# Oracle Core
# Тема 9
# TRIGGERS

**Петров Александр**
**Senior developer группы разработки Oracle**

**Черный Евгений**
**Team Leader группы разработки Oracle**

# Содержание

- Триггеры уровня инструкций DML
- Compound триггеры DML
- Instead of триггеры
- Crossedition триггеры
- Системные триггеры
- Операции с триггерами
- Права для операций с триггерами

# Триггеры уровня инструкций DML

- **Когда триггер будет запущен**
- до выполнения dml инструкции
- после выполнения dml инструкции

- **Как триггер будет запущен**
- Для целой операции
- Для каждой записи
- Составные триггеры

- **На какие действия будет запущен триггер**
- Вставка записей
- Обновление записей
- Удаление записей

# Триггеры уровня инструкций DML

```
CREATE OR REPLACE TRIGGER trigger_name
    BEFORE INSERT OR UPDATE OF f1, f2 OR DELETE ON table_name
    FOR EACH
    REFERENCING OLD AS OLD NEW AS NEW
    WHEN (old.id < 10)
    DECLARE
        i INT;
BEGIN
    dbms_output.put_line( 'trigger fire!' );
EXCEPTION
    WHEN OTHERS
THEN
        RAISE;
END;
```

trigger before операция
- trigger before операция со строкой
  - операция со строкой
- trigger after операция со строкой
- …
- trigger before операция со строкой
  - операция со строкой
- trigger after операция со строкой

trigger after операция

# Триггеры уровня инструкций DML

```
CREATE OR REPLACE TRIGGER tr_test_1
    BEFORE INSERT ON table_test
BEGIN
    dbms_output.put_line( 'before insert' );
END;


CREATE OR REPLACE TRIGGER tr_test_2
    AFTER INSERT ON table_test
BEGIN
    dbms_output.put_line( 'after insert' );
END;

CREATE OR REPLACE TRIGGER tr_test_3
    BEFORE INSERT ON table_test
    FOR EACH ROW
BEGIN
    IF :old.text IS NULL THEN :new.text := 'change text'; END IF;
    dbms_output.put_line( 'before insert row: '||nvl(:old.text,'null')||' => '||nvl(:new.text,'null'));
END;
```

```
before insert
   before insert row: null => change text
   after insert row: null => change text
   before insert row: null => change text
   after insert row: null => change text
   before insert row: null => change text
   after insert row: null => change text
after insert
```

CRE Ошибка при компиляции:

BEG ORA-04084: невозможно изменить значение NEW для этого типа триггера

END;

```
insert into table_test select rownum, 'ins_' || rownum, sysdate from
dual
    connect by rownum <= 3;
```

# Триггеры уровня инструкций DML

```
CREATE OR REPLACE TRIGGER tr_test_455_after
    AFTER INSERT OR UPDATE ON table_test
    FOR EACH ROW
BEGIN
    IF inserting
    THEN
        dbms_output.put_line( 'after insert: '||nvl(:old.text, 'null')||' => '||nvl(:new.text,'null'));
    ELSIF updating('dt')
    THEN
        dbms_output.put_line( 'after update(dt): '||nvl(:old.text,'null')||' =>
'||nvl(:new.text,'null'));
    ELSIF deleting
        dbms_output.put_line( 'after delete: '||nvl(:old.text,'null')||' => '||nvl(:new.text,'null'));
    END IF;
END;
```

```
update table_test set text = 'upd_2' where id = 1;
update table_test set dt = sysdate, text = 'upd_3' where id = 1;
```

before update: ins_1 => upd_2
before update: upd_2 => upd_3
after update(dt): upd_2 => upd_3

