



# Триггеры

(в соответствии со стандартом SQL-1999,  
с особенностями синтаксиса Oracle)

# Определения

**Триггер** – это процедура, которая автоматически запускается при возникновении определенных событий.

**Событие триггера** – событие, управляющее запуском триггера; описывается в виде логических условий.

В Oracle различают следующие типы триггеров:

- **простые триггеры** – они привязаны к определённой таблице, срабатывают при поступлении команд DML и выполняются в рамках этой команды;
- **триггеры INSTEAD OF** – они привязаны к определённому представлению, и выполняются вместо события триггера, которое является командой DML;
- **триггеры для событий уровня схемы** – они срабатывают при выполнении команд DDL и при наступлении таких событий, как подключение и отключение от базы данных, а также возникновение серверной ошибки.

# Назначение триггеров

- Проверка сложных ограничений целостности.
- Автоматизация обработки данных.
- Аудит действий пользователей.
- Установка начальных значений при добавлении данных в таблицы.
- Проверка дифференцированных прав доступа.

# Общий синтаксис создания триггера (SQL:1999)

trigger\_definition ::=

```
CREATE TRIGGER trigger_name
{ BEFORE | AFTER }
{ INSERT | DELETE | UPDATE [ OF column_comma_list ] }
ON table_name
[ REFERENCING old_or_new_values_alias_list ]
triggered_action
```

triggered\_action ::= [ FOR EACH { ROW | STATEMENT } ]  
[ WHEN left\_paren conditional\_expression right\_paren ]  
triggered\_SQL\_statement

triggered\_SQL\_statement ::= SQL\_procedure\_statement  
| BEGIN ATOMIC  
SQL\_procedure\_statement\_semicolonlist  
END

old\_or\_new\_values\_alias ::= OLD [ ROW ] [ AS ] correlation\_name  
| NEW [ ROW ] [ AS ] correlation\_name  
| OLD TABLE [ AS ] identifier  
| NEW TABLE [ AS ] identifier

# Синтаксис создания обычного триггера (Oracle)

```
CREATE [OR REPLACE] TRIGGER <имя триггера>
  { BEFORE | AFTER }
  { INSERT | DELETE | UPDATE [ OF column_commlist ] }
  ON <имя таблицы>
  [ REFERENCING old_or_new_values_alias_list ]
  [ FOR EACH { ROW | STATEMENT } ]
  [ WHEN <условие> ]
[ DECLARE
-- описание переменных, констант и др. элементов программы
]
BEGIN
  -- программа на процедурном языке (PL/SQL)
  [ EXCEPTION
  -- обработка исключительных ситуаций
  ]
END;
/
```

# Основные параметры простого триггера

- INSERT | DELETE | UPDATE [of column] – событие триггера.  
Событием триггера может быть одна команда или любая комбинация указанных команд.
- BEFORE | AFTER – время срабатывания триггера: перед выполнением события триггера или после него.  
Ограничения целостности проверяются во время выполнения события триггера.
- ON <имя таблицы> – таблица, к которой привязан триггер.
- FOR EACH { ROW | STATEMENT } – область действия триггера (для каждой строки или для команды).
- WHEN <условие> – условие срабатывания триггера.  
Если оно не выполняется, триггер не будет запущен.

# Запуск и выполнение триггеров

Триггер запускается **автоматически** при наступлении события триггера.

Триггер выполняется в рамках той транзакции, к которой относится событие триггера, поэтому процедура триггера **не может содержать команды управления транзакциями и команды DDL.**

Если команда инициирует выполнение более чем одного триггера, то, в зависимости от типов, они выполняются в таком порядке:

1. "Перед началом выполнения команды" (Before-statement trigger)
2. "Перед обработкой записи" (Before-row trigger)
3. "После обработки записи" (After-row trigger)
4. "После окончания выполнения команды" (After-statement trigger)

