MECHANIKA
CZASOPISMO TECHNICZNE
TECHNICAL TRANSACTIONS
MECHANICS
WYDAWNICTWO POLITECHNIKI KRAKOWSKIEJ
4-M/2011
ZESZYT 7
ROK 108
ISSUE 7
YEAR 108

DARIUSZ KARPISZ*

IMPLEMENTACJA ASERCJI W RELACYJNYCH BAZACH DANYCH NA PRZYKŁADZIE SYSTEMU ORACLE

IMPLEMENTATION OF ASSERTION FOR RELATIONAL DATABASES WITH ORACLE EXAMPLES

Streszczenie

W artykule przedstawiono problem właściwej implementacji *więzów integralności* baz danych w postaci *asercji* we współczesnych systemach zarządzania bazami danych. Jak pokazano, w tym zakresie rozwój języka SQL nie kreuje nowych rozwiązań, a jedynie dostosowuje nowy standard do metod tworzenia *asercji* przy wykorzystaniu *wyzwalaczy* w takich systemach jak *Oracle*. Mimo ciągłego doskonalenia dostępnych w ramach baz danych funkcjonalności, prawidłowa obsługa nowych mechanizmów nastręcza wiele trudności i nie jest zgodna z obowiązującymi standardami.

Słowa kluczowe: *asercje, wyzwalacze, więzy integralności, SQL, Oracle*

Abstract

The paper presents the problem of the proper implementation of *integrity constraints* of databases in modern database management systems. As shown, in this regard, the development of SQL language does not create new solutions, but only adjusts to the new methods for creating *assertions* using *triggers* in systems such as the *Oracle*. Despite continuous improvement in the available database functionality, correct implementation of new mechanisms is difficulties and is not compatible with current standards.

Keywords: *assertion, triggers, integrity checks, SQL, Oracle*

* Dr inż. Dariusz Karpisz, Instytut Informatyki Stosowanej, Wydział Mechaniczny, Politechnika Krakowska.

1. Wstęp

W pierwszych systemach bazodanowych, pozbawionych zaawansowanej funkcjonalności opartej na proceduralnych rozszerzeniach języka SQL, asercje były jednym z elementów zwiększających zakres automatycznego monitorowania zasad integralności danych.

Wizja aktywnej bazy danych czuwającej nad spełnieniem pewnych określonych własności czy też niezmienników była przyczyną rozwoju wyzwalaczy i asercji. O ile więzy typu *CHECK* stały się standardowym elementem często występującym w definicjach relacji, to w przypadku asercji nawet uznani producenci rozwiązań bazodanowych borykają się z prawidłową obsługą implantacji asercji. W artykule omówiony zostanie problem prawidłowej implementacji polecenia *CREATE ASSERATION*, który przy jednoczesnej rozbudowie innych funkcjonalności systemu wydaje się niemożliwy do rozwiązania.

Mimo rozwoju języka SQL i deklarowanej zgodności popularnych systemów DBMS (**D**atabase **M**anagement **S**ystem) ze standardem SQL92 [1], aseracje nie stały się standardowym elementem implementowanym w interpreterach SQL. Również oczekiwane zmiany w ramach SQL3 (SQL:1999) [2] nie wpłynęły na stopień integracji struktur aseracyjnych w obecnych systemach *DBMS*. Podobnie stało się w przypadku systemów Oracle 10g i 11g. Implantacja asercji w systemach rodziny Oracle wydaje się prosta, ale rodzi również wiele problemów związanych z różnorodnością oferowanych struktur i ich wzajemnych powiązań.

1.2. Tworzenie aseracji zgodnie ze standardami SQL

Zgodnie z definicją asercji w standardzie SQL92, jako aseracją określono każdy warunek integralności zawierający sekwencję wyszukiwania [1].

