Министерство образования и науки Российской Федерации

Калужский филиал
федерального государственного бюджетного образовательного
учреждения высшего образования
**«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»**
(КФ МГТУ им. Н.Э. Баумана)

С.А. Глебов

ИСПОЛЬЗОВАНИЕ ТРИГГЕРОВ

Методические указания по выполнению лабораторной работы
по курсу «Базы данных»

Калуга – 2018

УДК 004.65
ББК 32.972.134
Г53

Методические указания составлены в соответствии с учебным планом КФ МГТУ им. Н.Э. Баумана по направлению подготовки 09.03.04 «Программная инженерия» кафедры «Программного обеспечения ЭВМ, информационных технологий и прикладной математики».

Методические указания рассмотрены и одобрены:

- Кафедрой «Программного обеспечения ЭВМ, информационных технологий и прикладной математики» (ФН1-КФ) протокол № 7 от «21» февраля 2018 г.

И.о. зав. кафедрой ФН1-КФ _____ к.т.н., доцент Ю.Е. Гагарин

- Методической комиссией факультета ФНК протокол № 2 от «28» 02 2018 г.

Председатель методической комиссии факультета ФНК _____ к.х.н., доцент К.Л. Анфилов

- Методической комиссией КФ МГТУ им.Н.Э. Баумана протокол №2 от «06» 03 2018 г.

Председатель методической комиссии КФ МГТУ им.Н.Э. Баумана _____ д.э.н., профессор О.Л. Перерва

Рецензент:
к.т.н., доцент кафедры ЭИУ6-КФ _____ А.Б. Лачихина

Авторы
к.ф.-м.н., доцент кафедры ФН1-КФ _____ С.А. Глебов

Аннотация

Методические указания по выполнению лабораторной работы по курсу «Базы данных» содержат руководство по созданию триггеров в базах данных, а также задание на выполнение лабораторной работы.

Предназначены для студентов 3-го курса бакалавриата КФ МГТУ им. Н.Э. Баумана, обучающихся по направлению подготовки 09.03.04 «Программная инженерия».

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Настоящие методические указания составлены в соответствии с программой проведения лабораторных работ по курсу «Базы данных» на кафедре «Программное обеспечение ЭВМ, информационные технологии и прикладная математика» факультета фундаментальных наук Калужского филиала МГТУ им. Н.Э. Баумана.

Методические указания, ориентированные на студентов 3-го курса направления подготовки 09.03.04 «Программная инженерия», содержат руководство по созданию триггеров в базах данных и задание на выполнение лабораторной работы.

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ ВЫПОЛНЕНИЯ

Целью выполнения лабораторной работы является сформировать практические навыки разработки триггеров.

Основными задачами выполнения лабораторной работы являются:

- разработать триггер для реализации автоинкремента
- разработать триггер для обеспечения каскадных воздействий
- разработать триггер для обеспечения бизнес-правил
- разработать триггер для журнализации

Результатами работы являются:

- Разработанные триггеры.
- Подготовленный отчет.

ТРИГГЕРЫ

Триггер — это особый вид хранимой процедуры, автоматически вызываемой SQL-сервером при обновлении, удалении или добавлении новой записи. Непосредственно вызвать триггер нельзя. Нельзя и передавать им входные параметры и получать от них значения выходных параметров. Триггеры всегда реализуют действие.

Триггер всегда привязан к определенной таблице.

По событию ТБД триггеры различаются на вызываемые при:

- добавлении новой записи;
- изменении существующей записи;
- удалении записи;
- универсальные

По отношению к событию, влекущему их вызов, триггеры различаются на:

- выполняемые до наступления события;
- выполняемые после наступления события.

В клоне Yaffil 1.0 реализована поддержка универсальных триггеров, срабатывающих при любой операции.

Преимущества использования триггеров:

- Автоматическое обеспечение каскадных воздействий в дочерних таблицах при изменении, удалении записи в родительской таблице выполняется на сервере. Пользователю нет необходимости заботиться о программной реализации каскадных воздействий. Поскольку каскадные воздействия выполняет сервер, нет необходимости пересылать изменения в таблицах БД из приложения на сервер, что снижает загрузку сети.
- Изменения в триггерах не влекут необходимости изменения программного кода в клиентских приложениях и не требуют распространения новых версии клиентских приложений.