# Пример 1.

## Сложное ограничение целостности

- Зарплата сотрудника должна попадать в один из интервалов
- значений зарплаты, которые установлены
- для разных категорий сотрудников в таблице sal\_grade.

```
create or replace trigger check_salary
  before INSERT or UPDATE of salary ON emp
  for each row
  when :new.salary >= 12000
declare cnt number(3);
begin
  select count(*) into cnt from sal_grade
  where :new.salary between low_value and high_value
  if cnt=0 then raise_application_error(-20050, 'Зарплата не попадает ни в
  один из допустимых интервалов');
  end if;
end;
/
-- :new. – обращение к полю новой записи (добавляемой или изменённой).
```



## Пример 2.

### Аудит действий пользователей

*-- Отслеживаем действия пользователей над таблицей TAB  
-- и фиксируем данные о них в специальной таблице TAB\_LOG  
-- записывая туда имя пользователя, признак действия и дату.*

```
create or replace trigger audit_tab
```

```
  after INSERT or UPDATE or DELETE ON tab
```

```
declare ch char:='U';
```

```
begin
```

```
  if INSERTING then ch:= 'I';
```

```
  elsif DELETING then ch:= 'D';
```

```
  end if;
```

```
  insert into tab_log values ( substr (user, 1, 30), ch, sysdate);
```

```
end;
```

```
/
```

*-- INSERTING, DELETING и UPDATING – условные предикаты,*

*-- позволяющие определить, какая операция явилась событием триггера*

## Пример 3.

# Автоматизация обработки данных

*-- Структура архивной таблицы:*

```
create table emp_post(  
    id    number(6) not null,  
    post  varchar2(40) not null,  
    sdate date not null);
```

*-- Автоматическое добавление данных о предыдущей должности  
-- сотрудника в архив с указанием даты, когда произошел перевод  
-- на другую должность.*

```
create or replace trigger cross_emp  
    before UPDATE of post ON emp  
    for each row  
begin  
    insert into emp_post  
        values(:old.id, :old.post, trunc(sysdate));  
end;  
/
```

*-- :old. – обращение к полю старой записи (удалённой или изменяемой).*

## Пример 4.

### Установка значений по умолчанию

```
-- Если дата приема сотрудника на работу не указана,  
-- то установить текущую дату.  
-- Перевести ФИО сотрудника в верхний регистр.  
create or replace trigger set_emp  
  before INSERT or UPDATE ON emp  
  for each row  
begin  
  if :new.date_get IS NULL then :new.date_get := trunc(sysdate);  
  else :new.date_get := trunc(:new.date_get);  
  end if;  
  :new.name := upper(:new.name);  
end;  
/
```

# Проблема мутирующих таблиц

**Пример.** В таблице **contracts** содержатся выплаты по договорам: номер договора, получатель выплаты и выплата в процентах от суммы договора. Проверить, что сумма выплат по одному договору не превышает 100%.

```
create or replace trigger check_payment
  after INSERT or UPDATE of payment ON contracts
  for each row
declare summ number(5,2):=0;
begin
  select sum(payment) into summ from contracts
    where id=:new.id;
  if summ>100 then raise_application_error(-20150,
    'Сумма выплат больше 100 процентов');
  end if;
end;
/
```

**Вопрос:** к чему может привести операция чтения или изменения данных в таблице, к которой привязан триггер уровня строки?

# Вариант решения проблемы мутирующих таблиц

Таблица OBJECTS содержит иерархию следующего вида:

Здание

    Часть здания

        Помещение

            Часть помещения

Например, таблица Objects может иметь следующее содержимое (obj\_whole\_id – внешний ключ на поле obj\_id):

obj_id	Name	prop1	prop2	obj_whole_id
10	здание 22	складское помещение	неотапливаемое	null
11	1 этаж	складское помещение	неотапливаемое	10
12	комната 101	складское помещение	неотапливаемое	11
13	комната 102	складское помещение	неотапливаемое	11
14	тамбур	складское помещение	неотапливаемое	13
15	2 этаж	офисное помещение	отапливаемое	10
...	...	...	...	...

# Вариант решения проблемы мутирующих таблиц

Постановка задачи:

**В каждой записи, начиная со второго уровня иерархии, есть ссылка на элемент верхнего уровня и два общих свойства, которые надо менять при изменении ссылки.**

Таким образом, изменение одной записи повлечет за собой изменения во всех подчинённых записях.

