**Оглавление**

**Общие сведения о триггерах Oracle**

**DML triggers**

**Псевдозаписи**

**Instead of dml triggers**

**Instead of triggers on Nested Table Columns of Views**

**Составные DML триггера (compound DML triggers)**

Структура составного триггера

**Основные правила определения DML триггеров**

**Ограничения DML триггеров**

**Ошибка мутирования таблицы ORA-04091**

**Системные триггеры (System triggers)**

**Триггеры уровня схемы (schema triggers)**

**Триггеры уровня базы данных (database triggers)**

**Instead of create triggers**

**Атрибуты системных триггеров**

**События срабатывания системных триггеров**

**Компиляция триггеров**

**Исключения в триггерах**

**Порядок выполнения триггеров**

**Права для операций с триггерами**

**Словари данных с информацией о триггерах:**

# Общие сведения о триггерах Oracle

Триггер – это именованный pl/sql блок, который хранится в базе данных.

* Можно сделать триггер enable или disable
* Нельзя самому вызвать триггер, он всегда срабатывает только на определенное событие автоматически(если он enable)
* Не стоит создавать рекурсивные триггера. Т.е., например, триггер after update, в котором выполняется update той же таблицы. В этом случае триггер будет срабатывать рекурсивно до тех пор, пока не закончится память.

**Классификация триггеров**:

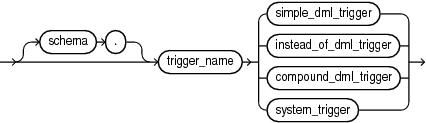
* DML trigger (на таблицу или представление)
* System trigger (на схему или базу данных)
* Conditional trigger (те, которые имеют условие when)
* Instead of trigger (dml триггер на представление или system триггер на команду create)

**Зачем использовать триггеры:**

* Для автоматической генерации значений виртуального поля
* Для логгирования
* Для сбора статистики
* Для изменения данных в таблицах, если в dml операции участвует представление
* Для предотвращения dml операций в какие-то определенные часы
* Для реализации сложных ограничений целостности данных, которые невозможно осуществить через описательные ограничения, установленные при создании таблиц
* Для организации всевозможных видов аудита
* Для оповещения других модулей о том, что делать в случае изменения информации в БД
* Для реализации бизнес логики
* Для организации каскадных воздействий на таблицы БД
* Для отклика на системные события в БД или схеме

Description of create_trigger.gif follows

где **plsql\_trigger\_source**, это такая конструкция:

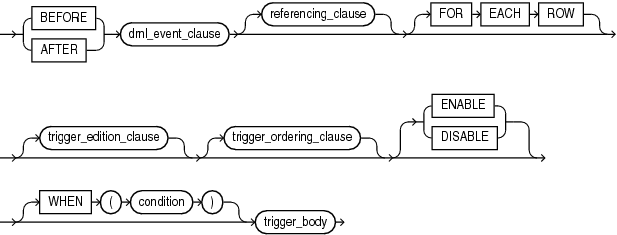


Конструкции **simple\_dml\_trigger, instead\_of\_dml\_trigger, compound\_dml\_trigger** и **system\_trigger** будут приведены в соответствующих разделах статьи.

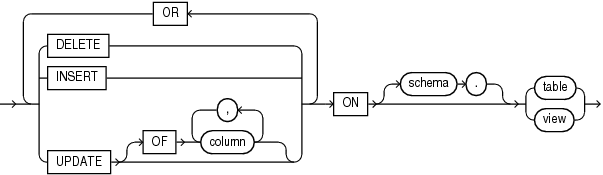
# DML triggers

* DML триггеры создаются для таблиц или представлений, срабатывают при вставке, обновлении или удалении записей.
* Триггер может быть создан в другой схеме, отличной от той, где определена таблицы. В таком случае текущей схемой при выполнении триггера считается схема самого триггера.
* При операции MERGE срабатывают триггеры на изменение, вставку или удаление записей в зависимости от операции со строкой.
* Триггер – часть транзакции, ошибка в триггере откатывает операцию, изменения таблиц в триггере становятся частью транзакции.
* Если откатывается транзакция, изменения триггера тоже откатываются.
* В триггерах запрещены операторы DDL и управления транзакциями (исключения – автономные транзакции).