Zgodnie z powyższym, aseracją można zdefiniować jako element schematu bazy danych za pomocą polecenia SQL w notacji BNF [3]:

<assertion definition> ::=
  CREATE ASSERTION <constraint name> <assertion check>
   [<constraint attributes>]
<assertion check> ::=
  CHECK
    <left paren> <search condition> <right paren>

Deklaracja postaci:

CREATE ASSERTION name CHECK condition);

mimo iż pozornie wydaje się prosta do zrozumienia i implementacji nie występuje w specyfikacji popularnych DBMS. Warto również wspomnieć, iż do dziś programiści dyskutują nad możliwością wprowadzenia asercji do takich systemów, jak Oracle czy PostgreSQL, wzorem zapomnianego obecnie DEC RDB (kontynuacja jako Oracle RDB dla OpenVMS).

W standardzie SQL3 [2] zmieniono oficjalną definicję określając aseracją jako ograniczenie integralności definiujące własny schemat nieistniejący w postaci definicji tabeli. W systemach Oracle asercja definiowana jako warunek wyszukiwania może

być implementowana m.in. w postaci obiektu sterowanego danymi za pomocą perspektyw materializowanych. Istotne jest również dodanie do standardu SQL3 wyzwalaczy sterowanych zdarzeniami. Wzorując się na wyzwalaczach, powyższa, nowa definicja asercji została rozszerzona o sterowanie zdarzeniami, co w istocie odpowiada funkcjonalności wyzwalaczy [4 i 5]. Taką sytuację pokazuje kod poniżej:

<SQL3 assertion> ::= CREATE ASSERTION <constraint name>
{BEFORE COMMIT | AFTER <assertion events>}
CHECK (<condition>)
[FOR [EACH ROW OF] <table name>]
<constraint evaluation>
<assertion events> ::= {INSERT | DELETE | UPDATE [OF <column names>]}
ON <table name>

Uproszczona klauzula tworzenia wyzwalaczy (Oracle):

<PL/SQL trigger> ::= CREATE [OR REPLACE] TRIGGER <trigger_name>
{BEFORE|AFTER} <trigger events>
[REFERENCING [NEW AS <new_row_name>]
[OLD AS <old_row_name>]]
[FOR EACH ROW [WHEN (<trigger_condition>)]]
<trigger_body>
<trigger events> ::= {INSERT | DELETE | UPDATE [ON <table_name>]}

Jak widać z przedstawionych klauzul do tworzenia asercji i wyzwalaczy, forma asercji proponowana w standardzie SQL3 jest praktycznie identyczna (choć bardziej ograniczona) niż np. wyzwalacze PL/SQL znane od 1992 roku (SQL92). Była to jedna z ważniejszych zmian w koncepcji *EDA* (**E**vent-**D**riven **A**rchitecture) [6]. Takie podejście zdradza częściowo niechęć producentów systemów bazodanowych do implantacji w swoich DBMS klauzuli *create assertion*, skoro wydaje się ona być powtórzeniem funkcjonalności dostępnej w postaci wyzwalaczy czy UDF.

1.3. Przykłady implementacji aseracji z wykorzystaniem technik standardowych

Często stosowanym sposobem tworzenia więzów typu *asercja* jest wykorzystanie klauzuli *check*. Wyróżnikiem asercji w tym przypadku jest zastosowanie zapytania jako warunku nastąpienia zdarzenia związanego ze spełnieniem lub niespełnieniem tejże asercji. Przekład realizacji asercji sprawdzającej, czy pracownik (EMP) o za niskich obrotach miesięcznych (*EMP_SALES*), tj. poniżej 10 tys., nie otrzymał premii (*BONUS*), przedstawia poniższy kod.