Замечание. При откате транзакции откатываются также и все изменения, внесенные в БД триггерами.

Триггер создается оператором

CREATE TRIGGER ИмяТриггера FOR ИмяТаблицы
[ACTIVE | INACTIVE]
{BEFORE | AFTER}
{DELETE | INSERT | UPDATE)
[POSITION номер]
AS <тело триггера>

– ACTIVE | INACTIVE — указывает, активен триггер или нет. Можно определить триггер «про запас», установив для него INACTIVE. В дальнейшем можно переопределить триггер как активный. По умолчанию действует ACTIVE.
– BEFORE | AFTER — указывает, будет выполняться триггер до (BEFORE) или после (AFTER) запоминания изменений в БД.
– DELETE | INSERT | UPDATE — указывает операцию над ТБД, при выполнении которой срабатывает триггер.
– POSITION номер — указывает, каким по счету будет выполняться триггер в случае наличия группы триггеров, обладающих одинаковыми характеристиками операции и времени (до, после операции) вызова триггера. Значение номера задается числом в диапазоне 0..32767. Триггеры с меньшими номерами выполняются раньше.

Например, если определены триггеры
CREATE TRIGGER A FOR Movie BEFORE INSERT POSITION 1 ...
CREATE TRIGGER C FOR Movie BEFORE INSERT POSITION O...
CREATE TRIGGER D FOR Movie BEFORE INSERT POSITION 44 ...
CREATE TRIGGER B FOR Movie AFTER INSERT POSITION 100 ...
CREATE TRIGGER E FOR Movie AFTER INSERT POSITION 44 ...
для операции добавления новой записи в таблицу MOVIE они будут выполнены в последовательности C, A, D, E, B.

Для определения тела триггера используется процедурный язык. В него добавляется возможность доступа к старому и новому значениям столбцов изменяемой записи OLD и NEW — возможность, недоступная при определении тела хранимых процедур. Структура тела триггера:

[<объявление локальных переменных>]
BEGIN

```
  < оператор>
END
```

Заголовок триггера имеет формат

```
CREATE TRIGGER ИмяТриггера FOR ИмяТаблицы
[ACTIVE | INACTIVE]
{BEFORE | AFTER}
{DELETE | INSERT | UPDATE)
[POSITION номер]
```

Значение OLD.ИмяСтолбца позволяет обратиться к состоянию столбца, имевшему место до внесения возможных изменений, а значение NEW.ИмяСтолбца — к состоянию столбца после внесения изменений.

В том случае, если значение в столбце не изменилось, OLD.ИмяСтолбца будет равно NEW.ИмяСтолбца.

Следующий триггер, используя генератор, присваивает уникальное значение ключевому полю ID_T, но только в том случае, если пользователь сам не указал значения идентификатора:

```
CREATE TRIGGER TRIG_TOVAR_BI FOR TOVAR
ACTIVE BEFORE INSERT POSITION 0
AS
begin
  IF (NEW."ID_T" IS NULL) THEN NEW."ID_T" =
      GEN_ID ("GEN_TOVAR_ID",1);
end
```

Теперь оператор добавления записей может выглядеть следующим образом:

```
INSERT INTO PROD (ID_PARENT, NAME, PRICE)
VALUES (14, "Ушанка", 20);
```

Здесь не указано значение первичного ключа. Перед вставкой этой записи сработает триггер TRIG_TOVAR_BI, который подставит новое значение генератора в поле ID_T, т.к. его значение не указано (NULL).

Использование контекстных переменных иллюстрирует следующая таблица:

	NEW	OLD
BEFORE INSERT	Full Access	No access
AFTER INSERT	Read Only	No access
BEFORE UPDATE	Full Access	Read Only
AFTER UPDATE	Read Only	Read Only
BEFORE DELETE	No access	Read Only
AFTER DELETE	No access	Read Only