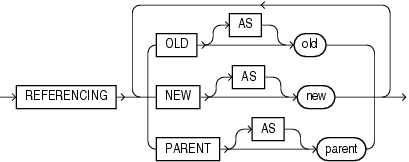
Конструкция **simple\_dml\_trigger**:



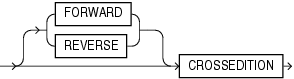
**Где, dml\_event\_clause**:



**referencing\_clause**:



**trigger\_edition\_clause**:



**trigger\_body**:

Description of trigger_body.gif follows

**По привязанному объекту делятся на**:

* На таблице
* На представлении (instead of trigger)

**По событиям запуска**:

* Вставка записей (insert)
* Обновление записей (update)
* Удаление записей (delete)

**По области действия**:

* Уровень всей команды (statement level triggers)
* Уровень записи (row level triggers)
* Составные триггеры (compound triggers)

**По времени срабатывания**:

* Перед выполнением операции (before)
* После выполнения операции (after)

**Crossedition triggers** - служат для межредакционного взаимодействия,например для переноса и трансформации данных из полей, отсутствующих в новой редакции, в другие поля.

Условные предикаты для определения операции, на которую сработал триггер:

|  |  |
| --- | --- |
| **Предикат** | **Описание** |
| Inserting | True, если триггер сработал на операцию Insert |
| Updating | True, если триггер сработал на операцию Update |
| Updating(‘colum’) | True, если триггер сработал на операцию Update, которая затрагивает определенное поле |
| Deleting | True, если триггер сработал на операцию Delete |

Эти предикаты могут использоваться везде, где можно использовать Boolean выражения.

Пример:

CREATE OR REPLACE TRIGGER t

BEFORE

**INSERT OR**

**UPDATE OF salary, department\_id OR**

**DELETE**

ON employees

BEGIN

CASE

WHEN **INSERTING** THEN

DBMS\_OUTPUT.PUT\_LINE('Inserting');

WHEN **UPDATING('salary')** THEN

DBMS\_OUTPUT.PUT\_LINE('Updating salary');

WHEN **UPDATING('department\_id')** THEN

DBMS\_OUTPUT.PUT\_LINE('Updating department ID');

WHEN **DELETING** THEN

DBMS\_OUTPUT.PUT\_LINE('Deleting');

END CASE;

END;

## Псевдозаписи

Существуют псевдозаписи, позволяющие обратиться к полям изменяемой записи и получить значения полей до изменения и значения полей после изменения. Это записи old и new. С помощью конструкции **Referencing** можно изменить их имена. Структура этих записей tablename%rowtype. Эти записи есть только у триггеров row level или у compound триггеров (с секциями уровня записи).

|  |  |  |
| --- | --- | --- |
| **Операция срабатывания триггера** | **OLD.column** | **NEW.column** |
| Insert | Null | Новое значение |
| Update | Старое значение | Новое значение |
| Delete | Старое значение | Null |

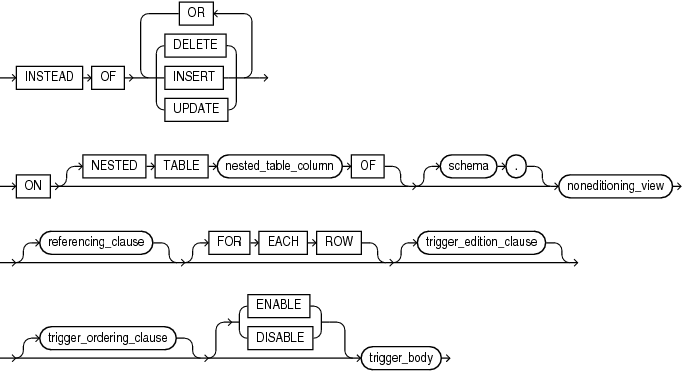
**Restrictions**:

* С псевдозаписями запрещены операции уровня всей записи ( :new = null;)
* Нельзя изменять значения полей записи old
* Если триггер срабатывает на delete, нельзя изменить значения полей записи new
* В триггере after нельзя изменить значения полей записи new

## Instead of dml triggers

* Создаются для представлений (view) и служат для замещения DML операций своим функционалом.
* Позволяют производить операции вставки/обновления или удаления для необновляемых представлений.

Конструкция **instead\_of\_dml\_trigger**:



* Это всегда триггер уровня записи (row level)
* Имеет доступ к псевдозаписям old и new, но не может изменять их
* Заменяет собой dml операцию с представлением (view)

Пример:

CREATE OR REPLACE VIEW order\_info AS

SELECT c.customer\_id, c.cust\_last\_name, c.cust\_first\_name,

o.order\_id, o.order\_date, o.order\_status

**FROM customers c, orders o**

WHERE c.customer\_id = o.customer\_id;

CREATE OR REPLACE TRIGGER order\_info\_insert

**INSTEAD OF INSERT ON order\_info**

DECLARE

duplicate\_info EXCEPTION;

PRAGMA EXCEPTION\_INIT (duplicate\_info, -00001);