# Триггеры уровня инструкций DML

```
CREATE OR REPLACE TRIGGER tr_test_99
    BEFORE UPDATE ON table_test
    FOR EACH ROW
DECLARE
    v_id NUMBER;
BEGIN
    SELECT id
    INTO   v_id
    FROM   test_schema.test_table1
    WHERE  text = :new.text;

    IF :new.id IS NULL
    THEN
        RAISE_APPLICATION_ERROR(-20001, 'ID is NULL');
    END IF;
    test_schema.test_procedure;
EXCEPTION
    WHEN OTHERS THEN
        log_error;
END;
```

```
CREATE OR REPLACE TRIGGER tr_test_99
    BEFORE UPDATE ON table_test
    FOR EACH ROW
DECLARE
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    logs.ifc_logs.info('trigger fire');
    COMMIT;
END;
```

# Триггеры уровня инструкций DML

```
CREATE OR REPLACE TRIGGER tr_test_1
    AFTER INSERT ON table_test
    FOR EACH ROW
BEGIN
    dbms_output.put_line('tr_test_1 fire');
END;


CREATE OR REPLACE TRIGGER tr_test_2
    AFTER INSERT ON table_test
    FOR EACH ROW follows tr_test_1
BEGIN
    dbms_output.put_line('tr_test_2 fire');
END;
```

```
tr_test_1 fire
tr_test_2 fire
```

# Триггеры уровня инструкций DML

## Инвалидный триггер не даст выполниться операции DML

```
CREATE OR REPLACE TRIGGER tr_test_0
    AFTER UPDATE ON table_test
    FOR EACH ROW follows tr_test_1
disable
BEGIN
    нет_такой_функции;
END;
```

```
insert into table_test values ( 1, 'ins_1', sysdate );
update table_test set text = 'upd_2' where id = 1;
```

# Триггеры уровня инструкций DML

```
CREATE OR REPLACE TRIGGER test_schema1.tr_test
    AFTER UPDATE ON test_schema2.testtable1
    FOR EACH ROW
BEGIN
    dbms_output.put_line( sys_context('USERENV', 'CURRENT_SCHEMA') );
END,
```

TEST_SCHEMA1

# Триггеры уровня инструкций DML

```
CREATE OR REPLACE TRIGGER tr_test_ai
    BEFORE INSERT ON table_test
    FOR EACH ROW
BEGIN
    SELECT COUNT(*)
    INTO   :new.text
    FROM   table_test
    WHERE  id < :new.id;
END;
```

ORA-04091 Таблица TABLE_TEST изменяется, триггер/функция может не заметить это

Обход проблемы чтения изменяемой таблицы:

- использовать триггеры уровня операции
- автономная транзакция в триггере
- использовать сторонние структуры (коллекции уровня пакета)
- Использовать COMPOUND TRIGGER
- **изменение алгоритма**

# Триггеры уровня инструкций DML

```
CREATE OR REPLACE TRIGGER tr_test_ai
    AFTER INSERT ON table_test
    FOR EACH ROW
DECLARE
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    INSERT INTO table_test
    VALUES
        (:new.id + 1,
         'рекурсия',
         NULL);
    COMMIT;
END;
```

ORA-00036 превышено максимальное число рекурсивных уровней SQL

# COMPOUND TRIGGER DML

```
create or replace trigger tr_table_test_compound
    for update or delete or insert on table_test
  compound trigger

  v_count  pls_integer := 0;

  procedure log( p_text in varchar2 ) is
  begin
   dbms_output.put_line( p_text );
  end log;

  before statement is
  begin
   log( 'before statement' );
  end before statement;

  before each row is
  begin
   log( 'before insert: ' || nvl( :old.text, 'null' ) || ' => ' || nvl( :new.text, 'null' ) );
  end before each row;

  after each row is
  begin
   log( 'after insert : ' || nvl( :old.text, 'null' ) || ' => ' || nvl( :new.text, 'null' ) );
   v_count := v_count + 1;
  end after each row;

  after statement is
  begin
   log( 'after statement (' || v_count || ')' );
  end after statement;
end tr_table_test_compound;
```

```
insert into table_test
  select rownum,
         'ins_' || rownum,
         sysdate
  from dual
  connect by rownum <= 3;
```

```
before statement
   before insert: null => ins_1
   after insert : null => ins_1
   before insert: null => ins_2
   after insert : null => ins_2
   before insert: null => ins_3
   after insert : null => ins_3
after statement (3)
```

