zawierającą co najwyżej 14 znaków.

Sytuacja jest drastyczna… miejsce w serwerowni się kończy…

Najczarniejsza godzina

*Zaproponować nową mniejszą tabelę Users\_Info i wykonać „hot swap” (podmianę tabeli w czasie działania aplikacji) za pomocą triggera INSTEAD OF. Przyjąć odpowiednie założenia. Wykazać działanie, opisać wady i zalety.*

## 4) Pytanie

Mam tabele:

CREATE TABLE A(x INT )

CREATE TABLE B(x INT )

CREATE TABLE C(x INT )

I triggery:

ALTER TRIGGER ATr ON A

FOR INSERT AS

BEGIN

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

INSERT INTO B SELECT \* FROM inserted;

COMMIT

END

ALTER TRIGGER BTr ON B

FOR INSERT AS

BEGIN

INSERT INTO C SELECT \* FROM inserted;

END

Jeśli wywołujące INSERT INTO A VALUES(3); było wykonane na poziomie izolacji REPETABLE READ, a domyślny poziom izolacji **bazy danych** to READ COMMITED… to:

* Na jakim poziomie izolacji wykona się trigger ATr?
* Czy i na jakim poziomie izolacji wykona się trigger BTr?

**

## 5) Dziennikowanie raz jeszcze, ale tym razem lepiej

Załóżmy że jako projektanci chcemy „śledzić” zmiany w tabeli zawierającej oceny użytkowników. Jeśli użytkownik wykona aktualizację swojego wpisu (!) to chcemy wiedzieć:

* **Jaki był dowolny historyczny wpis dowolnego użytkownika – bez względu na to ile razy go aktualizował** – np. jaki był pierwszy/przedostatni wpis usera o id = 10
* **I chcemy wiedzieć kiedy aktualizował –** np. jaki był wpis usera od id 10 w dniu 2020-05-01 00:42:22

**I tym razem chcemy to zrobić lepiej:** służy do tego dedykowany typ tabel tzw. tabele temporalne (temporal tables) ([link](https://docs.microsoft.com/en-us/sql/relational-databases/tables/temporal-tables?view=sql-server-ver15)). Tabele te służą do przechowywania danych „czasowych” – najczęściej przedziałów czasowych w których zaistniało pewne zdarzenie. Zasadniczo mamy dwa typy tabel temporalnych: Systemowe (dostępne w SQL Serverze – śledzące stan Bazy Danych #RodoFriendly) i Biznesowe (np. systemy rezerwacji stolików, pokojów itp.). Tabele te efektywnie implementują szereg warunków wg. których możemy analizować przedziały czasowe. (Odsyłam na sam dół dokumentacji MS).

*Wykonajcie konwersję tabeli z ocenami użytkowników do tabeli temporalnej i wykażcie, że można ustalić:*

* *Pełną historię ocen użytkownika wg. filmu*
* *Stan oceny na konkretną godzinę,*
* *Stan ocen pomiędzy dwoma godzinami*

## Termin listy

Do 25 maja 8:00. Wystaczy docx/pdf z SQL-ami i omówieniem.

Gro oceny będą stanowić wnioski i spostrzeżenia dot. Triggerów.

Dołączcie proszę diagram UML bazy danych przed zmianami.