Триггеры

- Триггер (trigger) — это хранимая процедура особого типа, которая не вызывается непосредственно, а исполнение которой обусловлено наступлением одного из событий, относящегося к одной конкретной таблице (представлению), или наступлению одного из событий базы данных.

DML триггеры (на таблицу или представление)

- DML триггеры выполняются на уровне строки (записи) каждый раз, когда изменяется образ строки.
- Они могут быть определены и для таблиц и представлений
- DML триггеры могут создать:
  - Администраторы (SYSDBA и/или владелец схемы БД)
  - Владелец таблицы (представления)
  - Пользователи с привилегией ALTER ANY {TABLE | VIEW}.

Синтаксис

```
CREATE [OR ALTER] TRIGGER имя FOR таблица
    {ACTIVE|INACTIVE}
    {BEFORE|AFTER}{DELETE|INSERT|UPDATE}
    [POSITION число]
    AS
    [DECLARE локальные переменные
    . . .]
    BEGIN
    операторы
    END^
```

Совместимость с SQL 2003

```
CREATE TRIGGER имя
    {ACTIVE|INACTIVE}
    {BEFORE|AFTER}
    {DELETE|INSERT|UPDATE}
    [POSITION число]
    ON таблица
AS
. . .
```

Атрибут состояния триггера

ACTIVE | INACTIVE

Перевод триггера из одного состояния в другое осуществляется командой ALTER TRIGGER.

```
alter trigger TR_DU for TABLE_1  inactive;
```

В неактивном состоянии – сохраняется в БД, но не срабатывает

Операции, активирующие триггер DML

INSERT | UPDATE | DELETE

Несколько операций

```
create trigger TR_DU for TABLE_1
        before DELETE OR UPDATE
```

логические контекстные переменные для проверки

INSERTING, UPDATING, DELETING

Пример

```
CREATE OR ALTER trigger . . .
active before insert or update or delete
position 0
AS
BEGIN
...
if (inserting  or  updating) then
  begin
    if (new.SERIAL_NUM is null) then
      new.SERIAL_NUM = gen_id(GEN_SERIALS, 1);
  end
...
END
```

Фазы выполнения

BEFORE | AFTER

Объединение фаз BEFORE и AFTER недопустимо, т.к. триггеры разных фаз имеют разные возможности

команда	before	constraints	after	завершение
insert/update/delete	Триггеры before	проверка	Триггеры after	commit/rollback
	exception	нарушение	exception	
	rollback	rollback	rollback	

Процесс выполнения операции

- ✓Формирование данных и передача серверу (посылка)
- ✓При получении посылки активируются триггеры фазы BEFORE
- ✓Если триггеры завершились успешно, проверка ограничений (CONSTRAINTS)
- ✓Если все ограничения проверены успешно, активируются триггеры фазы AFTER
- ✓Если триггеры завершились успешно, сохранение версии данных
- ✓Для фиксации версии данных COMMIT
- ✓Если на любом этапе триггеры или ограничения вызывают исключение транзакция помечается как ROLLED BACK

Контекстные переменные NEW и OLD

Содержат состояние строки, подвергающейся модификации оператором INSERT, UPDATE или DELETE (там, где это имеет смысл)

OLD.* - до выполнения операции модификации

NEW.* - после

Операторы могут ссылаться на них, используя следующие формы

NEW.*columnname*

И

OLD.*columnname*

Правила для NEW и OLD

- Во всех триггерах контекстные переменные OLD доступны только для чтения;
- В триггерах BEFORE UPDATE и BEFORE INSERT переменные NEW доступны для чтения и
  записи, за исключением COMPUTED BY столбцов;
- В INSERT триггерах ссылка на переменные OLD не допускается и вызовет исключение;
- В DELETE триггерах ссылка на переменные NEW не допускается и вызовет исключение;
- Во всех AFTER триггерах переменные NEW доступны только для чтения.

Вывод

В триггерах фазы BEFORE контекстная переменная NEW для операций INSERT и UPDATE может быть подвергнута изменению.

Именно измененное значение будет использовано при проверки ограничений (constraints) и для завершения выполнения операции.