Dane są relacje:

EMP(EMP_ID (PK), EMP_SALES);
BONUS(BON_ID (PK), BON_AMOUNT, EMP_ID (FK));

oraz asercja:

CREATE ASSERTION ASSERT_Check_Bonus CHECK
(NOT EXISTS
*(SELECT * FROM EMP a, BONUS b*
WHERE a.EMP_ID = b.EMP_ID AND a. EMP_SALES < 10000));

Dla powyższej deklaracji możliwe jest utworzenie więzów typu *CHECK* mimo iż asercja w pierwotnej postaci odnosi się do dwóch relacji, tzn.:

```
CREATE TABLE BONUS (
    BON_ID number(8) NOT NULL PRIMARY KEY,
    BON_AMOUNT number(7,2) NOT NULL,
    EMP_ID number(8) NOT NULL
            INT REFERENCES EMP(EMP_ID),
            CHECK (EMP_ID NOT IN
              (SELECT EMP_ID FROM EMP WHERE EMP_SALES < 10000)));
```

Przedstawiony przykład wpisuje się w przytoczoną wcześniej, nową definicję asercji w standardzie SQL3.

Niestety, w systemach z rodziny Oracle próba wykonania powyższego kodu spowoduje wyświetlenie błędu:

```
ORA-02251: subquery not allowed here
```

Problem braku implementacji podzapytań dla więzów integralności wydaje się uciążliwy, bo nawet proste konstrukcje typu:

```
ALTER TABLE BONUS ADD CONSTRAINT CHK_EMP_NOT_EMPTY
CHECK ((SELECT COUNT(*) FROM EMP) > 0);
```

wymagają zbudowania znacznie poważniejszych, ale i bardziej elastycznych mechanizmów.

W następnej części artykułu przedstawione zostaną bardziej skomplikowane konstrukcje, opierające się na wyzwalaczach i perspektywach materializowanych.

2. Aseracje w systemach Oracle Database

2.1. Aseracje i UDF

Jak wcześniej wskazano, jednym z problemów w tworzeniu klasycznych asercji z użyciem więzów typu *CHECK* jest brak wsparcia zapytań. Wynika to z faktu, że tego typu konstrukcje mają charakter deklaratywny, a więc nie są znane wyniki wewnętrznych operacji na etapie ich tworzenia. Problem ten można jednak w bardzo prosty sposób rozwiązać. Dla niektórych systemów DBMS wystarczy zdefiniować funkcję typu UDF. Nazwa funkcji jest znana na etapie deklaracji więzów, a więc nie pojawi się błąd dla podzapytań. Przykład takiego mechanizmu pokazuje kolejny kod źródłowy.

```
CREATE FUNCTION COUNT_EMP
RETURN integer IS
  ret_val integer;
BEGIN
  select count(*) into ret_val from EMP;
  RETURN ret_val;
END;
```

```
ALTER TABLE BONUS ADD CONSTRAINT CHK_EMP_NOT_EMPTY
CHECK (COUNT_EMP() > 0);
```

Funkcja *COUNT_EMP* powinna być wykonywana na etapie wywołania więzów integralności podczas operacji *DML*. Należy zauważyć, że więzy *CHK_EMP_NOT_EMPTY* będą wykonywane dla wszystkich operacji *DML* (*INSERT*, *UPDATE*, *DELETE*), choć powinny być wykonywane wyłącznie dla operacji *INSERT* lub ewentualnie *UPDATE*.

Przedstawione podejście nie jest z kolei zgodne ze standardem SQL, gdyż miesza deklaratywny kod SQL z jego proceduralnym rozszerzeniem i o ile sprawdzi się w Microsoft T-SQL, to w systemie Oracle niestety nie. Nie posiadamy również informacji na temat rodzaju zdarzenia, jakim mają być sterowane takie więzy integralności, co w przypadku bardziej skomplikowanych konstrukcji może być istotną wadą.

Implementację omawianej asercji z wykorzystaniem wyzwalaczy w języku PL/SQL przedstawia poniższy kod:

```
CREATE OR REPLACE TRIGGER T_BD_CHK_EMP_NOT_EMPTY
BEFORE INSERT OR UPDATE ON BONUS
DECLARE
      ret_val integer :=0;
BEGIN
      select count(*) into ret_val from EMP;
      IF ((INSERTING OR UPDATING) AND (ret_val < 1))
      THEN
            RAISE_APPLICATION_ERROR(-20001, 'EMP is EMPTY!');
   END IF;
END;
```

Informacja o sposobie działania tak skonstruowanych więzów integralności jest dostępna zarówno z jego kodu źródłowego, jak i ze słownika danych.