Бизнес-правило: в таблице товаров изменять цену можно только для конечных товаров (листьев дерева) и при этом цена всех родительских товаров должна изменяться автоматически. Написать триггеры, обеспечивающие эти правила.

```
CREATE EXCEPTION "NoPriceForParent"
'Нельзя изменять цену для родительского узла';
CREATE TRIGGER "No_Update_For_Parent" FOR TOVAR
ACTIVE BEFORE UPDATE POSITION 0
AS
begin
  if (
    (NEW."Price"<>OLD."Price")
    AND
    ...
    AND
    EXISTS (select * from tovar
          where tovar."ID_PARENT"=OLD."ID_T")
    )
 then
    EXCEPTION "NoPriceForParent";
end
```

Триггер отслеживает изменение только цены (смена, например, наименования бизнес-правилами никак не оговаривается) и проверяет существование таких записей, для которых наша текущая запись является предком. При совместном соблюдении этих двух условий

вызывается предварительно созданное исключение, которое откатывает изменение записи.

Триггер для каскадного изменения цены родительских объектов:

```
CREATE TRIGGER "Change_Price" FOR TOVAR
ACTIVE AFTER UPDATE POSITION 0
AS
DECLARE VARIABLE delta INTEGER;
DECLARE VARIABLE tmp INTEGER;
begin
  delta=NEW."Price"-OLD."Price";
  if (delta<>0) then
   begin
     select "Price"
     From Tovar
     Where ID_T=OLD."ID_PARENT"
     into :tmp;
     ...
     update tovar set "Price"=:tmp+:delta
        where "ID_T"=OLD."ID_PARENT";
   end
end
```

Триггер срабатывает после изменения цены конечного товара, вычисляет разницу (delta) и если цена изменилась, то в переменной tmp сохраняется цена родительского товара, а затем изменяет ее на величину delta.

Здесь, последний оператор UPDATE наталкивается на действие триггера NoPriceForParent, который запрещает изменение цены родительских узлов. Первое, что напрашивается, это изменение активности триггера NoPriceForParent:

```
ALTER TRIGGER "No_Update_For_Parent" INACTIVE;
```

но, это оператор относится к DDL и не может быть использован в ХП и триггерах.

Управлять состоянием активности триггеров можно недокументированным способом путем модификации системных таблиц:

```
UPDATE rdb$triggers trg
SET trg.rdb$trigger_inactive=1
WHERE trg.rdb$trigger_name='No_Update_For_Parent';
```

Однако, для нашей задачи этот метод тоже не подойдет. Дело в том, что триггеры работают в рамках той же транзакции, что и вызвавшее их изменение. Поэтому, если один триггер изменит состояние другого, то механизм "активных таблиц" который занимается запуском триггеров не увидит эти изменения, т.к. они еще не подтверждены.

В качестве решения этой проблемы можно предложить исключение из бизнес-правила: цену родительского объекта можно изменять только на значение -1 и со значения -1. Используя эту "дыру" триггеры будут выглядеть:

```
CREATE TRIGGER "Change_Price" FOR TOVAR
ACTIVE AFTER UPDATE POSITION 0
AS
DECLARE VARIABLE delta INTEGER;
DECLARE VARIABLE tmp INTEGER;
begin
  delta=NEW."Price"-OLD."Price";
  if (delta<>0) then
   begin
     select "Price"
     From Tovar
     Where ID_T=OLD."ID_PARENT"
     into :tmp;
     update tovar set "Price"=-1 where "ID_T"=OLD."ID_PARENT";
     update tovar set "Price"=:tmp+:delta
        where "ID_T"=OLD."ID_PARENT";
   end
end

CREATE TRIGGER "No_Update_For_Parent" FOR TOVAR
ACTIVE BEFORE UPDATE POSITION 0
AS
begin
```

```
  if (
    (NEW."Price"<>OLD."Price")
    AND
    not (OLD."Price"=-1 OR NEW."Price"=-1)
    AND
    EXISTS (select * from tovar
        where tovar."ID_PARENT"=OLD."ID_T")
    )
  then
    EXCEPTION "NoPriceForParent";
end
```

Журнал изменений в БД представляет собой таблицу БД, в которой фиксируются действия над всей базой данных или отдельными ее таблицами. В многопользовательских системах ведение такого журнала позволяет определить источник недостоверных или искаженных данных.