Индикатор последовательности запуска

POSITION n

Для каждой комбинации фаза-событие может быть определено более одного триггера.

Порядок, в котором они выполняются, может быть указан явно с помощью дополнительного аргумента POSITION в определении триггера.

Максимальная позиция равна 32767.

Триггеры с меньшей позицией вызываются первыми.

Если предложение POSITION опущено или несколько триггеров с одинаковыми фазой и событием имеют одну и ту же позицию, то такие триггеры будут выполняться в алфавитном
порядке их имен.

Тело триггера

- Тело триггера как и тело процедуры состоит из объявления локальных переменных и составного блока операторов

AS
  [<*declarations*>]
BEGIN
  [<*PSQL_statements*>]
END

В блоке операторов допустимы все конструкции, которые допустимы в процедурах.

Следует контролировать отсутствие рекурсии в теле триггера

Использование триггеров

- для проверки и исправления данных, нарушающих ограничение целостности;
- для отмены операций модификации, противоречащих ограничениям целостности в виде требований деловых правил;
- для замены операций модификации данных в таблицах вызовом хранимых процедур, проверяющих и поддерживающих деловые правила;
- для протоколирования событий

Пример – автоматическое исправление

```
create trigger BIU_AGENT for AGENT
   before insert  or update position 0
as
begin
   NEW.NAME_AG = UPPER(NEW.NAME_AG);
end
```

Пример – запрет ошибочных действий

```
create trigger AU_EMP for EMPLOYEE
   after update position 0
as
begin
   if (NEW.SALARY < OLD.SALARY) then
         exception ERROR_PAY;
end
```

Пример –автоматическая нумерация

```
create trigger BI_OPERATION for OPERATION
   before insert
as
begin
   if (NEW.ID is NULL) then
        NEW.ID=GEN_ID(GEN_OPID,1);
end
```

Изменение другой таблицы

```
create trigger BI_OPER for OPERATION
    before insert position 0
as
begin
 if (NEW.TYPEOP='R')
  update GOODS_WH  T
       set  T.QUANTITY = T.QUANTITY – NEW.QUANTITY
   where       T.ID_WH        = NEW.ID_WH and
               T.ID_GOODS = NEW.ID_GOODS
. . .
end
```

