Триггеры

Триггеры - это хранимые процедуры, которые срабатывают при определенном событии, в основном при выполнении операций модификации данных **INSERT**, **UPDATE**, **DELETE**. Существуют так же системные триггеры, которые срабатывают на события самой БД. В основном триггеры используются для:

1. Реализации сложных ограничений целостности данных, которые невозможно осуществить при создании таблиц.
2. Организации всевозможных видов аудита. Например, слежения за изменениями в какой-либо таблице БД и автоматического оповещения других модулей о том, что делать в случае изменения информации, содержащейся в таблице БД.
3. Для реализации так называемых "бизнес правил".
4. Для организации каскадных воздействий на таблицы БД.

Существуют триггеры, которые срабатывают до момента срабатывания одного из операторов **INSERT**, **UPDATE**, **DELETE** (триггеры BEFORE), и после момента срабатывания одного из этих операторов (триггеры AFTER). Для демонстрации работы триггеров создадим две вспомогательные таблицы: ADT и COSTOMERS.

```
CREATE TABLE ADT
(
    USAL varchar (50),
    TISP date
);
CREATE TABLE CUSTOMERS
(
    cust_num integer,
    company varchar(50),
    cust_rep integer,
    credit_limit float
);
```

Теперь попробуем с помощью нее и нового триггера организовать некий аудит в системе доступа к таблице **CUSTOMERS**. Для этого создадим простой триггер:

```
CREATE TRIGGER TESTTRIGGER
AFTER INSERT OR DELETE OR UPDATE ON CUSTOMERS
DECLARE
BEGIN
    INSERT INTO ADT VALUES ('Фамилия', SYSDATE);
END TESTTRIGGER;
```

После создания триггера, его нужно включить:

```
ALTER TRIGGER TESTTRIGGER ENABLE;
```

После выполнения команды вставки данных в таблицу CUSTOMERS:

INSERT INTO CUSTOMERS (cust_num, company, cust_rep, credit_limit) VALUES (2200, 'MyCompany', 107, 555.5643);

изменилось содержимое таблицы ADT:

SELECT USAL, TO_CHAR(TISP,'DD.MM.YYYY') TISM
FROM ADT;

Видно, что в таблице ADT появилась новая запись.

Задание: Удалите запись в таблице CUSTOMERS и повторите запрос на вставку данных в CUSTOMERS, посмотрите что изменилось в ADT. Измените какую-либо запись в CUSTOMERS, посмотрите, что при этом изменится в таблице ADT.

Итак, при выполнении команды добавления, изменения или удаления данных каждый раз вызывается триггер TESTTRIGGER, который вставляет в таблицу ADT новую запись (в нашем примере).

Помимо деления триггеров на AFTER и BEFORE, существует также деление триггеров на операторный триггер и строковый. Порядок активации триггеров в большинстве случаев таков:

1. Выполняется операторный триггер **BEFORE** (при его наличии)
2. Для каждой строки, на которую воздействует оператор:
    2.1. Выполняется строковый триггер **BEFORE** (при его наличии).
    2.2. Выполняется собственно оператор.
    2.3. Выполняется строковый триггер **AFTER** (при его наличии).
3. Выполняется операторный триггер **AFTER** (при его наличии).

Чтобы четко представлять, всю картину работы триггеров таблиц БД создадим пару таблиц и наполним их данными:

```
CREATE TABLE TSTTRIG
(
ID NUMBER PRIMARY KEY,
NM VARCHAR(50),
ROD VARCHAR(50),
INRW DATE
);
INSERT INTO TSTTRIG VALUES (7369, 'SMITH',  'CLERK',
TO_DATE('17-2-2000', 'DD-MM-YYYY'));
INSERT INTO TSTTRIG VALUES (7370, 'JONES',  'MANAGER',
TO_DATE('2-4-2001', 'DD-MM-YYYY'));
INSERT INTO TSTTRIG VALUES (7371, 'MILLER', 'SALESMAN',
TO_DATE('20-3-2003', 'DD-MM-YYYY'));
INSERT INTO TSTTRIG VALUES (7372, 'SCOTT',  'ANALYST',
TO_DATE('09-12-2001', 'DD-MM-YYYY'))
```

Создадим еще одну таблицу:

```
CREATE TABLE TSTSV
(
ID NUMBER PRIMARY KEY,
IDD VARCHAR(50),
ROD VARCHAR(50),
CONS NUMBER
);
INSERT INTO TSTSV VALUES (1, 'SMITH',  'CLERK', 7369);
INSERT INTO TSTSV VALUES (2, 'JONES',  'MANAGER', 7370);
INSERT INTO TSTSV VALUES (3, 'MILLER', 'SALESMAN', 7371);
INSERT INTO TSTSV VALUES (4, 'SCOTT',  'ANALYST', 7372);
```