**Общий принцип решения:**

1. Создать вспомогательную таблицу TEMP\_OBJ.
2. Создать триггер уровня строки на таблицу OBJECTS, который будет записывать во вспомогательную таблицу идентификатор изменённой строки и идентификатор текущей транзакции.
3. Создать триггер уровня предложения, который будет производить изменения свойств в таблице OBJECTS.

# Пример. Триггер уровня строки

```
CREATE OR REPLACE TRIGGER lau_obj
  BEFORE UPDATE
  ON objects
  referencing new as new old as old
  FOR EACH ROW
  declare
    tran_id varchar2(100);
  begin
    -- проверка изменения общего свойства
    if :old.prop1<>:new.prop1 OR :old.prop2<>:new.prop2 then
      tran_id := dbms_transaction.local_transaction_id;
      -- запись первичного ключа в доп. таблицу
      insert into temp_obj values(:new.obj_id, tran_id);
    end if;
  end;
/
```

# Продолжение примера.

## Триггер уровня предложения

```
CREATE OR REPLACE TRIGGER change_obj
AFTER UPDATE ON objects
declare
    CURSOR my_tran (par_tran_id IN varchar2) IS
        SELECT obj_id, local_tran_id FROM temp_obj
            WHERE local_tran_id = par_tran_id ;
    temp_tran_id varchar2(100);
    temp_1 varchar2(30);
    temp_2 date;
begin
    temp_tran_id:= dbms_transaction.local_transaction_id;
    FOR rec IN my_tran (temp_tran_id) LOOP
        SELECT prop1, prop2 into temp_1, temp_2 FROM objects
            WHERE obj_id = rec.obj_id;
        DELETE FROM temp_obj
            WHERE local_tran_id = temp_tran_id AND obj_id = rec.obj_id ;
        UPDATE objects SET prop1 = temp_1, prop2 = temp_2
            WHERE obj_whole_id = rec.obj_id;
    END LOOP;
end;
/
```



## Мутирующие таблицы: заключение

- Мутирующей считается таблица, изменяемая командой INSERT, UPDATE, DELETE, которая запустила этот триггер, а также таблицы, которые связаны с ней ссылочной целостностью DELETE CASCADE.
- Таблица не считается мутирующей для триггера уровня предложения, за исключением случая, когда триггер запускается как результат DELETE CASCADE.
- Триггер не может обращаться к мутирующей таблице: читать или изменять её.

# Триггеры INSTEAD OF (начиная с версии Oracle8i)

**Особенность выполнения:** триггеры INSTEAD-OF выполняются ВМЕСТО тех команд, которые являются событием триггера.

**Назначение:** обычно триггеры INSTEAD-OF применяются для обновления представлений, которые не являются обновляемыми.

Ограничения целостности для базовой таблицы представления проверяются при выполнении команд изменения данных, которые выполняются внутри триггера.

**Ограничения триггеров INSTEAD OF:**

- нельзя указывать тип BEFORE / AFTER;
- нельзя определять триггер уровня предложения;
- для команды UPDATE нельзя указывать список столбцов;
- нельзя указывать условие WHEN.

**Внимание! Если в теле триггера отсутствует команда DML, то событие триггера НЕ ВЫПОЛНЯЕТСЯ!**

# Синтаксис триггеров INSTEAD OF

```
CREATE [OR REPLACE] TRIGGER <имя триггера>
  INSTEAD OF { INSERT | DELETE | UPDATE }
  ON { <имя представления> | <имя объектного представления> }
  [ REFERENCING old_or_new_values_alias_list ]
  [ FOR EACH ROW ]
[DECLARE
-- описание переменных, констант и др. элементов программы
]
BEGIN
    -- программа на процедурном языке (PL/SQL)
  [EXCEPTION
    -- обработка исключительных ситуаций
  ]
END;
/
```

# Пример триггера INSTEAD OF.

## Исходные данные

### -- Таблица «Отделы»

```
create table DEPART (  
  did    number(3) primary key,           -- номер отдела  
  dname  varchar2(100) not null);        -- название
```

### -- Таблица «Должности»

```
create table POSTS (  
  post   varchar2(50) primary key,       -- название должности  
  sal    number(8,2) default 10000.0,   -- оклад  
  check (sal >= 4500));
```