BEGIN

**INSERT INTO customers**

**(customer\_id, cust\_last\_name, cust\_first\_name)**

**VALUES (**

**:new.customer\_id,**

**:new.cust\_last\_name,**

**:new.cust\_first\_name);**

**INSERT INTO orders (order\_id, order\_date, customer\_id)**

**VALUES (**

**:new.order\_id,**

**:new.order\_date,**

**:new.customer\_id);**

EXCEPTION

WHEN duplicate\_info THEN

RAISE\_APPLICATION\_ERROR (

num=> -20107,

msg=> 'Duplicate customer or order ID');

END order\_info\_insert;

/

## Instead of triggers on Nested Table Columns of Views

Можно создать триггер для вложенной в представлении таблицы. В таком триггере также присутствует дополнительная псевдозапись – parent, которая ссылается на всю запись представления (стандартные псевдозаписи old и new ссылаются только на записи вложенной таблицы)

**Пример**:

-- Create type of nested table element:

CREATE OR REPLACE TYPE nte

AUTHID DEFINER IS

OBJECT (

emp\_id NUMBER(6),

lastname VARCHAR2(25),

job VARCHAR2(10),

sal NUMBER(8,2)

);

/

-- Created type of nested table:

CREATE OR REPLACE TYPE emp\_list\_ IS

TABLE OF nte;

/

-- Create view:

CREATE OR REPLACE VIEW **dept\_view** AS

SELECT d.department\_id,

d.department\_name,

CAST (MULTISET (SELECT e.employee\_id, e.last\_name, e.job\_id, e.salary

FROM employees e

WHERE e.department\_id = d.department\_id

)

AS emp\_list\_

) **emplist**

FROM departments d;

-- Create trigger:

CREATE OR REPLACE TRIGGER dept\_emplist\_tr

**INSTEAD OF INSERT ON NESTED TABLE emplist OF dept\_view**

**REFERENCING NEW AS Employee**

**PARENT AS Department**

FOR EACH ROW

BEGIN

-- Insert on nested table translates to insert on base table:

INSERT INTO employees (

employee\_id,

last\_name,

email,

hire\_date,

job\_id,

salary,

department\_id

)

VALUES (

:Employee.emp\_id, -- employee\_id

:Employee.lastname, -- last\_name

:Employee.lastname || '@company.com', -- email

SYSDATE, -- hire\_date

:Employee.job, -- job\_id

:Employee.sal, -- salary

:Department.department\_id -- department\_id

);

END;

**Запускает триггер оператор insert:**

INSERT INTO TABLE (

SELECT d.emplist

FROM dept\_view d

WHERE department\_id = 10

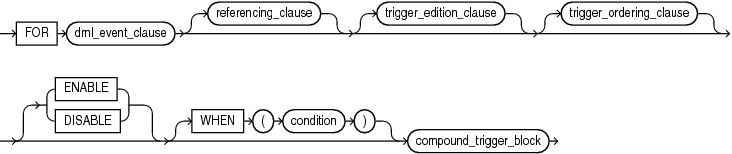
)

VALUES (1001, 'Glenn', 'AC\_MGR', 10000);

## Составные DML триггера (compound DML triggers)

Появившиеся в версии 11G эти триггера включают в одном блоке обработку всех видов DML триггеров.

Конструкция **compound\_dml\_trigger**:



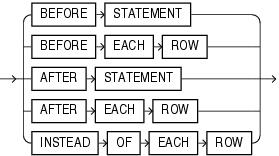
Где, **compound\_trigger\_block**:

Description of compound_trigger_block.gif follows

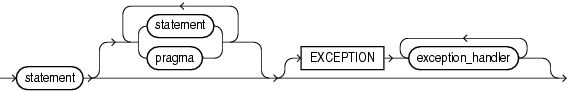
**timing\_point\_section**:

Description of timing_point_section.gif follows

**timing\_point**:



**tps\_body**:

****

* Срабатывают такие триггера при разных событиях и в разные моменты времени (на уровне оператора или строки, при вставке/обновлении/удалении, до или после события).
* Не могут быть автономными транзакциями.

В основном используются, чтобы:

* Собирать в коллекцию строки для вставки в другую таблицу, чтобы периодически вставлять их пачкой
* Избежать ошибки мутирующей таблицы (mutating-table error)

### Структура составного триггера

Может содержать переменные, которые живут на всем протяжении выполнения оператора, вызвавшего срабатывание триггера.

Такой триггер содержит следующие секции:

* Before statement
* After statement
* Before each row
* After each row

В этих 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.