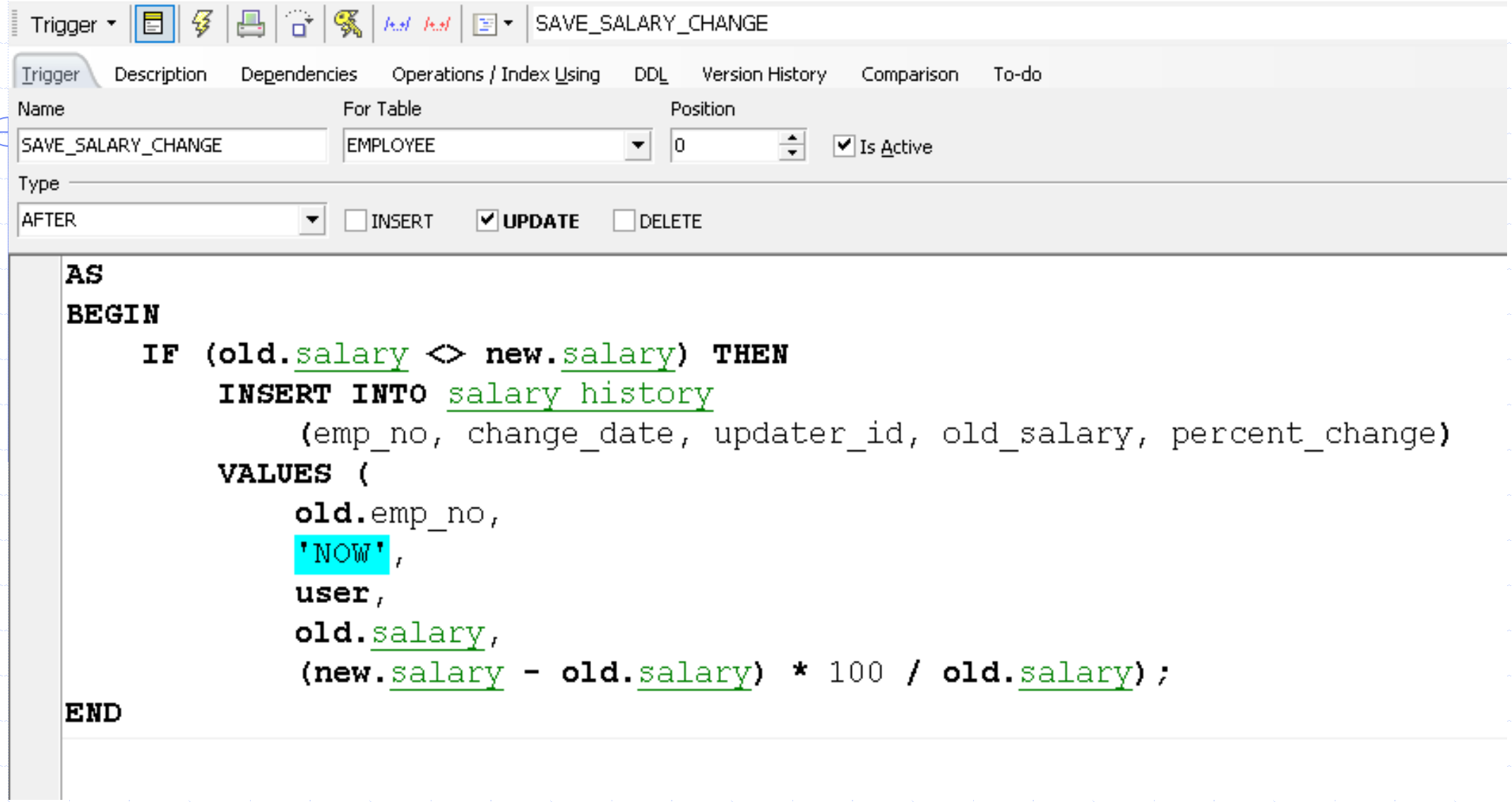
БД employee

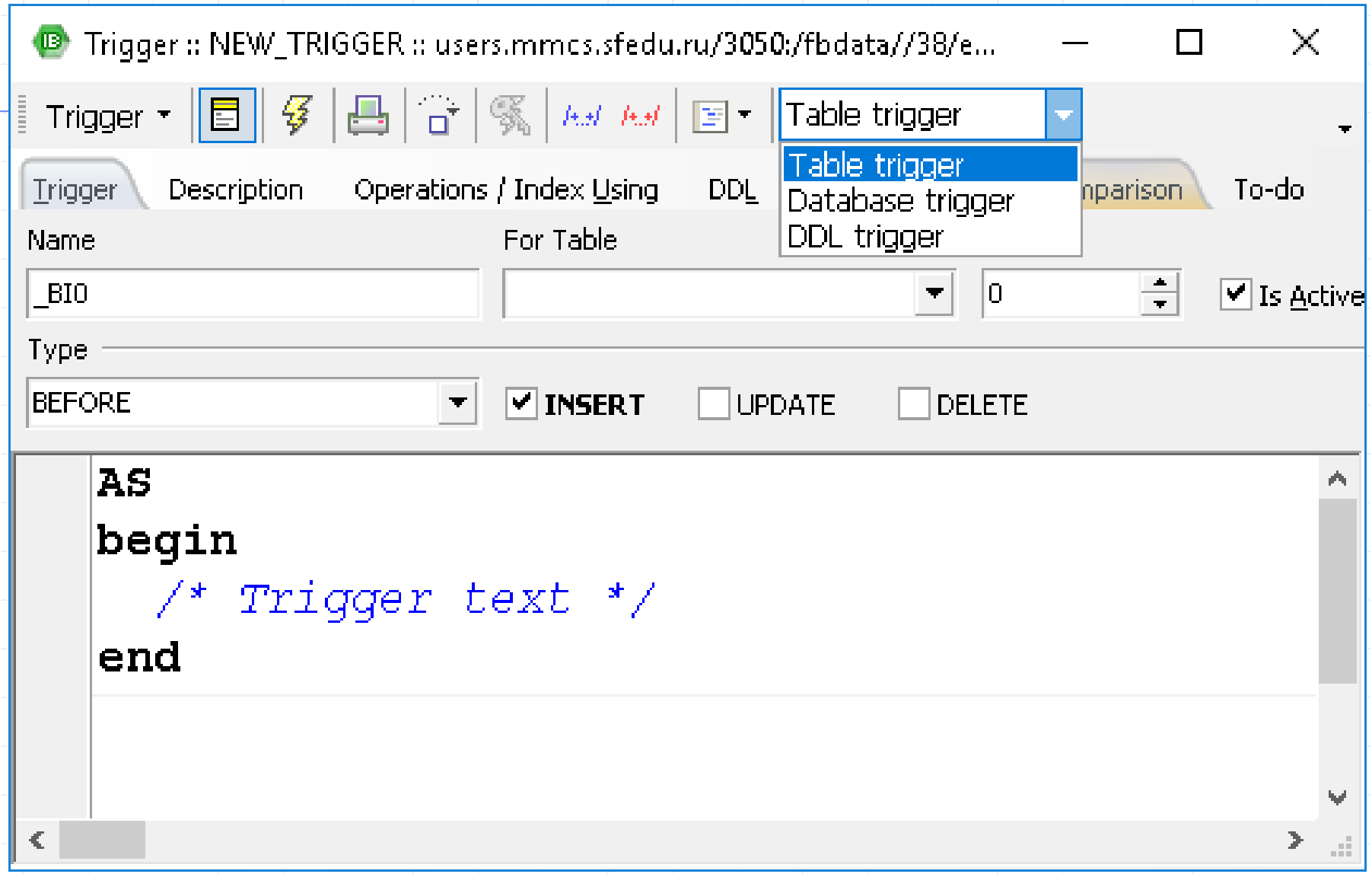
```
CREATE OR ALTER trigger save_salary_change for employee
active after update position 0   - - почему after update ?
AS
BEGIN
    IF (old.salary <> new.salary) THEN
        INSERT INTO salary_history
            (emp_no, change_date, updater_id, old_salary, percent_change)
        VALUES (old.emp_no,   'NOW',  user,
           old.salary,
           (new.salary - old.salary) * 100 / old.salary);
END
```

Журнализация событий

```
create trigger AI_GOODS for GOODS
ACTIVE AFTER INSERT
POSITION 1
as
declare variable tid integer;
begin
  tid = gen_id(log_table_gen,1);
  insert into log_table (id, operation, date_time, user_name)
  values (:tid, 'Добавился новый товар', 'NOW',
                                user);
end
```

```
create trigger TR_CUST_LOG
active after insert or update or delete position 10 on CUSTOMER
as
begin
  insert into CHANGE_LOG (LOG_ID, ID_TABLE, TABLE_NAME, MUTATION)
  values (next value for SEQ_CHANGE_LOG, old.CUST_NO, 'CUSTOMER',
          case
          when inserting then 'INSERT'
          when updating then 'UPDATE'
          when deleting then 'DELETE'
          end
          );
end
```


Trigger
SAVE_SALARY_CHANGE
Trigger
Description
Dependencies
Operations / Index Using
DDL
Version History
Comparison
To-do
Name
For Table
Position
SAVE_SALARY_CHANGE
EMPLOYEE
0
Is Active
Type
AFTER
INSERT
UPDATE
DELETE
AS
BEGIN
IF (old.salary <> new.salary) THEN
INSERT INTO salary_history
(emp_no, change_date, updater_id, old_salary, percent_change)
VALUES (
old.emp_no,
'NOW',
user,
old.salary,
(new.salary - old.salary) * 100 / old.salary);
END


