



Министерство образования и науки Украины
Севастопольский национальный технический университет

**ИЗУЧЕНИЕ ОСНОВ ЯЗЫКА МАНИПУЛИРОВАНИЯ
ДАНЫМИ SQL НА БАЗЕ СЕРВЕРА INTERBASE**

Методические указания
к лабораторным работам №7 - №9
по дисциплине
“ОРГАНИЗАЦИЯ БАЗ ДАННЫХ”
для студентов специальности 7.080401 –
"Информационные управляющие системы и технологии"
всех форм обучения

**Севастополь
2005**



УДК 004.92

Изучение основ языка манипулирования данными SQL на базе сервера Interbase. Методические указания к лабораторным работам №7- №9 по дисциплине “Организация баз данных”, для студентов специальности 7.080401 -"Информационные управляющие системы и технологии" всех форм обучения /Сост. Ю.В. Доронина, О.Л. Тимофеева, В.Е. Шишкевич. - Севастополь: Изд-во СевНТУ, 2005.-28 с.

Цель методических указаний: выработка у учащихся практических навыков по работе с реляционными базами данных.

Методические указания рассмотрены и утверждены на заседании кафедры Информационных Систем. Протокол № 7 от 19 мая 2005 г.

Рецензент: доц. департамента кибернетики и вычислительной техники, к.т.н. Литвинова Л.А.

Допущено учебно-методическим центром в качестве методических указаний.

СОДЕРЖАНИЕ

Лабораторная работа № 7. Пользовательские представления	4
Лабораторная работа № 8. Генераторы	10
Лабораторная работа № 9. Язык SQL. Триггеры	14

ЛАБОРАТОРНАЯ РАБОТА № 7. ПОЛЬЗОВАТЕЛЬСКИЕ ПРЕДСТАВЛЕНИЯ

1. Цель работы:

Ознакомится с принципом работы пользовательских представлений, продемонстрировать работу на примере.

2. Основные положения

2.1. Пользовательские представления

Информация в реляционной базе данных разбита по множеству таблиц таким образом, чтобы обеспечить безизбыточное хранение информации. Подобный принцип хорош для правильного функционирования базы данных, но работать пользователю с таблицами БД напрямую неудобно.

Например, расписание движения поездов на железной дороге может быть представлено таблицей, в соответствии с рисунком 1.

Номер поезда	Станция Отправления	Станция Прибытия	Время отправления	Время прибытия
012	Москва	Петушки	12:30	14:20

Рисунок 1 – Таблица «Расписание»

В реляционной БД можно предложить следующий набор таблиц (представленных на рисунках 2-4) для хранения информации о расписании:

Номер станции	Наименование станции
1	Москва
2	Петушки

Рисунок 2 – Таблица «Станция»

Номер поезда	Номер станции отправления	Номер станции прибытия
012	1	2

Рисунок 3 – Таблица «Поезд»

Номер поезда	Время отправления	Время прибытия
012	12:30	14:20

Рисунок 4 – Таблица «Расписание»

Очевидно, что если просто вывести содержимое таблицы "Расписание", оно мало, чем поможет пассажиру, который не помнит маршруты поездов по номерам. Для того чтобы выводить данные из БД в удобном для пользователя виде, существуют пользовательские представления (VIEW).

Пользовательское представление - это объект базы данных, представляющий из себя запрос на выборку, каким - либо образом комбинирующий информацию из базовых таблиц базы данных. Часто о представлениях говорят как о "виртуальных" таблицах. Физически данные из исходных таблиц в представления не переносятся, но со стороны представления выглядят как полноправные таблицы.

Так как представления - это объекты БД, то они сохраняются в файле БД. Представления можно определять на одной таблице, на наборе таблиц, в том числе можно использовать и другие представления. В базе данных хранится только определение пользовательского представления. Когда представление запрашивается клиентской программой, выполняется запрос на выборку из представления. Если данные из базовых таблиц представления изменены другим пользователем, то представление автоматически перестраивается сервером БД для того, чтобы отобразить изменения.