**Restrictions**:

* Нельзя обращаться к псевдозаписям old, new или parent в секциях уровня выражения (before statement и after statement)
* Изменять значения полей псевдозаписи new можно только в секции before each row
* Исключения, сгенерированные в одной секции, нельзя обрабатывать в другой секции
* Если используется оператор goto, он должен указывать на код в той же секции

**Пример**:

create or replace trigger tr\_table\_test\_compound

for update or delete or insert on table\_test

compound trigger

v\_count pls\_integer := 0;

before statement is

begin

dbms\_output.put\_line ( 'before statement' );

end before statement;

before each row is

begin

dbms\_output.put\_line ( 'before insert' );

end before each row;

after each row is

begin

dbms\_output.put\_line ( 'after insert' );

v\_count := v\_count + 1;

end after each row;

after statement is

begin

dbms\_output.put\_line ( 'after statement' );

end after statement;

end tr\_table\_test\_compound;

## Основные правила определения DML триггеров

* Update of – позволяет указать список изменяемых полей для запуска триггера
* Все условия в заголовке и When … проверяются без запуска триггера на стадии выполнения SQL
* В операторе When можно использовать только встроенные функции
* Можно делать несколько триггеров одного вида, порядок выполнения не определен по умолчанию, но его можно задать с помощью конструкции FOLLOWS TRIGGER\_FIRST
* Ограничения уникальности проверяются при изменении записи, то есть после выполнения триггеров before
* Секция объявления переменных определяется словом **DECLARE**
* Основной блок триггера подчиняется тем же правилам, что и обычные PL/SQL блоки

## Ограничения DML триггеров

* нельзя выполнять DDL statements (только в автономной транзакции)
* нельзя запускать подпрограммы с операторами контроля транзакций
* не имеет доступа к SERIALLY\_REUSABLE пакетов
* размер не может превышать 32К
* нельзя декларировать переменные типа LONG и LONG RAW

## Ошибка мутирования таблицы ORA-04091

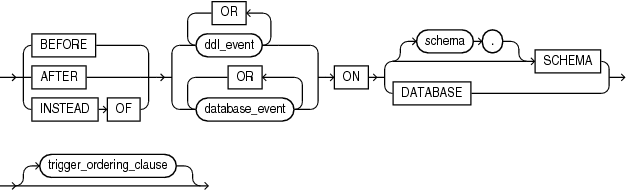
Если в триггере уровня строки попытаться получить или изменить данные в целевой таблицы, то Oracle не позволит это сделать и выкинет ошибку ***ORA-04091 Таблица TABLE\_TEST изменяется, триггер/функция может не заметить это***.

Для обхода данной проблемы используются следующие приемы:

* использовать триггеры уровня операции
* автономная транзакция в триггере
* использовать сторонние структуры (коллекции уровня пакета)
* использовать COMPOUND TRIGGER
* изменение самого алгоритма с выносом функционала из триггера

# Системные триггеры (System triggers)

Конструкция **system\_trigger**:



Такие триггеры относятся или к схеме, или ко всей базе данных.

Есть несколько вариантов, в какой момент времени срабатывает системный триггер:

* До того, как будет выполнена операция (на которую срабатывает триггер)
* После того, как будет выполнена операция (на которую срабатывает триггер)
* Вместо выполнения оператора Create

## Триггеры уровня схемы (schema triggers)

* Срабатывает всегда, когда пользователь-владелец схемы запускает событие (выполняет операцию), на которую должен срабатывать триггер.
* В случае, если любой другой пользователь запускает процедуру/функцию, которая вызывается с правами создателя, и в этой процедуре/функции выполняется операция, на которую создан системный триггер – этот триггер сработает.

**Пример триггера**:

CREATE OR REPLACE TRIGGER drop\_trigger

BEFORE DROP ON hr.SCHEMA

BEGIN

RAISE\_APPLICATION\_ERROR (

num => -20000,

msg => 'Cannot drop object');

END;

/

## Триггеры уровня базы данных (database triggers)

* Такой триггер срабатывает когда любой пользователь БД выполняет команду, на которую создан триггер.

**Пример триггера**:

CREATE OR REPLACE TRIGGER check\_user

AFTER LOGON ON DATABASE

BEGIN

check\_user;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR

(-20000, 'Unexpected error: '|| DBMS\_Utility.Format\_Error\_Stack);

END;

## Instead of create triggers

* Это триггер уровня схемы, который срабатывает на команду create и заменяет собой эту команду (т.е. вместо выполнения команды create выполняется тело триггера).

