Триггеры баз данных

Понятие триггера

ТРИГГЕР БАЗЫ ДАННЫХ - это хранимая программная единица PL/SQL, ассоциированная с конкретной таблицей базы данных.

В отличие от подпрограмм, которые должны вызываться явно, триггер базы данных вызывается неявно и не имеет атрибутов.

Акт выполнения триггера называется его *активизацией* (firing). Событием, запускающим триггер, является операция DML (INSERT, UPDATE или DELETE), выполняемая над таблицей или представлением базы данных. В Oracle8i эти функции расширены: триггер может срабатывать на системное событие, например на запуск или останов базы данных, а также на определенные виды операций DDL.

Триггеры данных используются для

- контроля за информацией, хранимой в таблице, посредством регистрации вносимых изменений и пользователей, производящих эти изменения.
- реализации сложных ограничений целостности данных, которые невозможно реализовать через декларативные ограничения, устанавливаемые при создании таблицы.
- автоматического оповещения других программ о том, что необходимо делать в случае изменения информации, содержащейся в таблице.
- осуществления сложных процедур защиты;
- поддержки дублированных таблиц.

Типы триггеров

Триггер DML активизируется оператором DML, и тип триггера определяется типом этого оператора. Триггеры DML задаются для операций ввода, обновления и удаления информации (INSERT, UPDATE, DELETE). Они активизируются до или после операции, на уровне строки или оператора.

В Oracle8i предлагается еще один вид триггеров. *Триггеры замещения* (instead of) можно создавать только для представлений (либо объектных, либо реляционных). В отличие от триггеров DML, которые выполняются в дополнение к операторам DML, триггеры замещения выполняются вместо операторов DML, вызывающих их срабатывание. Триггеры замещения должны быть строковыми триггерами.

В Orac1e8i и выше существует третий тип триггеров. *Системный триггер* активизируется не на операцию DML, выполняемую над таблицей, а на системное событие, например, на запуск или останов базы данных. Системные триггеры срабатывают и на операции DDL, такие как создание таблицы.

Привилегии для создания триггера

Для создания триггера базы данных необходимо иметь привилегии CREATE TRIGGER, а также либо владеть ассоциированной таблицей, либо иметь привилегии ALTER для ассоциированной таблицы, либо иметь привилегии ALTER ANY TABLE.

Триггер базы данных состоит из трех частей: события триггера, необязательного ограничения триггера и действия триггера. Когда происходит событие триггера, триггер базы данных возбуждается, и анонимный блок PL/SQL выполняет предписанное действие. Триггеры базы данных возбуждаются с привилегиями владельца, а не текущего пользователя. Поэтому владелец должен иметь должный доступ ко всем объектам, вовлекаемым в действие триггера.

Создание триггера

CREATE [OR REPLACE] TRIGGER ***имя_триггера***

[BEFORE | AFTER | INSTEAD OF] ***активизирующее_событие***
ссылочное_предложение

[WHEN ***условие_срабатывания*****]**
[FOR EACH ROW]
тело_триггера;

Внимание! Тело триггера не может превышать 32 Кбайт. Если триггер больше, то его следует уменьшить, перенеся часть программного текста в отдельно компилируемые модули или хранимые процедуры и вызывая их в теле триггера. Ограничение размера тела триггеров обусловлено частотой их выполнения.

Создание триггера DML

Триггер DML активизируется операцией INSERT (ввод), UPDATE (обновление) или DELETE (удаление), выполняемой над таблицей базы данных. Триггеры могут активизироваться до (BEFORE) или после (AFTER) операции и действовать на уровне строки или оператора. Тип триггера определяется комбинацией этих факторов. Существует 12 возможных видов: 3 оператора x 2 момента времени x 2 уровня. Ниже приведены примеры правильных триггеров DML:

- До выполнения операции обновления на операторном уровне
- После выполнения операции ввода на уровне строк
- До выполнения операции удаления на уровне строк

Кроме того, триггер может активизироваться несколькими типами операторов DML, выполняемых над конкретной таблицей: например, INSERT и UPDATE. Код триггера выполняется вместе с активизирующим оператором как часть одной транзакции.