W sposób jawny podawana jest informacja o rodzaju zdarzeń. Cały kod zamknięty jest w procedurze PL/SQL. Programista otrzymuje możliwość reagowania za pomocą klauzul warunkowych na konkretne zdarzenie, tj. w tym przypadku *IF INSERTING*, *IF UPDATING*. Tak zbudowany wyzwalacz wysyła komunikaty do aplikacji w warstwie logiki biznesowej czy prezentacji, które można w dowolny sposób obsłużyć, tworząc własną listę unikalnych wyjątków. Jak widać, możliwość konfiguracyjne ograniczone są w zasadzie wyobraźnią programisty i wymaganą funkcjonalnością.

2.2. Aseracje wykonujące kod DML

Implementacja asercji w sensie SQL3, tzn. mechanizmów kontrolnych sterowanych zdarzeniami, była i jest możliwa dzięki wykorzystaniu przedstawianych już wyzwalaczy. Ze względu na dowolne tworzenie funkcjonalności w kodzie PL/SQL możliwe jest budowanie mechanizmów do aktualizacji danych.

Dany jest schemat postaci:

```
KIND_EMP(KIND_ID,KIND_NAME, KIND_DESC,KIND_ASERT_COUNT);
EMP(EMP_ID,KIND_ID,EMP_NAME);
```

Relacja *KIND_EMP* przechowuje informacje na temat rodzajów pracowników, natomiast relacja *EMP* gromadzi dane na temat zatrudnionych. Atrybut *KIND_ASERT_COUNT* wskazuje na ilość pracowników w danej grupie.

Standardowy kod do aktualizacji wartości atrybutu *KIND_ASERT_COUNT* powinien wyglądać następująco:

```
UPDATE KIND_EMP SET KIND_ASERT_COUNT = (
                    SELECT COUNT(*) FROM EMP
                    WHERE EMP.KIND_ID=KIND_EMP.KIND_ID
                    );
```

Automatyczną aktualizację takiego kodu powinien gwarantować wyzwalacz reagujący na pełny zestaw zdarzeń *DML*, tzn. *INSERTING*, *UPDATING KIND_ID* oraz *DELETING* w obrębie relacji *EMP*.

```
CREATE OR REPLACE TRIGGER T_AIUD_EMP_ASERT
AFTER INSERT OR UPDATE OR DELETE
ON EMP FOR EACH ROW
BEGIN
    IF (INSERTING OR (UPDATING AND :NEW.KIND_ID <> :OLD.KIND_ID))
    THEN UPDATE KIND_EMP SET
KIND_ASERT_COUNT = KIND_ASERT_COUNT+1
WHERE KIND_ID = :NEW.KIND_ID;
    END IF;
    IF ((UPDATING AND :NEW.KIND_ID <> :OLD.KIND_ID) OR DELETING)
    THEN UPDATE KIND_EMP SET
KIND_ASERT_COUNT = KIND_ASERT_COUNT-1
WHERE KIND_ID = :OLD.KIND_ID;
    END IF;
END;
/
Trigger created.
```

Po prześledzeniu działania stworzonych więzów sterowanych zdarzeniami *DML*, można dojść do wniosku, że całość kodu jest poprawna i nie powinna nastręczać trudności w przyszłości.

```
ORA11GR2> select KIND_ID, KIND_NAME, KIND_ASERT_COUNT
from KIND_EMP
order by KIND_ID;

KIND_ID       KIND_NAME  KIND_ASERT_COUNT
------- ------------------ ----------------
    10   Java_programmer                20
    20    Cpp_Programmer                18
    30        Ora_DBA                    4
    40        IMS_DBA                    1
```

```
ORA11GR2> delete from EMP where KIND_ID = 30 and rownum = 1;
1 row deleted.

ORA11GR2> select KIND_ID, KIND_NAME, KIND_ASERT_COUNT
from KIND_EMP
order by KIND_ID;

KIND_ID       KIND_NAME  KIND_ASERT_COUNT
------- ------------------ ----------------
   10   Java_programmer          20
   20    Cpp_Programmer          18
   30        Ora_DBA             3        < - ZMIANA!
   40        IMS_DBA             1
ORA11GR2> commit;
```