Trigger :: NEW_TRIGGER :: users.mmcs.sfedu.ru/3050:/fbdata//38/e...
Trigger
Table trigger
Table trigger
Database trigger
DDL trigger
Trigger
Description
Operations / Index Using
DDL
nparison
To-do
Name
For Table
_BIO
0
Is Active
Type
BEFORE
INSERT
UPDATE
DELETE
AS
begin
/* Trigger text */
end

Триггеры базы данных

- Начиная с версии 2.1
- События базы данных
  - CONNECT
  - DISCONNECT
  - TRANSACTION START
  - TRANSACTION COMMIT
  - TRANSACTION ROLLBACK
- Права – только SYSDBA или владелец базы
- Указать для триггера несколько событий базы данных невозможно

Синтаксис

CREATE TRIGGER *имя*
[ACTIVE | INACTIVE]
ON *событиеБД*
[POSITION *number*]
AS
[*<declarations>*]
BEGIN
  [*<statements>*]
END

Пример

```
create trigger tr_connect active
on connect
as
begin
   in autonomous transaction
    insert into dblog
           values (current_user,
                   current_timestamp,
                 'connect');
end
```

Пример

```
create exception E_INCORRECT_WORKTIME
          'Рабочий день ещё не начался';
. . .
create trigger TR_LIMIT_WORKTIME
active on connect position 1
as
begin
  if ((current_user <> 'SYSDBA') and
      not(current_time between time '9:00' and time '17:00')) then
    exception E_INCORRECT_WORKTIME;
end
```

!!!

Вы не сможете подключиться к базе данных в случае исключения в триггере на событие CONNECT, а также отменяется старт транзакции при исключении в триггере на событие TRANSACTION START.

В обоих случаях база данных эффективно блокируется до тех пор, пока вы не отключите триггеры базы данных и не исправите ошибочный код

DDL триггеры

- Триггеры на события изменения метаданных (DDL триггеры) предназначены для обеспечения ограничений, которые будут распространены на пользователей, которые пытаются создать, изменить или удалить объект схемы БД.
- Другое их назначение — ведение журнала изменений метаданных
- DDL триггеры срабатывают на указанные события изменения метаданных в одной из фаз события.
  - BEFORE триггеры запускаются до изменений в системных таблицах.
  - AFTER триггеры запускаются после изменений в системных таблицах.

- Переменные доступные в пространстве имён DDL_TRIGGER (испоьзуются в теле триггера):

  EVENT_TYPE — тип события (CREATE, ALTER, DROP)
  OBJECT_TYPE — тип объекта (TABLE, VIEW и д.р.)
  DDL_EVENT — имя события (*<ddl event item>*),
  где *<ddl event item>* = EVENT_TYPE || ' ' || OBJECT_TYPE
  OBJECT_NAME — имя объекта метаданных
  SQL_TEXT — текст SQL запроса

- Триггеры на события изменения метаданных могут создать:
  - Администраторы
  - Владелец базы данных;
  - Пользователи с привилегией ALTER DATABASE

Пример

```
create exception E_INVALID_SP_NAME
'Неверное имя хранимой процедуры (должно начинаться с SP_)';
. . .
create trigger TRIG_DDL_SP before create procedure
as
begin
  if (rdb$get_context('DDL_TRIGGER', 'OBJECT_NAME')
          not starting 'SP_') then
    exception E_INVALID_SP_NAME;
end
```

Пример

```
create exception E_ACCESS_DENIED 'Access denied';
. . .
create trigger TRIG_DDL before any ddl statement
as
begin
  if (current_user <> 'SUPER_USER') then
    exception E_ACCESS_DENIED;
end
```

!!!

- В Firebird существуют привилегии на DDL операторы, поэтому прибегать к написанию DDL триггера нужно только в случае, если того же самого эффекта невозможно достичь стандартными методами

Триггеры и транзакции

- Действия триггера выполняются в рамках транзакции, в которой выполнено действие, запустившие триггер
- Если транзакция будет завершена откатом (rollback), то будут откачены и операция, запустившая триггер и все действия, выполненные в триггере
- Если необходимо сохранить действия в триггере, независимо от результата транзакции, эти действия должны быть выполнены в автономной транзакции. Например, действия журнализации.