Строим зависимость таблицы **TSTSV** от таблицы **TSTTRIG** по полям **CONS** и **ID** соответственно:

```
ALTER TABLE TSTSV ADD FOREIGN KEY (CONS)
REFERENCES TSTTRIG (ID);
```

Видоизменим таблицу **ADT**, добавив в нее еще два поля:

```
ALTER TABLE ADT ADD WDO VARCHAR(50);
ALTER TABLE ADT ADD PRIM VARCHAR(50);
```

Создадим для таблицы **TSTTRIG** два операторных триггера по одному на каждое из двух действий **BEFORE** и **AFTER**, срабатывание которого установим на событие **UPDATE**.

```
CREATE OR REPLACE TRIGGER BFOTST
BEFORE UPDATE ON TSTTRIG
DECLARE
BEGIN
    INSERT INTO ADT VALUES(‘BFOTST операторный‘,
SYSDATE, 'Update', 'Before Statement trigger');
END BFOTST;

ALTER TRIGGER BFOTST ENABLE;


CREATE OR REPLACE TRIGGER AFTTST
AFTER UPDATE ON TSTTRIG
DECLARE
BEGIN
    INSERT INTO ADT VALUES ('AFTTST операторный ',
SYSDATE, 'Update', 'After Statement trigger');
END AFTTST;

ALTER TRIGGER AFTTST ENABLE;
```

Операторные триггеры созданы. Теперь создадим строковые по тому же принципу, но отличаться они будут наличием конструкции **FOR EACH ROW**.

```
CREATE OR REPLACE TRIGGER BFOTSTR
BEFORE UPDATE ON TSTTRIG FOR EACH ROW
DECLARE
BEGIN
    INSERT INTO ADT VALUES ('BFOTSTR строковый', SYSDATE, 'Update',
'Before Row trigger');
END BFOTSTR;
ALTER TRIGGER BFOTSTR ENABLE;

CREATE OR REPLACE TRIGGER AFTTSTR
AFTER UPDATE ON TSTTRIG FOR EACH ROW
DECLARE
BEGIN
    INSERT INTO ADT VALUES('AFTTSTR строковый', SYSDATE, 'Update',
        'After Row trigger');
END AFTTSTR;
ALTER TRIGGER AFTTSTR ENABLE;
```

Почистим таблицу ADT:

```
DELETE FROM ADT;
```

А вот теперь попробуем ввести такой оператор, для таблички **TSTTRIG**:

```
UPDATE TSTTRIG SET ROD = 'SPOOKY' WHERE ID IN (7369, 7370) ;
```

Должно быть изменено две строки, так как условие оператора **UPDATE** соответствует в нашем случае двум записям.

```
SELECT * FROM ADT;
```

В результате имеем 6 записей. Операторный триггер сработал только два раза на **BEFORE** и **AFTER**, а вот строковый четыре раза!

Вернем полям таблицы **TSTTRIG** прежние значения и очистим таблицу **ADT**:

```
UPDATE TSTTRIG SET ROD = 'CLERK' WHERE ID = 7369;
UPDATE TSTTRIG SET ROD = 'MANAGER' WHERE ID = 7370;
DELETE FROM ADT;
```

Реализуем добавление записи сразу в две таблицы БД. Для работы нам понадобиться создать последовательность – объект базы данных, с помощью которого можно генерировать значения первичных ключей таблицы:

```
CREATE SEQUENCE SV
START WITH 5
INCREMENT BY 1;
```

Создадим триггер, для копирования данных во 2-ю таблицу:

```
CREATE OR REPLACE TRIGGER AFTINSTTRIG
AFTER INSERT ON TSTTRIG FOR EACH ROW
DECLARE
BEGIN
    INSERT INTO TSTSV (ID, IDD, ROD, CONS)
    VALUES (SV.NEXTVAL, :NEW.NM, :NEW.ROD, :NEW.ID);
END AFTINSTTRIG;

ALTER TRIGGER AFTINSTTRIG ENABLE;
```

Триггер **AFTINSTTRIG** срабатывает после вставки в таблицу **TSTTRIG**.

Выполним оператор добавления данных в таблицу **TSTTRIG** .

```
INSERT INTO TSTTRIG VALUES (8001, 'BLAKE',
'MANAGER', TO_DATE('8-5-1999', 'DD-MM-YYYY'));
```

Смотрим содержимое таблиц **TSTTRIG** и **TSTSV**.

```
SELECT * FROM TSTTRIG;
SELECT * FROM TSTSV;
```

Видно, что новая строчка появилась в обеих таблицах.

Таким образом один **INSERT** работает на две таблицы. Триггер типа **AFTER**, т.к. поле **:NEW.ID** определяется после вставки!