Представление может быть составлено из:

- подмножества столбцов одной таблицы (выводятся не все столбцы таблицы)
- подмножества строк одной таблицы (выводятся не все строки таблицы)
- подмножества строк и столбцов одной таблицы
- подмножества строк и столбцов нескольких таблиц (объединения)

2.1.1. Преимущества представлений

- Упрощенный доступ к данным.
- Персонализированный доступ к данным. На одном и том же наборе таблиц можно сделать несколько представлений. Например, одно будет показывать информацию необходимую клерку, а другое - менеджеру.
- Независимость от изменений базовых таблиц БД. Можно поменять структуру БД, но переписать представление таким образом, чтобы оно отображало те же данные.
- Защита данных. В представлении можно выводить только те данные, которые должны быть видимы пользователю. Например, можно сделать список сотрудников сделать доступным для открытого просмотра, но не выводить поле "Зарплата" в нем.

2.1.2. Создание представлений

Представления создаются оператором CREATE VIEW. Синтаксис оператора:

```
CREATE VIEW name [( view_col [, view_col :])]
AS < select > [WITH CHECK OPTION];
```

Здесь

select - запрос на выборку. Представление нельзя создавать как select из хранимых процедур.

view_col - имена столбцов, выводимых представлением.

Если имена столбцов не указать, то будут использованы имена столбцов из запроса. Если имена столбцов указаны, они должны следовать в том же порядке и в таком же количестве, как возвращаются запросом на выборку. Имя столбца должно быть уникальным среди имен столбцов данного представления. Если в **базовом запросе присутствуют выражения**, имена столбцов должны быть **указаны обязательно**.

WITH CHECK OPTION - если указана данная опция, то запрещаются операции INSERT и UPDATE на обновляемых представлениях, если они не удовлетворяют условию WHERE в формулировке представления. То есть позволяют только те операции, которые будут "видимы" представлением.

2.1.3. Обновляемые и не обновляемые представления

Представление является обновляемым, и может распространять изменения, вносимые в представление на базовые таблицы, только при выполнении некоторых условий. Если условия не выполняются, то представление может служить только для чтения. Термины "обновляемое" и "только для чтения" относятся к содержимому представления, а не к его объявлению.

Условия обновляемости представлений:

- Представление является подмножеством одной таблицы, либо другого обновляемого представления.
- Все столбцы базовых таблиц не включенные в представление должны позволять состояние NULL.
- Оператор SELECT не должен содержать подзапросов, слова DISTINCT, раздела HAVING, агрегатных функций, объединения таблиц, пользовательских функций, хранимых процедур.

2.1.4. Примеры представлений

Описание базовых отношений приведено в Лабораторной работе №1 п.2.1. Пример редактируемого представления:

```
CREATE VIEW EMP_MNGRS (FIRST, LAST, SALARY) AS  
SELECT FIRST_NAME, LAST_NAME, SALARY  
FROM EMPLOYEE  
WHERE JOB_CODE = 'Mngr';
```

В следующем представлении используется подзапрос, поэтому представление не редактируемое:

```
CREATE VIEW ALL_MNGRS AS  
SELECT FIRST_NAME, LAST_NAME, JOB_COUNTRY  
FROM EMPLOYEE  
WHERE JOB_COUNTRY IN  
(SELECT JOB_COUNTRY FROM JOB  
WHERE JOB_TITLE = 'manager');
```

Не редактируемое представление, использующее соединение таблиц:

```
CREATE VIEW PHONE_LIST AS
  SELECT EMP_NO, FIRST_NAME, LAST_NAME,
         PHONE_EXT, LOCATION, PHONE_NO
  FROM EMPLOYEE, DEPARTMENT
  WHERE EMPLOYEE.DEPT_NO=DEPARTMENT.DEPT_NO
```

2.1.5. Удаление представлений

Удалить представление можно с помощью команды DROP VIEW:

DROP VIEW name;

Здесь name - имя удаляемого представления. Не существует способов изменить существующее представление. Можно удалить старое представление и заново создать новое.

3. Порядок выполнения лабораторной работы

1. В соответствии с вариантом задания создать представления.
2. Просмотреть хранимые данные, используя представления.
3. Используя редактируемое представление внести изменения в таблицу. Просмотреть полученный результат.
4. Удалить полученные представления.
5. Написать отчет.