# INSTEAD OF DML trigger

```
CREATE OR REPLACE TRIGGER tr_test_ai
    AFTER INSERT OR UPDATE ON table_test
    FOR EACH ROW
BEGIN
    IF inserting
    THEN
        dbms_output.put_line('trigger on table fire! INSERT');
    ELSIF updating
    THEN
        dbms_output.put_line('trigger on table fire! UPDATE text=' || :new.text || ' dt=' ||
            to_char(:new.dt, 'yyyy-mm-dd'));
    END IF;
END;


CREATE OR REPLACE TRIGGER tr_test_instead
    INSTEAD OF UPDATE ON vi_table_test
    FOR EACH ROW
BEGIN
    UPDATE table_test
    SET    text = :new.text,
           dt   = :new.dt + 1
    WHERE  id = :new.id;
END;
```

```
trigger on table fire! INSERT
trigger on table fire! UPDATE text=upd_2 dt=2015-01-02
```

# INSTEAD OF DML trigger

```
CREATE OR REPLACE TRIGGER tr_test_instead
    INSTEAD OF UPDATE ON vi_table_test
    FOR EACH ROW
DECLARE
    v_body  xmltype;
    v_nlist dbms_xmldom.domnodelist;
    n       dbms_xmldom.domnode;
    e       dbms_xmldom.domelement;
BEGIN
    SELECT BODY
    INTO   v_body
    FROM   table_test
    WHERE  id = :new.id;
    v_nlist := dbms_xmldom.getelementsbytagname(dbms_xmldom.newdomdocument(v_body), 'person');

    FOR i IN 0 .. dbms_xmldom.getlength(v_nlist) - 1
    LOOP
        n := dbms_xmldom.item(v_nlist, i);
        e := dbms_xmldom.makeelement(n);
        IF dbms_xmldom.getnodevalue(dbms_xmldom.getfirstchild(n)) = :new.name
        THEN
            dbms_xmldom.setattribute(e, 'phone', :new.phone);
        END IF;
    END LOOP;

    UPDATE table_test
    SET    BODY = v_body
    WHERE  id = :new.id;
END;
```

| | ID | NAME | PHONE |
|---|---|---|---|
| 1 | 1 | Черный | 9999 |
| 2 | 1 | Бобович | 7890 |
| 3 | 1 | Зацепин | 5712 |

```
update vi_table_test set phone = '9999' where id=1 and name = 'Черный';
```

# Crossedition Triggers

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

# Триггеры уровня инструкций DML

Ограничения триггеров:

- нельзя выполнять DDL statements (только AT)
- нельзя запускать подпрограммы с операторами контроля транзакций
- не имеет доступа к SERIALLY_REUSABLE пакетов
- размер не может превышать 32K
- нельзя декларировать переменные типа LONG и LONG RAW
- мутирование таблицы

# Триггеры уровня инструкций DML

- Триггеры удобно использовать для дополнительных проверок.
- Возможность изменения или добавления значений полей.
- Возможность изменить алгоритм в одном месте для всего множества операций.
- Логирование, возможность «промежуточного» слоя между пользователями и БД.
- Прозрачная корректировка входящих данных.

- Каждый триггер замедляет операции DML с таблицей до 30%.
- Трудность отладки, усложнение поиска ошибок.
- Вымывание функциональности из основных алгоритмов.
- Инвалидный триггер – невозможность операции.
- Триггер BEFORE генерит дополнительные REDO для update.

# Системные триггеры

Системные триггеры подразделяются:

- **SCHEMA Triggers**
- **DATABASE Triggers**
- **INSTEAD OF CREATE Triggers**

События срабатывания системных триггеров:

- **DDL**
- **System event**

# Системные триггеры (schema)

```
CREATE OR REPLACE TRIGGER testschema.tr_ddl_0
    BEFORE drop ON testschema1.schema
BEGIN
    dbms_output.put_line('trigger [before drop] fire!');
END;


CREATE OR REPLACE TRIGGER testschema.tr_ddl_2
    AFTER CREATE ON SCHEMA
BEGIN
    dbms_output.put_line('trigger [after create] fire!');
END;
```

trigger [after create] fire!

trigger [after create] fire!