Jak pokazano w przykładzie, wyzwalacz w prawidłowy sposób dokonuje modyfikacji danych w tabeli *KIND_EMP*. Nie brano niestety pod uwagę wszystkich więzów integralności, które powinny być lub są zdefiniowane dla relacji *EMP* i *KIND_EMP*. Narzucenie na kolumnę *KIND_ASERT_COUNT* więzów ograniczających możliwość zwolnienia ostatniego specjalisty danego typu może spowodować istotne ograniczenia w funkcjonowaniu bazy danych. Niestety narzucenie więzów typu:

```
alter table KIND_EMP add constraint CHK_KIND_ASERT_COUNT
(KIND_ASERT_COUNT>0) deferrable initially deferred;
```

może spowodować wystąpienie błędu ORA-02290: *check constraint violated* i wycofanie całej transakcji.

3. Wnioski

Stosunkowo nowym rozwiązaniem ukrywania danych na zasadach widoków są perspektywy materializowane. Niestety w przypadkach prób implementacji asercji z wykorzystaniem więzów typu *check* zarówno dla klauzuli *deferrable initially immediate*, jak i *deferrable initially deferred* [7] dla *CHK_KIND_ASERT_COUNT*, przedstawiony wcześniej problem będzie się powtarzał.

Jak pokazano, istotne zmiany w standardzie SQL3 [2] stanowiące o zwiększeniu funkcjonalności asercji przez ich zbliżenie do funkcjonalności wyzwalaczy nie zlikwidowało problemów z prawidłowym działaniem mechanizmu asercji. Sama klauzula *CREATE ASSERTION* zgodna z SQL3, jest ubogą implementacją wyzwalaczy (na przykład w modelu rozszerzonym O r a c l e).

Należy pamiętać, że mimo, iż funkcjonalność asercji i wyzwalaczy wydaje się podobna, to twórcy standardu SQL3 przewidzieli inny sposób funkcjonowania tych mechanizmów. Aseracje zgodne z przytaczanym standardem nie zostaną prawdopodobnie nigdy zaimplementowane. Wiele z zamierzonych więzów integralności można w prosty sposób zaimplementować dzięki mechanizmom typu *CHECK* rozszerzonym o możliwość stosowania funkcji użytkownika w formie *UDF*. Silnym narzędziem są również wyzwalacze, a ich stosowanie jest o tyle wygodne, że ich kod i sposób działania jest jawnie

dostępny w perspektywach słownika danych. Nieomówione tu mechanizmy perspektyw materializowanych oraz zewnętrznych funkcji sterujących, np. za pośrednictwem interfejsu Oracle Pro*C, to kolejne narzędzia, które oddalają konieczność stosowania mechanizmów asercji zgodnych ściśle ze standardem.

Literatura

[1] ISO/IEC 9075:1992, *Database Language SQL (revision of ISO/IEC 9075:1989)*.
[2] ISO/IEC 9075:1999, *Information technology – Database languages – SQL – Part 2: Foundation (SQL/Foundation)*.
[3] Melton J., *Understanding the New SQL: A Complete Guide*, Morgan Kaufmann, San Francisco 1993.
[4] Widom J., Ceri S., *Active database systems: triggers and rules for advanced database processing*, Morgan Kaufmann, San Francisco 1996.
[5] Knolmayer G., Herbst H., Schlesinger M., *Enforcing business rules by the application of trigger concepts*, Proceedings Priority Programme Informatics Research, Information Conference, Berne: Swiss National Science Foundation, 1994.
[6] Oracle Corp., *ORACLE: 15 Years Of Event Processing Leadership*, Oracle Corp. 2007.
[7] Tropashko V., *Complex Constraints*, DBAzine, 2005.