4. Содержание отчета

1. Отчет состоит из титульного листа, цели работы, описания процесса выполнения работы и вывода.
2. Отчет должен содержать:
 - исходные данные,
 - описание представлений,
 - тексты запросов и результаты их выполнения
 - объяснение полученных результатов

5. Задание на работу

Таблица 1 - Задание на работу

Вариант	Задание
1	Создать представления "СписокСтатей", "СписокЖурналов" (объединяя таблицу "Публикация" с таблицами "Сведения о статье", "Сведения о журнале"), "Публикация" (Фамилия и инициалы автора, Наименование темы, Заглавие, Дата опубликования, Тип).

Продолжение таблицы 1

2	Создать представления Слушатели_курсов (Название курса, ФИО преподавателя, Количество студентов на курсе), Расписание (Название курса, Тип занятия, Дата, Номер аудитории)
3	Создать представления: Операция, подставив вместо номеров текстовые значения, Загрузка_хирурга(ФИО хирурга, месяц, количество операций)
4	Представление объединяет таблицу "Личность" с таблицей "Ребенок" или "Родитель"
5	Создать представления: "ПомещениеТип" (в таблицу "Помещение" вместо номера типа помещения подставить наименование типа помещения), КлиентыФирмы(Наименование фирмы, Количество продавцов, Количество покупателей)
6	Создать представления: Рабочие (ФИО рабочего, Название работы, Выслуга лет, Минимальная ставка, Максимальная ставка)
7	Создать представления: "Список отпусков", объединив таблицы "Трудовая деятельность" и "Отпуск"; Премии(ФИО сотрудника, Сумма премий, Сумма дней отпусков)
8	Создать представления: Курсы (ФИО преподавателя, Наименование курса, Количество студентов), Факультеты(Наименование факультета, Количество курсов, Количество студентов, Количество преподавателей)
9	Создать представления: Экзамены (Код специальности, Наименование Вуза, ФИО абитуриента, Наименование предмета, Оценка), Успеваемость (Наименование Вуза, Код специальности, Процент пятерок, Процент четверок, Процент троек).
10	Создать представления: Расписание (Дата рейса, Номер поезда, Наименование станции отправления, Наименование станции прибытия, Количество проданных билетов); Пассажиры (Номер паспорта, Количество рейсов).
11	Создать представления: Экзамены (Дата, Наименование факультета, Наименование предмета, ФИО экзаменатора, Группа); Отметки (ФИО студента, Название предмета, Оценка).
12	Создать представления: Псевдонимы (Номер персоны, ФИО, Псевдоним), Дела (Номер дела, Описание, Осуждено персон по делу, Суммарный срок, Средний срок).
13	Создать представления: Результаты (Название соревнований, ФИО спортсмена, Количество забитых мячей), Лучшие (Название команды, Название соревнования, ФИО спортсмена, забившего больше всего голов).
14	Создать представления: Программа_концерта (Дата концерта, Время концерта, Номер зала, ФИО артиста, Номер по порядку в концерте, Содержание выступления); Артисты (ФИО артиста, количество выступлений в год).

Продолжение таблицы 1

15	Создать представления: Родственники (Табельный номер сотрудника, ФИО сотрудника, ФИО ребенка, Год рождения ребенка); Дочерние_фирмы (Название головной фирмы, Название дочерней фирмы).
16	Создать представления: Перевозки (Дата, Номер путевки, ФИО водителя, Место отправления, Место назначения, Количество грузов);
17	Создать представления: Номера_гостиницы (Название гостиницы, Адрес, Разряд, Порядковый номер номера, Количество комнат, Разряд номера); Клиенты (ФИО клиента, Количество заказов, Суммарный срок проживания).
18	Создать представления: Сложность (Наименование диагноза, Количество единиц оборудования, Количество препаратов, Количество нетрадиционных методов); КурсПрепарат (Номер курса, Побочный эффект).
19	Создать представления: Машины_инженера, состоящее из всех не ключевых полей таблиц "Инженер" и "Машина"; Комплектующие (Номер машины, Стоимость комплектующих)
20	Создать представления: Партии (Название партии, количество партийцев, Суммарная стоимость имущества); Газеты (Наименование, Тираж, Главный редактор, Количество сотрудников, Стоимость).