### -- Таблица «Сотрудники»

```
create table EMP (  
  id     number(6) primary key,         -- идентификатор сотрудника  
  name   varchar2(60) not null,        -- ФИО сотрудника  
  did    number(3) references depart,   -- номер отдела  
  post   varchar2(50) references posts, -- должность  
  exp    number(4,2) default 1, ...);  -- количество ставок (от 0.25 до 1)
```

### -- Представление «Должности сотрудников»

```
create or replace view STAFF as  
  select d.did, d.dname, e.id, e.name, e.exp, p.post, p.sal  
  from depart d, emp e, posts p  
  where d.did=e.did and e.post=p.post;
```

# Пример триггера INSTEAD OF. Изменение данных

```
CREATE TRIGGER staff_update
INSTEAD OF UPDATE ON staff FOR EACH ROW
BEGIN
  IF :new.id<>:old.id OR :new.did<>:old.did OR :new.name<>:old.name
    OR :new.dname<>:old.dname
    THEN raise_application_error(-20160, 'Нельзя изменять название и
      номер отдела, имя и ID сотрудника через представление STAFF');
  END IF;
  IF :new.post<>:old.post OR :new.EXP<>:old.EXP THEN
    update emp
      SET post=:new.post, EXP=:new.EXP
      where id = :old.id;
  END IF;
  IF :new.sal<>:old.sal THEN
    update posts
      SET sal = :new.sal
      where post = :old.post;
  END IF;
END;
/
UPDATE STAFF SET sal = sal+2000 where post='экономист';
```

# Триггеры для событий уровня схемы и БД (начиная с версии Oracle8i)

<i>Событие уровня БД</i>	<i>Описание триггера</i>
STARTUP	Срабатывает при запуске сервера БД
SHUTDOWN	Срабатывает при останове сервера БД
SERVERERROR	Срабатывает при возникновении серверной ошибки
LOGON	Срабатывает при успешном подключении к системе клиентского приложения
LOGOFF	Срабатывает перед отключением клиентского приложения
<i>Событие уровня схемы</i>	<i>Описание триггера</i>
CREATE	Срабатывает при добавлении к схеме нового объекта командой CREATE
DROP	Срабатывает перед попыткой удалить объект командой DROP
ALTER	Срабатывает при изменении объекта командой ALTER

# Атрибуты событий уровня схемы и БД

<i>Имя</i>	<i>Тип</i>	<i>Описание</i>
SYSEVENT	varchar2(30)	Имя события, активизировавшего триггер.
LOGIN_USER	varchar2(30)	Имя пользователя, инициировавшего сеанс работы с Oracle.
INSTANCE_NUM	number	Имя экземпляра СУБД.
DATABASE_NAME	varchar2(50)	Имя БД.
DICTIONARY_OBJ_OWNER	varchar2(30)	Владелец объекта из словаря-справочника, действие с которым привело к активизации триггера.
DICTIONARY_OBJ_NAME	varchar2(30)	Имя объекта из словаря-справочника, действие с которым привело к активизации триггера.
DICTIONARY_OBJ_TYPE	varchar2(30)	Тип объекта из словаря-справочника, действие с которым привело к активизации триггера.
IS_SERVERERROR	boolean	Функция, возвращающая TRUE при наличии указанной ошибки в текущем магазине ошибок; FALSE в противном случае.
SERVER_ERROR	number	Функция, возвращающая номер ошибки на указанном месте магазина ошибок. 1 соответствует верхушке магазина.
DES_ENCRYPTED_PASSWORD	varchar2(30)	Зашифрованный (DES) пароль создаваемого или изменяемого пользователя.