Определим в базе данных таблицу MOVIE_LOG:

```
CREATE TABLE "MOVIE_Log" (
    "Time"       TIMESTAMP,
    "User"       VARCHAR(31),
    "Column"     VARCHAR(15),
    "Old_Value"  VARCHAR(100),
    "New_Value"  VARCHAR(100),
    "Action"  VARCHAR(3) CHECK ("Action" IN ('INS', 'UPD', 'DEL')),
    "ID_M"       INTEGER,
    CONSTRAINT "FK_Log" FOREIGN KEY (ID_M)
    REFERENCES MOVIE (ID_M) ON DELETE CASCADE
        ON UPDATE CASCADE
  );
```

в которую будем автоматически записывать любые изменения, добавления, удаления в таблице MOVIE. При этом будем фиксировать время (Time), пользователя (USER), измененный столбец (COLUMN), старое и новое значение столбца (OLD_VALUE и NEW_VALUE),

операцию (INS, UPD, DEL) над таблицей, а также идентификатор той записи, над которой происходит изменение (ID_M).

```
CREATE TRIGGER MOVIE_AI0 FOR MOVIE
ACTIVE AFTER INSERT POSITION 0
AS
BEGIN
  INSERT INTO "Log"
  VALUES (current_timestamp, current_user, 'MOVIE', null, null, null,
      'INS', NEW."ID_M");
END
CREATE TRIGGER MOVIE_AU0 FOR MOVIE
ACTIVE AFTER UPDATE POSITION 0
AS
begin
 if (OLD."ID_M"<>NEW."ID_M") then
   INSERT INTO "Log"
   VALUES (current_timestamp, current_user, 'ID_M',
       OLD."ID_M", NEW."ID_M", 'UPD', NEW."ID_M");

 if (OLD."TITLE"<>NEW."TITLE") then
   INSERT INTO "Log"
   VALUES (current_timestamp, current_user, 'TITLE',
       OLD."TITLE", NEW."TITLE", 'UPD', NEW."ID_M");

 if (OLD."ID_PR"<>NEW."ID_PR") then
   INSERT INTO "Log"
   VALUES (current_timestamp, current_user, 'ID_PR',
       OLD."ID_PR", NEW."ID_PR", 'UPD', NEW."ID_M");

 if (OLD."YEAR"<>NEW."YEAR") then
   INSERT INTO "Log"
   VALUES (current_timestamp, current_user, 'YEAR',
       OLD."YEAR", NEW."YEAR", 'UPD', NEW."ID_M");
```

```
  if (OLD."LEN"<>NEW."LEN")then
    INSERT INTO "Log"
    VALUES (current_timestamp, current_user, 'LEN',
          OLD."LEN", NEW."LEN", 'UPD', NEW."ID_M");
  else
    if ((OLD."LEN" IS NULL) OR (NEW."LEN" IS NULL)) then
      INSERT INTO "Log"
      VALUES (current_timestamp, current_user, 'LEN',
            OLD."LEN", NEW."LEN", 'UPD', NEW."ID_M");

  if (OLD."KIND"<>NEW."KIND") then
    INSERT INTO "Log"
    VALUES (current_timestamp, current_user, 'KIND',
          OLD."KIND", NEW."KIND", 'UPD', NEW."ID_M");
END

CREATE TRIGGER PROD DEL_LOG FOR PROD
ACTIVE
AFTER UPDATE
AS
BEGIN
  INSERT INTO PROD_LOG(DAT_IZM, ACT, OLD_TQVAR,
NEW_FIO)
  VALUES ("NOW","DEL",OLD.FIO,"") ;
END
```

Пусть в таблицу MOVIE внесены некоторые изменения. Тогда, выполнив оператор

```
SELECT * FROM MOVIE_LOG;
```

получим историю изменений таблицы MOVIE:

DAT_IZM	DEISTV	OLD_FIO	NEW_FIO
30-JUN-1997	ADD		И. Рязанов
30-JUN-1997	UPD	И. Рязанов	Э. Рязанов
30-JUN-1997	DEL	Э. Рязанов	

ФУНКЦИИ, ОПРЕДЕЛЯЕМЫЕ ПОЛЬЗОВАТЕЛЕМ

В InterBase есть возможность использовать функции, определяемые пользователем (User Defined Functions).