6. Контрольные вопросы

1. Что такое "пользовательское представление"?
2. Условия редактируемости представлений?
3. Назначение пользовательских представлений?
4. Привести примеры пользовательских представлений?
5. Отличие представлений от таблиц?
6. Организация ограничения доступа пользователей к набору данных?
7. Условие обеспечения целостности данных?
8. Внешняя модель и создание представлений при разработке БД?

ЛАБОРАТОРНАЯ РАБОТА № 8. ГЕНЕРАТОРЫ

1. Цель работы:

Ознакомится с принципом работы генераторов, продемонстрировать работу на примере

2. Основные положения

2.1. Генераторы

Генератор – объект базы данных, служащий для генерации последовательностей целых чисел. Во многих таблицах используются искусственные первичные ключи. Например, в таблице «Должность» два поля – «Номер должности» (первичный ключ) и «Наименование должности». В данном случае первичный ключ – искусственный, он не является характеристикой сущности «Должность», а был введен специально (искусственно) для того, чтобы не сносить текстовое поле в дочерние таблицы. В случае применения искусственного ключа по натуральному ключу «Наименование должности» должен быть создан альтернативный ключ.

Каждый раз, занося новую строку в таблицу «Должность», в качестве номера первичного ключа надо использовать следующее целое число. Это можно сделать двумя способами:

- 1 Сначала запросом получить максимальный номер должности в таблице, затем прибавить к нему единицу и затем занести новую строку.
- 2 Воспользоваться генератором, который подставит следующее значение автоматически.

Первый подход хоть и неудобен, но работает в однопользовательских базах. В многопользовательских базах нет гарантии, что два клиента одновременно не сгенерируют одинаковый номер. Этого недостатка лишен генератор – каждый клиент получит разные номера.

2.1.1. Создание генератора

Для создания генератора служит команда **CREATE GENERATOR**. После создания, генератор имеет значение 0. Синтаксис оператора:

CREATE GENERATOR name;

Здесь name - имя генератора. Имя генератора, как правило, содержит имя поля, для которого он предназначен. Например, генератор для поля TaskID может называться TaskID_GEN.

Для того чтобы установить генератор в другое значение, необходимо использовать команду **SET GENERATOR**:

SET GENERATOR name **TO** value;

Здесь

name - имя генератора,

value - новое значение генератора (число в диапазоне от -2^{64} до $2^{64} - 1$).

После вызова оператора SET GENERATOR следующее значение, которое вернет функция GEN_ID() (см. ниже) будет равно value + инкремент, указанный в функции. Перед вызовом функции необходимо удостовериться, что после того как генератор будет переустановлен, он будет возвращать уникальные значения (если он используется для генерации первичного ключа).

2.1.2. Использование генераторов

Генератор представляет собой переменную для хранения некоторого текущего значения. После создания генератора место под переменную отведено, генератору присвоено начальное значение. Для того чтобы получить следующее значение генератора, необходимо применить функцию GEN_ID:

GEN_ID(name, step);

Здесь name - имя генератора, step - инкремент генератора.

Функция GEN_ID может быть использована для ввода значений в таблицу, в триггере или хранимой процедуре. Пример использования функции:

```
INSERT INTO Subject (SubjectID, SubjectName)
VALUES (GEN_ID(SubjectID_GEN,1),'OBD');
```

Генератор возвращает 64-битовое значение. Столбец, в котором сохраняется значение генератора, должен быть соответствующего типа (DECIMAL или NUMERIC). В процедуре переменная для сохранения значения генератора должна быть типа ISC_INT64.

2.1.3. Удаление генератора

Не существует оператора DROP GENERATOR. Генератор можно удалить, удалив его из системной таблицы:

```
DELETE FROM RDB$GENERATORS
WHERE RDB$GENERATORS_NAME = 'SubjectID_GEN';
```

3. Порядок выполнения лабораторной работы

- 1 В соответствии с вариантом задания создать генераторы.
- 2 Изменить значение генератора, в соответствии с хранимыми данными.
- 3 Ввести данные в таблицу, используя генератор (не менее 5 строк).
Просмотреть полученный результат.

