СОСТАВНЫЕ ТРИГГЕРЫ (*COMPOUNDS DML* *TRIGGERS*).

ОБХОД МУТАЦИИ ТАБЛИЦ.

Триггеры PL/SQL уровня команд активизируются после вставки, обновления или удаления строк конкретной таблицы. Это самый распространенный тип триггеров. По мере создания триггеров, содержащих все больший объем логики, становится трудно следить за тем, какие триггеры связаны с теми или иными правилами и как триг­геры взаимодействуют друг с другом. Для решения этой проблемы было бы удобно разместить триггеры строк и команд вместе в одном объекте кода. Для этого Oracle предоставляет возможность использования составных триггеров.

Чтобы лучше понять структуру и принцип действия составных триггеров – можно вспомнить пакеты, в которых весь сопутствующий код и логика находятся в одном месте, что упрощает его отладку и изменение. К этому стоит добавить, что составные триггеры позволяют использовать глобальные переменные, определяемые вместе с кодом, который с ними работает. Срабатывают такие триггеры при разных событиях и в разные моменты времени (на уровне оператора или строки, при вставке/обновлении/удалении, до или после события).

Рассмотрим основы синтаксиса составных триггеров:

* BEFORE STATEMENT — код этого раздела выполняется до команды DML;
* BEFORE EACH ROW — код этого раздела выполняется перед обработкой каждой строки командой DML;
* AFTER EACH ROW — код этого раздела выполняется после обработки каждой строки командой DML;
* AFTER STATEMENT — код этого раздела выполняется после команды DML.

В таких триггерах нет секции инициализации, но для этих целей можно использовать секцию BEFORE STATEMENT. Если в триггере нет ни BEFORE STATEMENT секции, ни AFTER STATEMENT секции, и оператор не затрагивает ни одну запись, такой триггер не срабатывает.

Сразу стоит оговориться об ограничениях составных триггеров:

* Нельзя обращаться к псевдозаписям («псевдо» объясняется тем, что они обладают не всеми свойствами настоящих записей PL/SQL) OLD, NEW в секциях уровня выражения (BEFORE STATEMENT и AFTER STATEMENT);
* Изменять значения полей псевдозаписи NEW можно только в секции BEFORE EACH ROW;
* Исключения, созданные в одной секции, нельзя обрабатывать в другой секции. Если такое исключение будет где-то внутри определенной составной конструкции, то во второй конструкции использовать его уже будет нельзя, придется заново выводить;
* Если используется оператор GOTO, он должен указывать на код в той же секции;
* Нельзя создавать переменные типа LONG и LONG RAW.

Такой тип триггеров очень удобно использовать, для решения проблемы мутации таблицы (ошибка ORA - 04091). Такая ошибка возникает, если в триггере уровня строки выполняется изменение или чтение данных из той же самой таблицы, для которой данный триггер должен был сработать. Oracle не позволит это сделать и выдаст ошибку. Но зачастую, практическая задача подразумевает именно работу с заблокированной таблицей, раньше это решалось через пакет, но этот способ очень объемный и сложный в исполнении. Сейчас проблему мутации легко обойти с помощью составного триггера.

Доступность этого способа, во-первых, связана с тем, что в триггере уровня инструкции, в отличие от триггера уровня строки, мутации не возникает. Во-вторых, можно использовать глобальные переменные для всех секций составного триггера.

ИСПОЛЬЗОВАНИЕ СОСТАВНЫХ ТРИГГЕРОВ НА ПРАКТИКЕ

Для начала создадим 2 таблицы: «employees» и «aud\_empl».

CREATE TABLE employees(

emp\_id varchar2(50) NOT NULL PRIMARY KEY,

name varchar2(50) NOT NULL,

salary number NOT NULL

);

CREATE TABLE aud\_emp(

upd\_by varchar2(50) NOT NULL,

upd\_dt date NOT NULL,

field varchar2(50) NOT NULL,

old\_value varchar2(50) NOT NULL,

new\_value varchar2(50) NOT NULL);

Задача будет заключаться в том, чтобы любое изменение в таблице «employees» отражалось в таблице аудита. А именно, при обновлении каждой строки таблицы «employees», вместо выполнения операции вставки в таблицу аудита сохраняем данные в (buffer). Как только будет достигнут порог (например, 1000 записей), буферизованные данные записываются в таблицу аудита и сбрасывается счетчик буферизации.  
И, наконец, в рамках оператора AFTER очищаются все оставшиеся данные, оставшиеся в буфере.