# Системные триггеры (schema)

```
CREATE OR REPLACE TRIGGER tr_ddl_03
    BEFORE ddl ON SCHEMA
BEGIN
    dbms_output.put_line( 'owner= ' || ora_dict_obj_owner);
    dbms_output.put_line( 'name = ' || ora_dict_obj_name);
    dbms_output.put_line( 'type = ' || ora_dict_obj_type);
    dbms_output.put_line( 'event= ' || ora_sysevent);

    IF ora_dict_obj_name = 'TABLE_INCORRECT'
    THEN
        RAISE_APPLICATION_ERROR(- 20001, 'Таблица не может быть
создана');
    END IF;
END;
```

ORA-20001 'Таблица не может быть создана'

# Системные триггеры (schema)

```
CREATE OR REPLACE TRIGGER t_log_ddl
    BEFORE ddl ON DATABASE
DECLARE
    PRAGMA AUTONOMOUS_TRANSACTION;
    sql_text_list ora_name_list_t;
    sql_text      VARCHAR2(2000 CHAR);
    n             NUMBER;
    i             NUMBER;
    curr_time     TIMESTAMP WITH TIME ZONE;
BEGIN
    IF ora_dict_obj_type = 'MATERIALIZED VIEW' OR ora_dict_obj_type = 'SNAPSHOT'
    THEN
        RETURN;
    END IF;

    n := nvl(ora_sql_txt(sql_text_list), 0);
    FOR i IN 1 .. n
    LOOP
        sql_text := substr(sql_text || sql_text_list(i), 0, 2000);
    END LOOP;
    curr_time := systimestamp;

    INSERT INTO log_ddl
    VALUES
        (curr_time,
         ora_login_user,
         sys_context('USERENV', 'OS_USER'),
         sys_context('USERENV', 'HOST'),
         sys_context('USERENV', 'IP_ADDRESS'),
         sql_text);
    COMMIT;
END;
```

# Системные триггеры (database)

```
CREATE OR REPLACE TRIGGER tr_ddl_05
    AFTER grant ON DATABASE
DECLARE
    ngl          NUMBER;
    npl          NUMBER;
    v_grant_type VARCHAR2(30);
    v_grant_list ora_name_list_t;
    v_priv_list  ora_name_list_t;
BEGIN
    v_grant_type := ora_dict_obj_type;
    ngl          := ora_grantee(v_grant_list);
    npl          := ora_privilege_list(v_priv_list);

    IF v_grant_type = 'OBJECT PRIVILEGE'
    THEN
        dbms_output.put_line('grants:');

        FOR i IN 1 .. ngl
        LOOP
            dbms_output.put_line(v_grant_list(i) || ' ' || v_priv_list(i));
        END LOOP;
    END IF;
END;
```

grants:
TESTUSER SELECT

# Системные триггеры (database)

```
CREATE OR REPLACE TRIGGER tr_ddl_06
    BEFORE drop ON DATABASE
BEGIN
    RAISE_APPLICATION_ERROR(-20001, 'Ot takaya zagogulina!');
END;
```

Complete!

# Системные триггеры (INSTEAD OF CREATE )

```
CREATE OR REPLACE TRIGGER tr_instead_of
    INSTEAD OF CREATE ON SCHEMA
BEGIN
    log(ora_dict_obj_owner || '.' || ora_dict_obj_name);
END;
```

DBADMIN.TEST_TABLE

# Системные триггеры (events)

*События срабатывания триггеров (**events**):*

- ***AFTER STARTUP***
- ***BEFORE SHUTDOWN***
- ***AFTER DB_ROLE_CHANGE***
- ***AFTER SERVERERROR***
- ***AFTER LOGON***
- ***BEFORE LOGOFF***
- ***AFTER SUSPEND***