Функция должна находится в DLL и регистрироваться в БД:

```
DECLARE EXTERNAL FUNCTION Имя_функции
[список входных параметров]
RETURNS {тип_данных [BY VALUE] | CSTRING (Длина)}
ENTRY_POINT "Имя_функции_в_DLL"
MODULE_NAME "Имя_файла_DLL"
```

Список входных параметров необязателен, он задается указанием типа данных для каждого параметра, через запятую. После RETURNS указывается тип передаваемого результата и способ его передачи: по значению (BY VALUE) или по ссылке (тип CSTRING соответствует типу PCHAR).

Текст библиотеки:

```
library Project1;
uses
  SysUtils, Classes;
{$R *.RES}
function Upper_Rus (InStr: PChar): PChar; cdecl; export;
begin
  Result:=PChar(ANSIUpperCase(String(InStr)))
end;
exports
   Upper_Rus;
begin
end.
```

После компиляции Project1.dll размещается в каталоге UDF (для версий Interbase 6.x и клонов) или там же где размещается сам сервер C:\Program Files\InterBase Corp\Interbase\bin\)

Затем пишется запрос:

```
DECLARE EXTERNAL FUNCTION UPPER_RUS
CSTRING (256)
RETURNS CSTRING (256)
ENTRY_POINT 'UPPER_RUS'
```

MODULE_NAME 'PROJECT1'

Теперь можно обращаться к функции:

SELECT UPPER_RUS(Name) FROM Tovar;Обработка исключений и ошибок

Эти расширенные возможности позволяют перенести часть бизнес-логики с уровня приложения на уровень БД.

Исключения схожи с исключениями в языках программирования высокого уровня, но со своими особенностями. Исключение Interbase — это сообщение об ошибке, которое имеет собственное, задаваемое программистом имя и текст сообщения об ошибке.

Создается оператором:

CREATE EXCEPTION <имя исключения> <текст исключения>

Например, CREATE EXCEPTION My_Except 'Ошибка'

Удаляются исключения: DROP EXCEPTION <имя исключения>

Изменить: ALTER EXCEPTION <имя исключения> <текст исключения>

Использовать исключение в хранимой процедуре или триггере:

EXCEPTION <имя исключения>

Например, ХП получает два параметра: делимое и делитель и возвращает частное. В случае деления на 0 должно возникнуть предварительно созданное исключение.

```
CREATE EXCEPTION "Div_Zero" 'На ноль делить нельзя!';
CREATE PROCEDURE DELENIE (
   A NUMERIC(15,2),
   B NUMERIC(15,2))
RETURNS (
   C NUMERIC(15,2))
AS
begin
  if (B=0 OR B IS NULL) then
     begin
       EXCEPTION "Div_Zero";
       C=0;
     end
  else C=A/B;
end
```

В результате вызова этой процедуры из приложения с параметром B=0 получим сообщение, изображенное на рисунке 1.

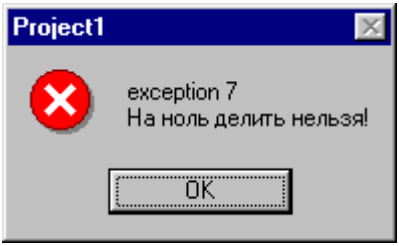

Рис. 1. Сообщение об ошибке

Это результат обработки нашего исключения сервером Interbase.

Когда возникает исключение, сервер прерывает работу ХП или триггера и откатывает все изменения, сделанные в текущем блоке BEGIN...END, причем если процедура является процедурой выборкой, то отменяются действия лишь до последнего оператора SUSPEND, исключая сам SUSPEND.

Например, приведенная ниже тривиальная процедура вернет следующий НД (плюс сообщение исключения):

```
CREATE PROCEDURE NEW_PROCEDURE
RETURNS (NAME VARCHAR(20))
AS
begin
  FOR SELECT "NAME"
     FROM Tovar
     ORDER BY 1
     INTO :NAME
  DO
    BEGIN
      IF (C LIKE 'Од%') then
           EXCEPTION "My_Except";
      SUSPEND;
    END
end
```

C
Товары
Продовольственные
Промышленные
Молочные
Мясные
Молоко
Кефир
Сметана
Колбаса
Мясо
Обувь

Одежда

Для обработки исключений существует конструкция алгоритмического языка:

```
WHEN EXCEPTION <Имя исключения> DO
BEGIN
  /*обработка исключения */
END
```

Использование этой конструкции позволяет избежать сообщения об ошибке и реализовать собственные действия по обработке исключения.