CREATE OR REPLACE TRIGGER aud\_emp

FOR INSERT OR UPDATE

ON employees

COMPOUND TRIGGER

TYPE t\_emp\_changes IS TABLE OF aud\_emp%ROWTYPE INDEX BY SIMPLE\_INTEGER;

v\_emp\_changes t\_emp\_changes;

v\_index SIMPLE\_INTEGER := 0;

v\_threshhold CONSTANT SIMPLE\_INTEGER := 1000;

v\_user VARCHAR2(50);

PROCEDURE flush\_logs

IS

v\_updates CONSTANT SIMPLE\_INTEGER := v\_emp\_changes.count();

BEGIN

FORALL v\_count IN 1..v\_updates

INSERT INTO aud\_emp

VALUES v\_emp\_changes(v\_count);

v\_emp\_changes.delete();

v\_index := 0;

END flush\_logs;

AFTER EACH ROW

IS

BEGIN

IF INSERTING THEN

v\_index := v\_index + 1;

v\_emp\_changes(v\_index).upd\_dt := SYSDATE;

v\_emp\_changes(v\_index).upd\_by := SYS\_CONTEXT ('USERENV', 'SESSION\_USER');

v\_emp\_changes(v\_index).emp\_id := :NEW.emp\_id;

v\_emp\_changes(v\_index).action := 'Create';

v\_emp\_changes(v\_index).field := '\*';

v\_emp\_changes(v\_index).from\_value := 'NULL';

v\_emp\_changes(v\_index).to\_value := '\*';

ELSIF UPDATING THEN

IF ( (:OLD.EMP\_ID <> :NEW.EMP\_ID)

OR (:OLD.EMP\_ID IS NULL AND :NEW.EMP\_ID IS NOT NULL)

OR (:OLD.EMP\_ID IS NOT NULL AND :NEW.EMP\_ID IS NULL)

)

THEN

v\_index := v\_index + 1;

v\_emp\_changes(v\_index).upd\_dt := SYSDATE;

v\_emp\_changes(v\_index).upd\_by := SYS\_CONTEXT ('USERENV', 'SESSION\_USER');

v\_emp\_changes(v\_index).emp\_id := :NEW.emp\_id;

v\_emp\_changes(v\_index).field := 'EMP\_ID';

v\_emp\_changes(v\_index).from\_value := to\_char(:OLD.EMP\_ID);

v\_emp\_changes(v\_index).to\_value := to\_char(:NEW.EMP\_ID);

v\_emp\_changes(v\_index).action := 'Update';

END IF;

IF ( (:OLD.NAME <> :NEW.NAME)

OR (:OLD.NAME IS NULL AND :NEW.NAME IS NOT NULL)

OR (:OLD.NAME IS NOT NULL AND :NEW.NAME IS NULL)

)

THEN

v\_index := v\_index + 1;

v\_emp\_changes(v\_index).upd\_dt := SYSDATE;

v\_emp\_changes(v\_index).upd\_by := SYS\_CONTEXT ('USERENV', 'SESSION\_USER');

v\_emp\_changes(v\_index).emp\_id := :NEW.emp\_id;

v\_emp\_changes(v\_index).field := 'NAME';

v\_emp\_changes(v\_index).from\_value := to\_char(:OLD.NAME);

v\_emp\_changes(v\_index).to\_value := to\_char(:NEW.NAME);

v\_emp\_changes(v\_index).action := 'Update';

END IF;

IF ( (:OLD.SALARY <> :NEW.SALARY)

OR (:OLD.SALARY IS NULL AND :NEW.SALARY IS NOT NULL)

OR (:OLD.SALARY IS NOT NULL AND :NEW.SALARY IS NULL)

)

THEN

v\_index := v\_index + 1;

v\_emp\_changes(v\_index).upd\_dt := SYSDATE;

v\_emp\_changes(v\_index).upd\_by := SYS\_CONTEXT ('USERENV', 'SESSION\_USER');

v\_emp\_changes(v\_index).emp\_id := :NEW.emp\_id;

v\_emp\_changes(v\_index).field := 'SALARY';

v\_emp\_changes(v\_index).from\_value := to\_char(:OLD.SALARY);

v\_emp\_changes(v\_index).to\_value := to\_char(:NEW.SALARY);

v\_emp\_changes(v\_index).action := 'Update';

END IF;

END IF;

IF v\_index >= v\_threshhold THEN

flush\_logs();

END IF;

END AFTER EACH ROW;

AFTER STATEMENT IS

BEGIN

flush\_logs();

END AFTER STATEMENT;

END aud\_emp;

Теперь необходимо выполнить вставку в таблицу «employees» и сразу обновление.

INSERT INTO employees VALUES (1, 'emp1', 10000);

INSERT INTO employees VALUES (2, 'emp2', 20000);

INSERT INTO employees VALUES (3, 'emp3', 16000);

UPDATE employees

SET salary = 2000

WHERE salary > 15000;

Теперь проверяем таблицу аудита

SELECT \* FROM aud\_emp;

Результат выполнения:

EMP\_ID,UPD\_BY,UPD\_DT,ACTION,FIELD,FROM\_VALUE,TO\_VALUE

1,Aditya,1/22/2014 10:59:33 AM,Create,\*,NULL,\*

2,Aditya,1/22/2014 10:59:34 AM,Create,\*,NULL,\*

3,Aditya,1/22/2014 10:59:35 AM,Create,\*,NULL,\*

2,Aditya,1/22/2014 10:59:42 AM,Update,SALARY,20000,2000

3,Aditya,1/22/2014 10:59:42 AM,Update,SALARY,16000,2000

Теперь любые изменения в любом поле сотрудников будут записаны в таблицу аудита.