Виды триггеров DML

Категория	Значение	Комментарии
Оператор	INSERT, DELETE или UPDATE	Определяет, какой оператор DML активизирует триггер.
Момент времени	BEFORE или AFTER	Определяет момент активизации триггера: до или после выполнения оператора.
Уровень	Строка или оператор	Если триггер является строковым, то он активизируется один раз для каждой из строк, на которые воздействует оператор, вызывающий срабатывание триггера. Если триггер является операторным, то он активизируется один раз до или после оператора. Строковые триггеры идентифицируются предложением FOR EACH ROW (для каждой строки) в описании триггера.

Порядок активизации триггеров DML

Триггеры активизируются при выполнении оператора DML.

Алгоритм выполнения оператора DML таков:

1. Выполняются операторные триггеры BEFORE (при их наличии).
2. Для каждой строки, на которую воздействует оператор:
   A. Выполняются строковые триггеры BEFORE (при их наличии).
   B. Выполняется собственно оператор.
   C. Выполняются строковые триггеры AFTER (при их наличии).
3. Выполняются операторные триггеры AFTER (при их наличии).

Создание триггера DML. Пример 1.

```
CREATE SEQUENCE trig_seq
    START WITH 1
    INCREMENT BY 1;

CREATE OR REPLACE PACKAGE TrigPackage AS
    - Глобальный счетчик для использования в триггерах
    v_Counter NUMBER;
END TrigPackage;

CREATE OR REPLACE TRIGGER classesBStatement
    BEFORE UPDATE ON classes
BEGIN
    - Сначала сбросим счетчик.
    TrigPackage.v_Counter := 0;
    INSERT INTO temp_table (num_col, char_col)
       VALUES (trig_seq.NEXTVAL,
       'Before Statement: counter = ' | |
TrigPackage.v_Counter);
    - А теперь увеличим его значение для следующего
триггера.
    TrigPackage.v_Counter := TrigPackage.v_Counter+ 1;
END ClassesBStatement;
```

Создание триггера DML. Пример 1(продолжение)

```
CREATE OR REPLACE TRIGGER ClassesAStatementl
   AFTER UPDATE ON classes
BEGIN
   INSERT INTO temp_table (num_col. char_col)
      VALUES (trig'_seq.NEXTVAL,
      'After Statement 1: counter = ' |
TrigPackage.v_Counter);
   - Увеличим для следующего триггера.
   TrigPackage.v_Counter := TrigPackage.v_Counter+ 1;
END ClassesAStatementl;

CREATE OR REPLACE TRIGGER ClassesAStatement2
   AFTER UPDATE ON classes
BEGIN
   INSERT INTO temp_table (num_col, char_col)
      VALUES (trig_seq.NEXTVAL,
      'After Statement 2: counter = ' ||
TrigPackage.v_Counter);
   - Увеличим для следующего триггера.
   TrigPackage.v_Counter := TrigPackage.v_Counter+ 1;
END ClassesAStatement2;
```

Создание триггера DML. Пример 1(продолжение)

```
CREATE OR REPLACE TRIGGER ClassesBRowl
   BEFORE UPDATE ON classes
   FOR EACH ROW
BEGIN
   INSERT INTO temp_table (num_col, char_col)
      VALUES (trig_seq.NEXTVAL,
      'Before Row 1: counter = ' ||
TrigPackage.v_Counter);
   - Увеличим для следующего триггера.
   TrigPackage.vJDounter := TrigPackage.v_Counter+ 1;
END ClassesBRowl;

CREATE OR REPLACE TRIGGER ClassesBRow2
   BEFORE UPDATE ON classes
   FOR EACH ROW
BEGIN
   INSERT INTO temp_table (num_col, char__col)
      VALUES (trig_seq.NEXTVAL,
      'Before Row 2: counter = ' |
TrigPackage.v_Counter);
   - Увеличим для следующего триггера.
   TrigPackage.v_Counter := TrigPackage.v_Counter+ 1;
END ClassesBRow2;
```

Создание триггера DML. Пример 1(продолжение)

```
CREATE OR REPLACE TRIGGER ClassesBRowS
   BEFORE UPDATE ON classes
   FOR EACH ROW
BEGIN
   INSERT INTO temp_table (num_col, char_col)
      VALUES (trig_seq.NEXTVAL,
      'Before Row 3: counter = ' ||
TrigPackage.v_Counter);
   - Увеличим для следующего триггера.
   TrigPackage.v_Counter := TrigPackage,v_Counter+ 1;
END ClassesBRowS;

CREATE OR REPLACE TRIGGER ClassesARow
   AFTER UPDATE ON classes
   FOR EACH ROW
BEGIN
   INSERT INTO temp_table (num_col, char_col)
   VALUES (trig_seq.NEXTVAL,
      'After Row: counter = ' || TrigPackage.v_Counter);
   - Увеличим для следующего триггера.
   TrigPackage.v_Counter := TrigPackage.v_Counter+ 1;
END ClassesARow;
```

Создание триггера DML. Пример 1(продолжение)

Выполним следующий оператор UPDATE:

```
UPDATE classes
    SET num_credits = 4
    WHERE department IN ('HIS', ' C S ' ) ;
```

Этот оператор воздействует на четыре строки. Операторные триггеры BEFORE и AFTER выполняются по разу, а строковые триггеры BEFORE и AFTER — по четыре раза.