**Пример триггера**:

CREATE OR REPLACE TRIGGER t

INSTEAD OF CREATE ON SCHEMA

BEGIN

EXECUTE IMMEDIATE 'CREATE TABLE T (n NUMBER, m NUMBER)';

END;

/

## Атрибуты системных триггеров

|  |  |  |
| --- | --- | --- |
| **Атрибут** | **Возвращаемое значение и тип** | **Пример** |
| **ora\_client\_ip\_address** | Varchar2  ip-адрес клиента | IF (ora\_sysevent = 'LOGON') THEN  v\_addr := **ora\_client\_ip\_address**;  END IF; |
| **ora\_database\_name** | Varchar2(50)  имя базы данных | v\_db\_name := **ora\_database\_name**; |
| **ora\_des\_encrypted\_password** | Varchar2 зашифрованный по стандарту DES пароль пользователя, который создается или изменяется | IF (ora\_dict\_obj\_type = 'USER') THEN  INSERT INTO event\_table  VALUES (**ora\_des\_encrypted\_password**);  END IF; |
| **ora\_dict\_obj\_name** | Varchar2(30)  имя объекта, над которым совершается операция DDL | INSERT INTO event\_table  VALUES ('Changed object is ' ||  **ora\_dict\_obj\_name**); |
| **ora\_dict\_obj\_name\_list (**  **name\_list OUT ora\_name\_list\_t**  **)** | Pls\_integer  количество изменненых командой объектов  Name\_list – список измененных командой объектов | IF (ora\_sysevent='ASSOCIATE STATISTICS') THEN  number\_modified :=  **ora\_dict\_obj\_name\_list(name\_list)**;  END IF; |
| **ora\_dict\_obj\_owner** | Varchar2(30)  владелец объекта, над которым совершается операция DDL | INSERT INTO event\_table  VALUES ('object owner is' ||  **ora\_dict\_obj\_owner**); |
| **ora\_dict\_obj\_owner\_list (**  **owner\_list OUT ora\_name\_list\_t**  **)** | Pls\_integer  количество владельцев измененных командой объектов  Owner\_list – список владельцев изменных командой объектов | IF (ora\_sysevent='ASSOCIATE STATISTICS') THEN  number\_modified :=  **ora\_dict\_obj\_name\_list(owner\_list)**;  END IF; |
| **ora\_dict\_obj\_type** | Varchar2(20)  тип объекта, над которым совершается операция ddl | INSERT INTO event\_table  VALUES ('This object is a ' ||  **ora\_dict\_obj\_type**); |
| **ora\_grantee (**  **user\_list OUT ora\_name\_list\_t**  **)** | Pls\_integer  количество пользователей, участвующих в операции grant  User\_list – список этих пользователей | IF (ora\_sysevent = 'GRANT') THEN  number\_of\_grantees :=  **ora\_grantee(user\_list)**;  END IF; |
| **ora\_instance\_num** | Number  номер инстанса | IF (**ora\_instance\_num** = 1) THEN  INSERT INTO event\_table VALUES ('1');  END IF; |
| **ora\_is\_alter\_column (**  **column\_name IN VARCHAR2**  **)** | Boolean  True, если указанное поле было изменено операцией alter. Иначе false | IF (ora\_sysevent = 'ALTER' AND  ora\_dict\_obj\_type = 'TABLE') THEN  alter\_column := **ora\_is\_alter\_column('C')**;  END IF; |
| **ora\_is\_creating\_nested\_table** | Boolean  true, если текущее событие – это создание nested table. Иначе false | IF (ora\_sysevent = 'CREATE' AND  ora\_dict\_obj\_type = 'TABLE' AND  **ora\_is\_creating\_nested\_table**) THEN  INSERT INTO event\_table  VALUES ('A nested table is created');  END IF; |
| **ora\_is\_drop\_column (**  **column\_name IN VARCHAR2**  **)** | Boolean  true, если указанное поле удалено. Иначе false | IF (ora\_sysevent = 'ALTER' AND  ora\_dict\_obj\_type = 'TABLE') THEN  drop\_column := **ora\_is\_drop\_column('C')**;  END IF; |
| **ora\_is\_servererror (**  **error\_number IN VARCHAR2**  **)** | Boolean  true, если сгенерированно исключение с номером error\_number. Иначе false | IF **ora\_is\_servererror(error\_number)** THEN  INSERT INTO event\_table  VALUES ('Server error!!');  END IF; |
| **ora\_login\_user** | Varchar2(30)  имя текущего пользователя | SELECT **ora\_login\_user** FROM DUAL; |