Теперь создадим триггер для автоматического изменения данных второй таблицы при изменении первой.

```
CREATE OR REPLACE TRIGGER AFTUPDTTRIG
AFTER UPDATE ON TSTTRIG FOR EACH ROW DECLARE
BEGIN
    UPDATE TSTSV
    SET
      TSTSV.IDD = :NEW.NM,
      TSTSV.ROD = :NEW.ROD,
      TSTSV.CONS = :NEW.ID
    WHERE TSTSV.CONS = :OLD.ID;
END AFTUPDTTRIG;

ALTER TRIGGER AFTUPDTTRIG ENABLE;
```

Данный триггер следит за изменением полей **NM**, **ROD**, **ID** в таблице **TSTTRIG** и отражает изменения в таблице **TSTSV**. Например, запрос:

```
UPDATE TSTTRIG
SET
ROD = 'SPOOKY'
WHERE ID IN (7369, 7370);
```

Изменит по две строчки в двух таблицах (**TSTTRIG** и **TSTSV**).

```
SELECT * FROM TSTTRIG;
SELECT * FROM TSTSV;
```

Теперь выполним обработку удаления соответствующей строчки из таблицы **TSTSV** при удалении из **TSTTRIG**.

```
CREATE OR REPLACE TRIGGER BFRDELTTRIG
BEFORE DELETE ON TSTTRIG FOR EACH ROW
DECLARE
BEGIN
   DELETE FROM TSTSV
   WHERE TSTSV.CONS = :OLD.ID;
END BFRDELTTRIG;

ALTER TRIGGER BFRDELTTRIG ENABLE;
```

Здесь используется **BEFORE** (то есть «перед тем как удалить, убери за собой»). Смотрим как он работает.

```
DELETE FROM TSTTRIG
WHERE ID = 8001;
```

Удалена строчка данных не только из таблицы TSTTRIG, но и из таблицы TSTSV. Проверьте это.

Заметим, что в строковом триггере **BEFORE** можно менять псевдозапись **:new**. Менять псевдозапись **:new** в строковом триггере **AFTER** не имеет смысла, так как событие уже обработано. А вот псевдозапись **:old** никогда не модифицируется, а только считывается.

Условие WHERE в триггере

С помощью **WHERE** можно заставить триггер работать по условию! Само условие **WHERE** в триггере применимо к типу строчных триггеров. Рассмотрим, как это реализуется на практике. Предварительно добавим по одному полю в наши таблички из прошлых шагов следующим образом:

```
ALTER TABLE TSTTRIG ADD COST NUMBER;
ALTER TABLE TSTSV ADD ITOG NUMBER;
```

В этих полях будут хранится данные для последующего использования. Добавим данные в таблицу **TSTTRIG**, с помощью операторов **UPDATE**:

```
UPDATE TSTTRIG SET COST = 30532
WHERE NM = 'SMITH‘;

UPDATE TSTTRIG SET COST = 80478
WHERE NM = 'JONES‘;

UPDATE TSTTRIG SET COST = 20785
WHERE NM = 'MILLER‘;

UPDATE TSTTRIG SET COST = 10205
WHERE NM = 'SCOTT‘;
```

Создадим триггер с условием:

```
CREATE OR REPLACE TRIGGER WHENTRG
BEFORE INSERT OR UPDATE OF COST ON TSTTRIG
FOR EACH ROW
WHEN (new.COST > 10000)
DECLARE
BEGIN
    UPDATE TSTSV
    SET TSTSV.ITOG = :new.COST + :old.COST
    WHERE TSTSV.CONS = :old.ID;
END WHENTRG;

ALTER TRIGGER WHENTRG ENABLE;
```

Строка **OF COST ON TSTTRIG** определяет поле, на которое устанавливает условие триггера и собственно само условие **WHEN (new.COST > 10000)** - псевдозапись **new** записана как - **new**, а не **:new** ! Это важно! В условии псевдозаписи записываются БЕЗ ДВОЕТОЧИЯ! Двоеточие применяется только в теле триггера. Условие можно строить и по другому, так как нужно вам.

Данный триггер производит довольно глупое, но зато наглядное действие - при вставке или изменении, он запоминает сумму чисел в поле **ITOG** таблички **TSTSV**.

Изменим одну из записей таблички **TSTTRIG** (запомнив, что ее старое содержимое было равно 10205):

```
UPDATE TSTTRIG
SET COST = 20205
WHERE NM = 'SCOTT';
```

Посмотрим, что изменилось в полях наших с вами таблиц:

```
SELECT * FROM TSTTRIG;
SELECT * FROM TSTSV;
```

В столбце ITOG было записано число 30410 = 10205 + 20205, т.е. условие дало **TRUE** и триггер сработал.