- ***BEFORE SERVERERROR*** к сожалению пока не поддерживается

# Системные триггеры (events)

```
CREATE OR REPLACE TRIGGER tr_event_startup
    AFTER startup ON DATABASE
BEGIN
     RAISE_APPLICATION_ERROR(- 20001, 'Don’t start!');
END;


CREATE OR REPLACE TRIGGER tr_event_shutdown
    BEFORE shutdown ON DATABASE
BEGIN
    release_some_collect;
END;

CREATE OR REPLACE TRIGGER tr_event_logon
    AFTER logon ON test.schema
BEGIN
    RAISE_APPLICATION_ERROR(-20001, ’TEST, go home!');
END;


CREATE OR REPLACE TRIGGER tr_event_logoff
    BEFORE logoff ON DATABASE
BEGIN
    log;
END;
```

# Системные триггеры (events)

```
CREATE OR REPLACE TRIGGER event_after_ servererror
    AFTER servererror ON DATABASE
BEGIN
    IF is_servererror(1476)
    THEN
        dbms_output.put_line('fire [error 1476]');
    END IF;
END;
```

fire [error 1476]

\+ ОШИБКА

# Системные триггеры (events)

```
CREATE OR REPLACE TRIGGER tr_event_suspend
    AFTER suspend ON DATABASE
DECLARE
    v_error_type       VARCHAR2(100);
    v_object_type      VARCHAR2(100);
    v_object_owner     VARCHAR2(100);
    v_table_space_name VARCHAR2(100);
    v_object_name      VARCHAR2(100);
    v_sub_object_name  VARCHAR2(100);
BEGIN
    IF dbms_resumable.space_error_info(error_type       => v_error_type,
                                       object_type      => v_object_type,
                                       object_owner     => v_object_owner,
                                       table_space_name => v_table_space_name,
                                       object_name      => v_object_name,
                                       sub_object_name  => v_sub_object_name)
    THEN

        IF v_error_type = 'SPACE QUOTA EXCEEDED'
           AND v_object_type = 'TABLE SPACE'
        THEN
            send_sms(sys_context('USERENV', 'CURRENT_USER')); --Пинаем админа, увеличим квоту
        END IF;
    END IF;
END;
```

# Операции с триггерами

✓ Отключение триггеров:

```
alter trigger TRIGGER_NAME disable;
```

✓ Включение триггеров:

```
alter trigger TRIGGER_NAME enable;
```

✓ Включение | отключение всех триггеров на таблице:

```
alter table TABLE_NAME enable | disable all triggers;
```

✓ Компиляция триггеров:

```
alter trigger TRIGGER_NAME compile;
```

✓ Информация о триггерах: **dba_triggers**
✓ Код там же, где и всё: **dba_source**
✓ А вот его валидность смотреть: **dba_objects**

# Права для операций с триггерами

- `grant create trigger to USER;`

- `grant create any trigger to USER;`
- `grant alter any trigger to USER;`
- `grant drop any trigger to USER;`

- `grant ADMINISTER DATABASE TRIGGER to USER;`

# Summarizing

Триггер это именованный программный модуль, который хранится в базе данных и срабатывает в ответ на определенной событие.

**Событие может быть связано с таблицей, представлением, схемой, или базой данных**:

- DML заявление (DELETE, INSERT или UPDATE)
- DDL заявление (CREATE, ALTER или DROP)
- Операции базы данных (SERVERERROR, LOGON, LOGOFF, STARTUP или SHUTDOWN)

**Триггеры используют для:**

- ре… ь чер…
- ор…
- оп… содержащейся в БД
- реализации бизнес логики
- организации каскадных воздействий на таблицы БД
- отклика на системные события в БД или схеме

1. **Триггеры – полезная вещь!**
2. **Можешь обойтись без них – не применяй!**

**Триггеры дают разработчику широкий спектр возможностей по реализации логики в БД, но требуют аккуратного использования из-за своего воздействия на быстродействие SQL операторов и системных событий.**

# Список использованных материалов


1. PL/SQL Triggers (oracle documentation)
2. Create trigger (oracle documentation)
3. Alter trigger (oracle documentation)
4. Drop trigger (oracle documentation)
5. Crossedition triggers (oracle documentation)