Алгоритм действия следующий: когда происходит исключение выполнение ХП или триггера прерывается и Interbase ищет конструкцию WHEN EXCEPTION ... DO в текущем блоке BEGIN...END. Если не находит — поднимается на уровень выше (в смысле вложенных блоков BEGIN...END или вызова ХП одной из другой) и ищет обработчик там. И т.д. пока не закончится вложенность уровней ХП или не будет найден обработчик. В первом случае будет выдано стандартное сообщение об ошибке, содержащее текст исключения. Если обработчик найден, то выполняются действия в его блоке после DO, а затем управление передается на оператор, следующий за обработчиком.

NAME
0%
20%
25%
5%
5%
Бейсболка
Ветчина
Головные уборы
Докторская
Кефир
Колбаса
Костюм
Любительская
Молоко

Предыдущий пример с обработчиком:

```
CREATE PROCEDURE NEW_PROCEDURE
RETURNS (NAME VARCHAR(20))
AS
begin
  FOR SELECT "NAME"
      FROM Tovar
      ORDER BY 1
      INTO :NAME
  DO
    BEGIN
      if (NAME LIKE 'Од%') then
         EXCEPTION "My_Except";
      SUSPEND;
      WHEN EXCEPTION "My_Except" DO
        begin
          NAME='аджедО';
          SUSPEND;
        end
    END
end
```

Молочные
Мясные
Мясо
Обувь
аджедО
Пальто
Продовольственные
Промышленные
Сметана
Товары

Если опустить выделенный оператор SUSPEND, то значение 'аджедО' не будет передано в НД-результат.

Что за исключение "My_Except" — не принципиально.

Обработчик можно использовать также и для обработки ошибок Interbase по тому же принципу: сервер ищет обработчик последовательно по всем уровням вложенности, начиная с того, на котором возникла ошибка. Конструкция обработчика:

```
WHEN GDSCODE|SQLCODE <код_ошибки> DO
BEGIN
  /*обработка ошибки */
END
```

В зависимости от GDSCODE|SQLCODE обрабатываются ошибки либо Interbase, либо SQL. Например, попытка указать значение

ID_PARENT отличное от NULL и не являющееся одним из значений ID_T (первичного ключа),

```
CREATE PROCEDURE WHEN_SQLCODE_DO
AS
Begin
  INSERT INTO TOVAR VALUES (NULL, 100, 'Проба', 0);
end
```

приводит к ошибке:

```
violation of FOREIGN KEY constraint "INTEG_25" on table "TOVAR"
```

Номер этой ошибки (см. "www.ibase.ru/v6/doc/Langref.pdf") — -530. Поэтому добавляем в процедуру обработчик SQL-ошибки -530.

```
CREATE PROCEDURE WHEN_SQLCODE_DO
AS
begin
  INSERT INTO TOVAR VALUES (NULL, 100, 'Проба', 0);
  WHEN SQLCODE -530 DO
    INSERT INTO TOVAR VALUES (NULL, NULL, 'Неудачная вставка', 0);
end
```

Таким образом, внутри ХП можно "перехватить" практически любую ошибку и нужным образом на нее отреагировать. Можно перечислить несколько обработчиков, чтобы определить реакцию на различные ошибки, а можно описать один обработчик на все возможные ошибки SQL и Interbase и исключения. Для этого существует конструкция:

```
WHEN ANY DO
BEGIN
  /*действия при любой ошибке или исключении */
END
```

С помощью использования описанных механизмов можно сделать приложения БД значительно более отказоустойчивыми и дружественными.

ГЕНЕРАТОРЫ

Часто в состав первичного или уникального ключа входят цифровые поля, значения которых должны быть уникальны, то есть не повторяться ни в какой другой записи таблицы. В одних случаях такое значение является семантически значимым и формируется пользователем по определенному алгоритму — например, номер лицевого счета в банке. В других случаях лучше предоставить выработку такого значения приложению или серверу БД.