# Синтаксис триггеров уровня БД

```
CREATE [ OR REPLACE ] TRIGGER <имя триггера>
  { AFTER STARTUP | BEFORE SHUTDOWN |
    AFTER LOGON | BEFORE LOGOFF |
    AFTER SERVERERROR }
  ON DATABASE
  [ WHEN <условие> ]
[ DECLARE
  -- описание переменных, констант и др. элементов программы
]
BEGIN
  -- программа на процедурном языке (PL/SQL)
  [ EXCEPTION
    -- обработка исключительных ситуаций
  ]
END;
/
```



# События триггера, свойственные им правила и атрибуты

<i>Событие</i>	<i>Правило</i>	<i>Атрибуты</i>
LOGON	Условие можно указать, воспользовавшись USERID() или USERNAME()	SYSEVENT, LOGIN_USER, INSTANCE_NUM, DATABASE_NAME
LOGOFF	Условие можно указать, воспользовавшись USERID() или USERNAME()	SYSEVENT, LOGIN_USER, INSTANCE_NUM, DATABASE_NAME
BEFORE CREATE, AFTER CREATE	В пределах триггера удалять создаваемый объект нельзя. Триггер выполняется в рамках текущей транзакции.	SYSEVENT, LOGIN_USER, INSTANCE_NUM, DATABASE_NAME, DICTIONARY_OBJ_TYPE, DICTIONARY_OBJ_NAME, DICTIONARY_OBJ_OWNER
BEFORE ALTER, AFTER ALTER	В пределах триггера удалять изменяемый объект нельзя. Триггер выполняется в рамках текущей транзакции.	SYSEVENT, LOGIN_USER, INSTANCE_NUM, DATABASE_NAME, DICTIONARY_OBJ_TYPE, DICTIONARY_OBJ_NAME, DICTIONARY_OBJ_OWNER
BEFORE DROP, AFTER DROP	В пределах триггера изменять удаляемый объект нельзя. Триггер выполняется в рамках текущей транзакции.	SYSEVENT, LOGIN_USER, INSTANCE_NUM, DATABASE_NAME, DICTIONARY_OBJ_TYPE, DICTIONARY_OBJ_NAME, DICTIONARY_OBJ_OWNER

# Синтаксис триггеров уровня схемы

```
CREATE [ OR REPLACE ] TRIGGER <имя триггера>
  { AFTER LOGON | BEFORE LOGOFF |
    AFTER SERVERERROR |
    AFTER CREATE | BEFORE CREATE |
    AFTER ALTER | BEFORE ALTER |
    AFTER DROP | BEFORE DROP }
  ON <имя_схемы>.SCHEMA
  [ WHEN <условие> ]
[ DECLARE
  -- описание переменных, констант и др. элементов программы
]
BEGIN
  -- программа на процедурном языке (PL/SQL)
  [ EXCEPTION
    -- обработка исключительных ситуаций
  ]
END;
/
```

# Пример триггера уровня схемы

*-- Запрет любым пользователям удалять в схеме SCOTT  
-- таблицы, которые начинаются с EMP*

```
create or replace trigger no_drop_trg
  BEFORE DROP ON SCOTT.SCHEMA
  declare
    v_msg VARCHAR2(1000) := 'No drop allowed on ' ||
      DICTIONARY_OBJ_OWNER || '.' ||
      DICTIONARY_OBJ_NAME || ' from ' ||
      LOGIN_USER;
  begin
    if DICTIONARY_OBJ_OWNER = 'SCOTT' and
      DICTIONARY_OBJ_NAME LIKE 'EMP%' and
      DICTIONARY_OBJ_TYPE = 'TABLE'
    then
      raise_application_error(-20905, v_msg);
    end if;
  end;
/
```

# Отличия триггеров в SQL Server

Типы триггеров:

## 1) Триггер **AFTER**

Триггеры **AFTER** выполняются после выполнения действий инструкции **INSERT**, **UPDATE**, **MERGE** или **DELETE**. Для каждой из операций **INSERT**, **UPDATE** или **DELETE** в указанной инструкции **MERGE** соответствующий триггер вызывается для каждой операции **DML**.

## 2) Триггер **INSTEAD OF**

Триггеры **INSTEAD OF** переопределяют стандартные действия инструкции, вызывающей триггер.

## 3) Триггеры **CLR**

Триггер **CLR** может быть либо триггером **AFTER**, либо триггером **INSTEAD OF**. Триггер **CLR** может также являться триггером **DDL**. Вместо вызова хранимой процедуры на языке **Transact-SQL** триггер **CLR** вызывает один или несколько методов управляемого кода, являющихся членами сборки, созданной с помощью среды **.NET Framework** и загружены в **SQL Server**.