- 4 Удалить созданные генераторы.
- 5 Написать отчет.

4. Содержание отчета

Отчет состоит из титульного листа, цели работы, описания процесса выполнения работы и вывода.

Отчет должен содержать исходные данные, этапы работы по созданию и изменению генератора, ввод данных в таблицу, с использованием генератора, выборку информации из таблицы на каждом шаге работы.

5. Задание на работу

Таблица 1 – Задание на работу

Вариант	Задание
1	Создать генераторы для полей "Номер автора", "Номер темы", "Идентификатор".
2	Создать генераторы для полей "Номер преподавателя", "Номер курса", "Номер студента".
3	Создать генераторы для полей "Номер пациента", "Номер патента хирурга", "Номер операции".
4	Создать генератор для поля "Номер". Создать представления: "СписокДетей", "СписокРодителей" (представление объединяет таблицу "Личность" с таблицей "Ребенок" или "Родитель")
5	Создать генераторы для полей "Номер фирмы", «Номер типа помещения».
6	Создать генераторы для полей "Код курса", "Номер служащего", "Код профессии", "Номер ставки", "Код работы".
7	Создать генераторы для полей "Номер служащего", "Номер события", "Номер языка", "Тип образования".
8	Создать генераторы для полей "Номер специальности", "Номер факультета".
9	Создать генераторы для полей "Номер предмета", "Код специальности".
10	Создать генераторы для полей "Номер поезда", "Номер станции".
11	Создать генераторы для полей "Номер факультета", "Номер предмета".
12	Создать генераторы для полей "Номер персоны", "Номер дела", "Номер псевдонима".
13	Создать генераторы для полей "Номер соревнования", «Количество забитых мячей».
14	Создать генераторы для полей "Номер зала", "Номер концерта", "Номер фирмы", "Номер артиста".
15	Создать генераторы для полей "Код фирмы", "Номер номенклатуры".

Продолжение таблицы 1

16	Создать генераторы для полей "Номер фирмы", "Номер водителя", "Номер груза".
17	Создать генераторы для полей "Номер гостиницы", "Номер фирмы".
18	Создать генераторы для полей "Номер диагноза", "Номер курса", "Номер прибора", "Номер препарата", "Номер метода", "Номер ограничения".
19	Создать генераторы для полей "Номер инженера", "Номер ВЦ", "Номер пакета", "Номер комплектующего".
20	Создать генераторы для полей "Номер имущества", "Номер помещения", "Номер газеты".

6. Контрольные вопросы

1. Назначение генераторов?
2. Как сгенерировать следующее значение генератора?
3. Как переустановить значение генератора?
4. Как удалить генератор?
5. Повышение надежности данных?
6. Организация многопользовательского режима доступа к данным?

ЛАБОРАТОРНАЯ РАБОТА № 9. ЯЗЫК SQL. ТРИГГЕРЫ

1. Цель работы:

Ознакомится с принципом работы триггеров.

2. Основные положения

2.1. Создание триггеров

Триггер - процедура, автоматически вызываемая при операциях с таблицей. Под операциями понимаются операторы INSERT, UPDATE, DELETE. Каждый триггер может быть вызван до соответствующей операции или после нее. Преимущества использования триггеров:

- Автоматическое отслеживание целостности данных не только на уровне связи между таблицами, но и любым произвольным образом.
- Облегчение написания приложений БД и их поддержки.

2.1.1. Синтаксис оператора CREATE TRIGGER

Оператор состоит из заголовка триггера и тела триггера. Заголовок содержит:
имя триггера, уникальное по БД;
имя таблицы, с которой ассоциируется триггер;
указание на момент, когда триггер должен вызываться.

Тело триггера состоит из опционального списка локальных переменных и операторов.