| **ora\_partition\_pos** | Pls\_integer  в instead of trigger для create table позиция в тексте sql команды, где может быть вставлена конструкция partition | -- Retrieve ora\_sql\_txt into sql\_text variable  v\_n := **ora\_partition\_pos**;  v\_new\_stmt := SUBSTR(sql\_text,1,v\_n - 1)  || ' ' || my\_partition\_clause  || ' ' || SUBSTR(sql\_text, v\_n)); |
| **ora\_privilege\_list (**  **privilege\_list OUT ora\_name\_list\_t**  **)** | Pls\_integer  количество привилегий, участвующее в операции grant или revoke  Privilege\_list – список этих привилегий | IF (ora\_sysevent = 'GRANT' OR  ora\_sysevent = 'REVOKE') THEN  number\_of\_privileges :=  **ora\_privilege\_list(privilege\_list)**;  END IF; |
| **ora\_revokee (**  **user\_list OUT ora\_name\_list\_t**  **)** | Pls\_integer, количество пользователей, участвующих в операции revoke  User\_list – список этих пользователей | IF (ora\_sysevent = 'REVOKE') THEN  number\_of\_users := **ora\_revokee(user\_list)**;  END IF; |
| **ora\_server\_error (**  **position IN PLS\_INTEGER**  **)** | Number  код ошибки в указанной позиции error stack , где 1 – это вершина стека | INSERT INTO event\_table  VALUES ('top stack error ' ||  **ora\_server\_error(1)**); |
| **ora\_server\_error\_depth** | Pls\_integer  количество сообщений об ошибка в error stack | n := **ora\_server\_error\_depth**;  -- Use n with functions such as ora\_server\_error |
| **ora\_server\_error\_msg (**  **position IN PLS\_INTEGER**  **)** | Varchar2  сообщение об ошибке в указанном месте error stack | INSERT INTO event\_table  VALUES ('top stack error message' ||  **ora\_server\_error\_msg(1)**); |
| **ora\_server\_error\_num\_params (**  **position IN PLS\_INTEGER**  **)** | Pls\_integer  количество замещенных строк (с помощью формата %s) в указанной позиции error stack | n := **ora\_server\_error\_num\_params(1)**; |
| **ora\_server\_error\_param (**  **position IN PLS\_INTEGER,**  **param IN PLS\_INTEGER**  **)** | Varchar2  замещенный текст в сообщении об ошибке в указанной позиции error stack (возвращается param по счету замещенный текст) | -- Second %s in "Expected %s, found %s":  param := **ora\_server\_error\_param(1,2)**; |
| **ora\_sql\_txt (**  **sql\_text OUT ora\_name\_list\_t**  **)** | Pls\_integer  количество элементов в pl/sql коллекции sql\_text.  Сам параметр sql\_text возвращает текст команды, на которую сработал триггер | CREATE TABLE event\_table (col VARCHAR2(2030));  DECLARE  sql\_text ora\_name\_list\_t;  n PLS\_INTEGER;  v\_stmt VARCHAR2(2000);  BEGIN  n := **ora\_sql\_txt(sql\_text)**;  FOR i IN 1..n LOOP  **v\_stmt := v\_stmt || sql\_text(i);**  END LOOP;  INSERT INTO event\_table VALUES ('text of  triggering statement: ' || **v\_stmt**);  END; |
| **ora\_sysevent** | Varchar2(20)  название команды, на которую срабатывает триггер | INSERT INTO event\_table  VALUES (**ora\_sysevent**); |
| **ora\_with\_grant\_option** | Boolean  true, если привилегии выдаются with grant option. Иначе false. | IF (ora\_sysevent = 'GRANT' AND  **ora\_with\_grant\_option** = TRUE) THEN  INSERT INTO event\_table  VALUES ('with grant option');  END IF; |
| **ora\_space\_error\_info (**  **error\_number OUT NUMBER,**  **error\_type OUT VARCHAR2,**  **object\_owner OUT VARCHAR2,**  **table\_space\_name OUT VARCHAR2,**  **object\_name OUT VARCHAR2,**  **sub\_object\_name OUT VARCHAR2**  **)** | Boolean  true, если ошибка возникает из-за нехватки места. В выходных параметрах информация об объекте. | IF (**ora\_space\_error\_info (**  **eno,typ,owner,ts,obj,subobj) = TRUE)** THEN  DBMS\_OUTPUT.PUT\_LINE('The object '|| **obj**  || ' owned by ' || **owner** ||  ' has run out of space.');  END IF; |