При активизации каждого из триггеров будут видны изменения, сделанные предыдущими триггерами, а также изменения в базе данных, внесенные оператором. Порядок, в котором активизируются триггеры одного вида, не определен. Из приведенного примера следует, что каждый триггер видит изменения, вносимые более ранними триггерами. Если порядок важен, следует объединить все операции в один триггер.

Создание триггера DML. Пример 1(продолжение)

```
SQL> SELECT * FROM temp_table
  2 ORDER BY num_col;
```

NUM_COL	CHAR_COL
1	Before Statement: counter = 0
2	Before Row 3: counter = 1
3	Before Row 2: counter = 2
4	Before Row 1: counter = 3
5	After Row : counter = 4
6	Before Row 3: counter = 5
7	Before Row 2: counter = 6
8	Before Row 1: counter = 7
9	After Row : counter = 8
10	Before Row 3: counter = 9
11	Before Row 2: counter = 10
12	Before Row 1: counter = 11
13	After Row : counter = 12
14	Before Row 3: counter = 13
15	Before Row 2: counter = 14
16	Before Row 1: counter = 15
17	After Row : counter = 16
18	After Statement 2: counter = 17
19	After Statement 1: counter = 18

Идентификаторы корреляции : old и : new

Строковый триггер запускается один раз для каждой строки, обрабатываемой активизирующим оператором. Внутри триггера можно обращаться к данным строки, обрабатываемой в данный момент. Для этого служат два идентификатора корреляции — *:old* и *:new*.

Идентификатор корреляции (correlation identifier) — это переменная привязки PL/SQL особого рода.

Двоеточие перед идентификатором указывает на то, что это переменные привязки (подобны базовым переменным, используемым во встроенном PL/SQL), а не обычные переменные PL/ SQL. Компилятор PL/ SQL рассматривает их как записи типа *активизирующая_таблица%ROWTYPE,* где *активизирующая_таблица* — это таблица, для которой создан триггер. Следовательно, ссылка типа *:new.поле* будет достоверна, если только поле является полем активизирующей таблицы.

Идентификаторы корреляции : parent

В Oracle8i определен еще один идентификатор корреляции — *:parent*. Если триггер создается для вложенной таблицы, *:old* и :new ссылаются на ее строки, а *:parent* — на текущую строку родительской таблицы

Активизирующий оператор	**:old**	**:new**
INSERT	Не определено — во всех полях содержится NULL	Значения, которые будут введены после выполнения оператора
UPDATE	Исходные значения, содержащиеся в строке перед обновлением данных	Новые значения, которые будут введены после выполнения оператора
DELETE	Исходные значения, содержащиеся в строке перед ее удалением	Не определено — во всех полях содержится NULL

Идентификаторы корреляции. Пример.

```
CREATE OR REPLACE TRIGGER GenerateStudentID
    BEFORE INSERT OR UPDATE ON students
    FOR EACH ROW
BEGIN
    /* Заполним поле ID таблицы students следующим значением из
    student_sequence. Поскольку ID - это столбец таблицы students,
    :new.ID является допустимой ссылкой. */
    SELECT student_sequence.NEXTVAL
    INTO :new.ID
    FROM dual;
END GenerateStudentID;
```

GenerateNewStudentID модифицирует значение *:new*. ID. Это одно из полезных свойств *:new* — когда выполнение оператора завершается, используются те значения, которые содержатся в *:new*. Нельзя изменить *:new* в строковом триггере AFTER, так как оператор будет обработан раньше. Вообще говоря, *:new* модифицируется только в строковых триггерах BEFORE; *:old* никогда не модифицируется, а лишь считывается.