В InterBase отсутствует аппарат автоинкрементных столбцов. Вместо этого для установки уникальных значений столбцов можно использовать аппарат генераторов.

Генератором называется хранимый на сервере БД механизм, возвращающий уникальные значения, никогда не совпадающие со значениями, выданными тем же самым генератором в прошлом.

Для создания генератора используется оператор

CREATE GENERATOR <ИмяГенератора>;

Для генератора необходимо установить стартовое значение при помощи оператора

SET GENERATOR <ИмяГенератора> TO <СтартовоеЗначение>;

При этом *СтартовоеЗначение* должно быть целым числом.

Для получения уникального значения к генератору нужно обратиться с помощью функции

GEN_ID (<ИмяГенератора>, <шаг>) ;

Эта функция возвращает увеличенное на *шаг* предыдущее значение, выданное генератором (или увеличенное на шаг стартовое значение, если ранее обращений к генератору не было). Значение шага должно принадлежать диапазону –231...+231 -1.

Замечание. Не рекомендуется переустанавливать стартовое значение генератора или менять шаг при разных обращениях к GEN_ID. В противном случае генератор может выдать неуникальное значение, и как следствие будет возбуждено исключение при попытке запоминания новой записи в ТБД.

Пусть в БД определен генератор, возвращающий уникальное значение для столбца ID_T в таблице TOVAR:

CREATE GENERATOR GEN_TOVAR_ID;

SET GENERATOR GEN_TOVAR_ID TO 23;

Обращение к генератору непосредственно из оператора *INSERT:*

INSERT INTO PROD VALUES (GEN_ID(GEN_TOVAR_ID, 1), 14, "Ушанка", 20);

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

В базе данных, созданной в предыдущей лабораторной работе, разработать триггеры для автоинкремента, бизнес-правил, каскадных воздействий и журнала изменений.

КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ

1. Дайте определение термину «триггер».
2. Перечислите виды триггеров по событиям.
3. Перечислите виды триггеров по отношению к событию.
4. Приведите синтаксис команды создания триггера.
5. Приведите синтаксис команды создания пользовательской функции.
6. Приведите синтаксис команды создания исключения.
7. Приведите синтаксис команды перехвата исключений.
8. Дайте определение термину «генератор».
9. Приведите синтаксис команды создания генератора.

ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

На выполнение лабораторной работы отводится 2 занятия (4 академических часа: 3 часа на выполнение и сдачу лабораторной работы и 1 час на подготовку отчета).

Номер варианта студенту выдается преподавателем.

Отчет на защиту предоставляется в печатном виде.

Структура отчета (на отдельном листе(-ах)): титульный лист, формулировка задания (вариант), ход выполнения работы, результаты выполнения работы, выводы.

ОСНОВНАЯ ЛИТЕРАТУРА

1. Карпова, Т.С. Базы данных: модели, разработка, реализация : учебное пособие / Т.С. Карпова. - 2-е изд., исправ. - Москва : Национальный Открытый Университет «ИНТУИТ», 2016. - 241 с. : ил. ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429003

2. Давыдова, Е.М. Базы данных [Электронный ресурс] : учеб. пособие / Е.М. Давыдова, Н.А. Новгородова. — Электрон. дан. — Москва : ТУСУР, 2007. — 166 с. — Режим доступа: https://e.lanbook.com/book/11636. — Загл. с экрана.

3. Харрингтон, Д. Проектирование объектно ориентированных баз данных [Электронный ресурс] — Электрон. дан. — Москва : ДМК Пресс, 2007. — 272 с. — Режим доступа: https://e.lanbook.com/book/1231. — Загл. с экрана.

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

4. Голицина О.Л., Максимов Н.В., Попов И.И. Базы данных: учеб. пособие. – М.: Форум:Инфра-М, 2007.

5. Гагарин Ю.Е. Применение языка SQL в MS Access: учебно-методическое пособие. – М.: МГТУ им. Н.Э. Баумана, 2012.

Электронные ресурсы:

1. Научная электронная библиотека http://eLIBRARY.RU
2. Электронно-библиотечная система http://e.lanbook.com
3. Электронно-библиотечная система «Университетская библиотека онлайн» http://biblioclub.ru
4. Электронно-библиотечная система IPRBook http://www.iprbookshop.ru