## События срабатывания системных триггеров

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Событие** | **Описание** | **Конструкция When** | **Ограничения** | **Транзакция** | **Доступные атрибуты** |
| **AFTER STARTUP** | При запуске БД. Бывает только уровня БД. При ошибке пишет в системный лог. | Нет | Trigger cannot do database operations. | Запускает отдельную транзакцию и выполняет commit в конце  е | ora\_sysevent  ora\_login\_user  ora\_instance\_num  ora\_database\_name |
| **BEFORE SHUTDOWN** | Перед тем, как сервер начнет процесс останова. Бывает только уровня БД. При ошибке пишет в системный лог. | ora\_sysevent  ora\_login\_user  ora\_instance\_num  ora\_database\_name |
| **AFTER DB\_ROLE\_CHANGE** | При запуске БД в первый раз после смены ролей from standby to primary or from primary to standby.  используется только в конфигурации Data Guard, , бывает только уровня БД. | Нет | ora\_sysevent  ora\_login\_user  ora\_instance\_num  ora\_database\_name |
| **AFTER SERVERERROR** | Если случается любая ошибка (если с условием, то срабатывает только на ошибку, указанную в условии). При ошибке в теле триггера не вызывает себя рекурсивно. | ERRNO = eno‬‬‬‬ | Зависит от ошибки | ora\_sysevent  ora\_login\_user  ora\_instance\_num  ora\_database\_name  ora\_server\_error  ora\_is\_servererror  ora\_space\_error\_info |
| **BEFORE ALTER**  **AFTER ALTER** | Если объект изменяется командой alter | Условия на тип объекта, имя объекта, UID и/или USER | Запрещены ddl операции с объектом, из-за которого был запущен триггер. С другими объектами можно делать только следующий ddl: скомпилировать объект, create trigger, create/alter/drop table | Работает в той же транзакции | ora\_sysevent  ora\_login\_user  ora\_instance\_num  ora\_database\_name  ora\_dict\_obj\_type  ora\_dict\_obj\_name  ora\_dict\_obj\_owner  ora\_des\_encrypted\_password  (for ALTER USER events)  ora\_is\_alter\_column  (for ALTER TABLE events)  ora\_is\_drop\_column  (for ALTER TABLE events) |
| **BEFORE DROP**  **AFTER DROP** | При удалении объекта | ora\_sysevent  ora\_login\_user  ora\_instance\_num  ora\_database\_name  ora\_dict\_obj\_type  ora\_dict\_obj\_name  ora\_dict\_obj\_owner |
| **BEFORE ANALYZE**  **AFTER ANALYZE** | При срабатывании команды analyze | ora\_sysevent  ora\_login\_user  ora\_instance\_num  ora\_database\_name  ora\_dict\_obj\_name  ora\_dict\_obj\_type  ora\_dict\_obj\_owner |
| **BEFORE ASSOCIATE STATISTICS**  **AFTER ASSOCIATE STATISTICS** | При выполнении команды associate statistics | ora\_sysevent  ora\_login\_user  ora\_instance\_num  ora\_database\_name  ora\_dict\_obj\_name  ora\_dict\_obj\_type  ora\_dict\_obj\_owner  ora\_dict\_obj\_name\_list  ora\_dict\_obj\_owner\_list |
| **BEFORE AUDIT**  **AFTER AUDIT**  **BEFORE NOAUDIT**  **AFTER NOAUDIT** | При выполнении команды audit или noaudit | ora\_sysevent  ora\_login\_user  ora\_instance\_num  ora\_database\_name |
| **BEFORE COMMENT**  **AFTER COMMENT** | При добавлении комментария к объекту | ora\_sysevent  ora\_login\_user  ora\_instance\_num  ora\_database\_name  ora\_dict\_obj\_name  ora\_dict\_obj\_type  ora\_dict\_obj\_owner |
| **BEFORE CREATE**  **AFTER CREATE** | При создании объекта | ora\_sysevent  ora\_login\_user  ora\_instance\_num  ora\_database\_name  ora\_dict\_obj\_type  ora\_dict\_obj\_name  ora\_dict\_obj\_owner  ora\_is\_creating\_nested\_table  (for CREATE TABLE events) |
| **BEFORE DDL**  **AFTER DDL** | Срабатывает на большинство команд DDL, кроме: alter database, create control file, create database. | ora\_sysevent  ora\_login\_user  ora\_instance\_num  ora\_database\_name  ora\_dict\_obj\_name  ora\_dict\_obj\_type  ora\_dict\_obj\_owner |
| **BEFORE DISASSOCIATE STATISTICS**  **AFTER DISASSOCIATE STATISTICS** | При выполнении команды disassociate statistics | ora\_sysevent  ora\_login\_user  ora\_instance\_num  ora\_database\_name  ora\_dict\_obj\_name  ora\_dict\_obj\_type  ora\_dict\_obj\_owner  ora\_dict\_obj\_name\_list  ora\_dict\_obj\_owner\_list |
| **BEFORE GRANT**  **AFTER GRANT** | При выполнении команды grant | ora\_sysevent  ora\_login\_user  ora\_instance\_num  ora\_database\_name  ora\_dict\_obj\_name  ora\_dict\_obj\_type  ora\_dict\_obj\_owner  ora\_grantee  ora\_with\_grant\_option  ora\_privilege\_list |
| **BEFORE LOGOFF** | Срабатывает перед дисконнеком пользователя, бывает уровня схемы или БД | Условия на UID и/или USER | ora\_sysevent  ora\_login\_user  ora\_instance\_num  ora\_database\_name |
| **AFTER LOGON** | Срабатывает после того, как пользователь успешно установил соединение с БД. При ошибке запрещает пользователю вход. Не действует на SYS. | Запускает отдельную транзакцию и выполняет commit в конце | ora\_sysevent  ora\_login\_user  ora\_instance\_num  ora\_database\_name  ora\_client\_ip\_address |
| **BEFORE RENAME**  **AFTER RENAME** | При выполнении команды rename | Условия на тип объекта, имя объекта, UID и/или USER | Работает в той же транзакции | ora\_sysevent  ora\_login\_user  ora\_instance\_num  ora\_database\_name  ora\_dict\_obj\_name  ora\_dict\_obj\_owner  ora\_dict\_obj\_type |
| **BEFORE REVOKE**  **AFTER REVOKE** | При выполнении команды revoke | ora\_sysevent  ora\_login\_user  ora\_instance\_num  ora\_database\_name  ora\_dict\_obj\_name  ora\_dict\_obj\_type  ora\_dict\_obj\_owner  ora\_revokee  ora\_privilege\_list |
| **AFTER SUSPEND** | Срабатывает в случае, если sql команда приостанавливается по причине серверной ошибки (нехватки памяти).  При этом триггер должен изменить условия таким образом, чтобы выполнение команды было возобновлено) | ora\_sysevent  ora\_login\_user  ora\_instance\_num  ora\_database\_name  ora\_server\_error  ora\_is\_servererror  ora\_space\_error\_info |
| **BEFORE TRUNCATE**  **AFTER TRUNCATE** | При выполнении команды truncate | ora\_sysevent  ora\_login\_user  ora\_instance\_num  ora\_database\_name  ora\_dict\_obj\_name  ora\_dict\_obj\_type  ora\_dict\_obj\_owner |