Выполним другой запрос:

```
UPDATE TSTTRIG
SET COST = 5342
WHERE NM = 'MILLER';

SELECT * FROM TSTTRIG;
SELECT * FROM TSTSV;
```

В таблицах ничего не изменилось, поскольку условие 5342 > 10000 не выполнено, и триггер не сработал.

Добавим новую строчку в таблицу **TSTTRIG**:

```
INSERT INTO TSTTRIG VALUES (8001, 'BLAKE',
'MANAGER', TO_DATE('8-5-1999', 'DD-MM-YYYY'));
```

Запись **(8001, BOB, DUMMY, 09.11.1989, 24734)** - добавлена в таблицу **TSTTRIG**. Триггеры из прошлых шагов так же добавили запись - **(5, BOB, DUMMY, 8000, NULL)**. Но почему поле **ITOG** таблицы **TSTSV** содержит **NULL**? Ведь условие 24734 > 10000 дает **TRUE**! Дело в том, что триггер **WHENTRG** свою работу, но псевдозапись **:new** для триггеров по **INSERT** не определена, то есть содержит **NULL**! Отсюда **NULL + 24734 = NULL**!

Предикаты

В триггерах БД **Oracle** возможно применение предикатов - логических операторов. Они имеют следующие определения: **INSERTING**, **UPDATING**, **DELETING**. Это некие внутренние переменные среды **Oracle**, которые в зависимости от воздействующего на таблицу оператора принимают одно из значений: **TRUE** или **FALSE**. С их помощью можно значительно сэкономить время при написании кода.

Предикат	Принимаемое значение
INSERTING	TRUE если, активизирующий оператор INSERT. FALSE в противном случае.
UPDATING	TRUE если, активизирующий оператор UPDATE. FALSE в противном случае.
DELETING	TRUE если, активизирующий оператор DELETE. FALSE в противном случае.

Создадим таблицу **MYAUDIT**:

```
CREATE TABLE MYAUDIT
(
POLZ VARCHAR(15),
VIZM DATE,
OPER VARCHAR(20),
NZAP NUMBER,
HIST VARCHAR(50)
);
```

Данные в этой таблице будут меняться во время выполнения нашего примера. Применим на практике предикаты **INSERTING**, **UPDATING**, **DELETING** для написания триггера вида:

```
CREATE OR REPLACE TRIGGER AUDT_TSTTRIG
BEFORE INSERT OR UPDATE OR DELETE ON TSTTRIG FOR
EACH ROW
DECLARE
   TIP VARCHAR(10);
BEGIN
   IF INSERTING THEN
      TIP := 'INSERT';
   ELSIF UPDATING THEN
      TIP := 'UPDATE';
   ELSIF DELETING THEN
      TIP := 'DELETE';
   END IF;
   INSERT INTO MYAUDIT VALUES (USER, SYSDATE, TIP,
:new.ID,
      'Old Name: '||:old.NM||' New Name: '||:new.NM);
END AUDT_TSTTRIG;
```

Данный триггер имеет временное действие "ДО"!
Попробуем добавить запись в таблицу **TSTTRIG**:

```
INSERT INTO TSTTRIG VALUES (8002, 'ALFRED',
'MANAGER', TO_DATE('18-12-2002', 'DD-MM-YYYY'),
40967);
```

Проверим, что в таблице MYAUDIT появилась строчка:

```
SELECT * FROM MYAUDIT;
```

Нужно запомнить,
что псевдозапись **:old** для оператора **INSERT** триггера типа **BEFORE** не определена! А псевдозапись **:new** для поля **ID**, так же еще не получила значения!

Изменим запись в таблице TSTTRIG следующим образом:

```
UPDATE TSTTRIG
SET NM = 'ALF'
WHERE NM = 'ALFRED';
```

Проверим таблицу **MYAUDIT**:

```
SELECT * FROM MYAUDIT;
```

Альфред стал Альфом! Здесь строка **Old Name: ALFRED New Name: ALF** показывает, что псевдозаписи **:new** и **:old** применительно для оператора **UPDATE**, для триггера типа **BEFORE** определены!
Псевдозаписи **:new** так же можно изменить.

Попробуем удалить запись:

```
DELETE FROM TSTTRIG
WHERE NM = 'ALF';
```

Проверим таблицу **MYAUDIT**:

```
SELECT * FROM MYAUDIT;
```

Здесь строка **Old Name: ALF New Name: "NULL"** показывает, что псевдозаписи **:new** применительно для оператора **DELETE**, для триггера типа **BEFORE** не определена! Здесь, для поля **NZAP** получаем **NULL**. В данном случае в поле **OPER** после сработки условия **.. IF INSERTING THEN ..** получаем, что была операция **DELETE**.