Псевдозаписи

Хотя *:new* и *:old* синтаксически рассматриваются в качестве записей типа *активитрующая_таблица%ROWTYPE,* в действительности они записями не являются. Поэтому операции, применимые к записям, не могут быть выполнены над *:new* и *:old*. Например, эти псевдозаписи нельзя присваивать чему-либо в качестве целых записей.

Записи *:new* и *:old* разрешается использовать только в строковых триггерах. Если указать какую-либо из них в операторном триггере, будет выдана ошибка компиляции.

Поскольку операторный триггер выполняется лишь однажды (даже в том случае, когда в операторе обрабатывается несколько строк), псевдозаписи *:old* и *:new* не имеют никакого смысла.

Конструкция REFERENCING

При желании можно воспользоваться конструкцией REFERENCING и указать другие имена для *:old* и *:new*. Эта конструкция размещается после активизирующего события, перед условием WHEN:

REFERENCING [OLD AS *старое_имя*] [NEW AS *новое_имя*]

В теле триггера вместо *:old* и *:new* можно использовать *:старое_имя* и *:новое_имя*. Отметим, что в предложении REFERENCING идентификаторы указываются без двоеточия.

Конструкция REFERENCING. Пример.

```
CREATE OR REPLACE TRIGGER GenerateStudentID
   BEFORE INSERT OR UPDATE ON students
   REFERENCING new AS new _student
   FOR EACH ROW
BEGIN
   /* Заполним поле ID таблицы students
следующим   значением из student_sequence.
Поскольку ID - это столбец таблицы students,
:new_student.ID является допустимой  ссылкой.
*/
   SELECT student_sequence.NEXTVAL
   INTO :new_student.ID
   FROM dual;
END GenerateStudentID;
```

Предложение WHEN

Предложение WHEN можно использовать только для строковых триггеров. При наличии WHEN тело триггера будет выполняться только для тех строк, которые соответствуют условию, указанному в WHEN. Общий вид предложения WHEN таков

WHEN *условие_триггера*

где *условие_триггера* является логическим выражением, которое проверяется для каждой строки. В условии можно ссылаться на записи *:new* и *:old*, но двоеточие в данном случае не применяется. Двоеточие можно указывать только в теле триггера.

Предложение WHEN. Пример.

```
CREATE OR REPLACE TRIGGER CheckCredits
      BEFORE INSERT OR UPDATE OF current_credits ON students
      FOR EACH ROW
      WHEN (new.current_credits> 20)
BEGIN
      /* Тело триггера */
END;
```

Триггер CheckCredits можно также написать следующим образом:

```
CREATE OR REPLACE TRIGGER CheckCredits
      BEFORE INSERT OR UPDATE OF current_credits ON Students
      FOR EACH ROW
BEGIN
      IF :new.current_credits > 20 THEN
      /* Тело триггера */
      END IF;
END;
```

Триггерные предикаты: INSERTING, UPDATING и DELETING

Приведенный выше триггер UpdateMajorStats является триггером INSERT, UPDATE и DELETE. Внутри триггера такого типа (который срабатывает на различные виды операторов DML) можно использовать три логические функции, определяющие тип выполняемой операции. Это логические функции (предикаты) INSERTING, UPDATING и DELETING. Их работа описывается в таблице ниже.

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.

Триггерные предикаты. Пример.

```
CREATE TABLE RS_audit (
    change_type CHAR(1) NOT NULL,
    changed_by VARCHAR2(8) HOT NULL,
    timestamp DATE NOT NULL,
    old_student_id NUMBER(5),
    old_department CHAR(3),
    old_course NUMBER(3),
    old_grade CHAR(1),
    new_student_id NUMBER(5),
    new_department CHAR(3),
    new_course NUMBER(3),
    new_grade CHAR(1)
);
```

Триггерные предикаты. Пример (продолжение).

```
CREATE OR REPLACE TRIGGER LogRSChanges
      BEFORE INSERT OR DELETE OR UPDATE ON registered_students
      FOR EACH ROW
DECLARE
      v_ChangeType CHAR(1);
BEGIN
      /* Используем 'I' для INSERT, '0' для DELETE и 'IT для UPDATE. */
      IF INSERTING THEN
            v_ChangeType : = ' I';
      ELSIF UPDATING THEN
            v_ChangeType := ' U ' ;
      ELSE
            v_ChangeType := ' D ' ;
      END IF;
      /* Запишем в таблицу RS_audit все изменения, внесенные в таблицу
      registered_students. Для генерирования временной метки
      воспользуемся функцией SYSDATE, а для получения идентификатора
      текущего пользователя - функцией USER. */
      INSERT INTO RS_audit
            (changeType, changed_by, timestamp,
            old_student_id, old_department, old_course, old_grade,
            new_student_id, new_department, new_course, new_grade)
      VALUES
            (v_ChangeType, USER, SYSDATE,
            :old.student_id, :old.department, :old. course, :old.grade,
            :new. student_id, :new.department, :new. course, :new.grade);
END LogRSChanges;
```

Создание триггера DML. Пример 2.