# Компиляция триггеров

Если во время выполнения команды create trigger произошла ошибка, триггер все равно будет создан, но будет в невалидном состоянии. При этом все попытки выполнить операцию(на которую должен срабатывать триггер) над объектом, на котором висит такой триггер, будут завершаться ошибкой. Это не относится к случаям, когда:

* Триггер создан в состоянии disabled (или переведен в такое состояние)
* Событие триггера after startup on database
* Событие триггера after logon on database или after logon on schema и происходит попытка залогиниться под пользователем System

Чтобы перекомпилировать триггер, используйте команду alter trigger.

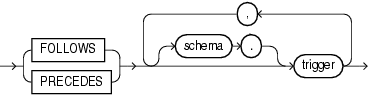
# Исключения в триггерах

В случае, если в триггере возникает исключение, вся операция откатывается (включая любые изменения, сделанные внутри триггера). Исключения из этого:

* Если событие триггера after startup on database или before shutdown on database
* Если событие триггера after logon on database и пользователь имеет привилегию administer database trigger
* Если событие триггера after logon on schema и пользователь или является владельцем схемы, или имеет привилегию alter any trigger

# Порядок выполнения триггеров

Конструкция **trigger\_ordering\_clause**:



1. Сначала выполняются все before statement триггера
2. Потом все before each row триггера
3. После все after each row триггера
4. И в конце все after statement триггера

Чтобы задать явно порядок выполнения триггеров, срабатывающих в одинаковый момент времени (потому что по умолчанию такой порядок не определен), используйте конструкции follows и precedes.

**Включение/отключение триггеров**   
Это может понадобиться, например, для загрузки большого объема информации в таблицу.

Выполнить включение/отключение триггера можно с помощью команды:

ALTER TRIGGER [*schema*.]*trigger\_name* { ENABLE | DISABLE };

Чтобы включить/отключить сразу все триггеры на таблице:

ALTER TABLE *table\_name* { ENABLE | DISABLE } ALL TRIGGERS;

Для изменения триггера можно или воспользоваться командой Create or replace trigger, или сначала удалить триггер drop trigger, а потом создать заново create trigger.

Операция alter trigger позволяет только включить/отключить триггер, скомпилировать его или переименовать.

Компиляция триггера:

alter trigger TRIGGER\_NAME compile;

# Права для операций с триггерами

Для работы с триггерами даже в своей схеме необходима привилегия create trigger, она дает права на создание, изменение и удаление.

grant create trigger to USER;

Для работы с триггерами во всех других схемах необходима привилегия \* any trigger. Обратите внимание, что права даются отдельно на создание, изменение и удаление.

grant create any trigger to USER;

grant alter any trigger to USER;

grant drop any trigger to USER;

Для работы с системными триггерами уровня DATABASE необходима привилегия ADMINISTER DATABASE TRIGGER.

grant ADMINISTER DATABASE TRIGGER to USER;

# Словари данных с информацией о триггерах:

* dba\_triggers – информация о триггерах
* dba\_source - код тела триггера
* dba\_objects – валидность триггера