Пример иллюстрирует прозрачную журнализацию событий. Триггер базы данных с именем reorder обеспечивает, что товар заказывается заново каждый раз, когда его имеющееся на складе количество (qty_on_hand) падает ниже пороговой точки.

```
CREATE TRIGGER reorder
   /* событие триггера */
   AFTER UPDATE OF qty_on_hand ON inventory  -- таблица
   FOR EACH ROW
   /* ограничение триггера */
   WHEN (new.reorderable = 'T')
BEGIN
   /* действие триггера */
   IF :new.qty_on_hand < :new.reorder_point THEN
      INSERT INTO pending_orders
      VALUES (:new.part_no, :new.reorder_qty, SYSDATE);
   END IF;
END;
```

Создание триггера. Пример 2. (продолжение)

- Имя во фразе ON (в примере, inventory) идентифицирует таблицу базы данных, ассоциированную с триггером базы данных.
- Событие триггера специфицирует предложение манипулирования данными SQL, которое воздействует на таблицу. В данном случае это предложение UPDATE. Если предложение триггера сбивается, оно откатывается.
- По умолчанию, триггер базы данных возбуждается один раз на всю таблицу. Необязательная фраза FOR EACH ROW указывает, что триггер должен возбуждаться один раз на каждую строку.
- Для того, чтобы триггер возбудился, однако, требуется, чтобы булевское выражение в фразе WHEN давало значение TRUE.
- Ключевое слово AFTER указывает, что триггер базы данных возбуждается после того, как обновление выполнено.
- Префикс :new представляет собой коррелирующее имя, которое отсылает к вновь измененному значению столбца. Внутри триггера базы данных вы можете обращаться как к новому, так и к старому (:old) значениям столбцов в измененных строках. Заметьте, что в фразе WHEN двоеточие не используется. Вы можете использовать фразу REFERENCING (здесь не показана), чтобы заменить :new и :old другими коррелирующими именами.

Создание триггера DML. Пример 3.

Как показывает следующий пример, действие триггера может включать вызовы встроенной процедуры ORACLE с именем raise_application_error, которая позволяет выдавать определенные пользователем сообщения об ошибках:

```
CREATE TRIGGER check_salary
    BEFORE INSERT OR UPDATE OF sal, job ON emp
    FOR EACH ROW
    WHEN (new.job != 'PRESIDENT')
DECLARE
    minsal  NUMBER;
    maxsal  NUMBER;
BEGIN
/* Дать интервал окладов для данной должности из справочника */
   SELECT losal, hisal INTO minsal, maxsal FROM sals
   WHERE job = :new.job;
   /* Если оклад вне диапазона, прибавка отрицательна, *
   /* или прибавка выше 10%, возбудить исключение.     */
   IF (:new.sal < minsal OR :new.sal > maxsal) THEN
       raise_application_error(-20225, 'Salary out of range');
    ELSIF (:new.sal < :old.sal) THEN
       raise_application_error(-20230, 'Negative increase');
    ELSIF (:new.sal > 1.1 * :old.sal) THEN
       raise_application_error(-20235, 'Increase exceeds 10%');
    END IF;
END;
```

Задания

1. Для отслеживания изменений информации о сотрудниках, создайте таблицу s_emp_log и напишите триггер, заносящий в s_emp_log информацию о времени изменения и информации о сотруднике до изменения.
2. Для отслеживания удалений информации о сотрудниках, напишите триггер, заносящий в s_emp_log удалённую информацию, а также время удаления.
3. Напишите триггер для таблице s_item, позволяющий контролировать изменение цены товара. Если новая цена отличается от старой более чем на 30%, выдаётся соответствующее сообщение и запрещается изменения данных. Реализовать используя исключения.
4. Требуется отслеживать статистические показатели, касающиеся продуктов. Т.е. для каждого существующего товара указывается количество заказчиков этого товара, количество заказанных единиц, сумма заказа. Результаты будут храниться в таблице major_stats.