Синтаксис оператора:

```
CREATE TRIGGER name FOR { table | view }
    [ACTIVE | INACTIVE]
    {BEFORE | AFTER} {DELETE | INSERT | UPDATE}
    [POSITION number]
AS < trigger_body>
```

```
< trigger_body> = [<variable_declaration_list>] < block>
```

```
< variable_declaration_list> =DECLARE VARIABLE variable datatype;
[DECLARE VARIABLE variable datatype; .]
< block> =
```

```
BEGIN
```

```
< compound_statement> [<compound_statement> .]
```

```
END
```

```
< compound_statement> = {<block> | statement;}
```

Используемые обозначения в синтаксисе оператора приведены в Таблице 1.

Таблица 1 – Используемые обозначения

name	Имя триггера. В имени обычно упоминают имя таблицы и момент запуска триггера. Например, WORKER_BI_T1 - триггер #1 таблицы WORKER, вызываемый перед добавлением (before insert).
table	Имя таблицы или пользовательского представления, с которым ассоциируется триггер.
ACTIVE INACTIVE	Указывает "активность" триггера. Не активные триггеры игнорируются. По умолчанию триггер активен.
BEFORE AFTER	Указывает момент запуска триггера до операции (BEFORE) или после операции (AFTER).
DELETE INSERT UPDATE	Указывает операцию, с которой связан триггер.
POSITION number	Указывает порядок срабатывания триггера. На одно и то же действие могут быть назначены несколько триггеров. В таком случае они должны различаться позициями. Триггеры вызываются в порядке увеличения позиции. Позиции триггеров не обязательно должны идти строго друг за другом (1, 2, 3) допустимы произвольные позиции (5, 25, 37). Триггеры с одинаковыми позициями вызываются по алфавиту, в соответствии с именами.
DECLARE VARIABLE var <datatype>	Описывает локальную переменную триггера. С помощью DECLARE VARIABLE можно описать только одну переменную, за оператором должна следовать точка с запятой. Var - имя локальной переменной, должно быть уникально для триггера. <datatype> - тип данных переменной
statement	Любой одиночный оператор на языке процедур и триггеров Interbase. Любой оператор за исключением BEGIN END должен заканчиваться точкой с запятой.
terminator	Разделитель, определенный оператором SET TERM. Разделитель служит для указания конца тела триггера. Используется только в isql.

2.1.2. Заголовок триггера

Все, что идет до части **AS** оператора **CREATE TRIGGER** составляет заголовок триггера. Заголовок триггера должен определять имя триггера и имя ассоциированной с триггером таблицы или пользовательского представления. Таблица или пользовательское представление должны существовать на момент

выполнения оператора **CREATE TRIGGER**. Оставшаяся часть оператора определяет, когда и как вызывается триггер:

- статус триггера, **ACTIVE** or **INACTIVE**. Если триггер активный, он вызывается при наступлении события триггера. Если триггер неактивный, он не вызывается.
- время вызова триггера: **BEFORE** (до) или **AFTER** (после) некоторого действия.
- операция, с которой связан триггер: **INSERT**, **UPDATE**, или **DELETE**. Может быть указана только одна операция. Если один и тот же триггер должен выполняться на несколько операций, то необходимо создать несколько триггеров с одинаковым телом, различающихся именем и операцией.
- (опционально) номер триггера по порядку среди всех триггеров ассоциированных с данной операцией на данной таблице - **POSITION**. Номер позиции может быть любым числом от 0 до 32767. Значение по умолчанию - 0. Триггеры с меньшими номерами вызываются раньше.

2.1.3. Тело триггера

Все, что следует за частью **AS** оператора **CREATE TRIGGER** составляет тело триггера. Тело триггера состоит из опционального списка локальных переменных, за которым идет блок операторов. Блок состоит из набора операторов на языке хранимых процедур и триггеров, заключенных в операторные скобки.

2.1.4. Язык хранимых процедур и триггеров Interbase

Язык хранимых процедур и триггеров Interbase (ЯХПТ) является полным языком для написания хранимых процедур и триггеров. Язык включает в себя:

- операторы манипулирования данными SQL (**INSERT**, **UPDATE**, **DELETE**) а также оператор **SELECT**;
- операторы и выражения SQL, включая функции, определяемые пользователем (UDF);
- расширения SQL, включая, оператор присваивания, операторы управления последовательностью выполнения, локальные переменные, операторы отсылки сообщений, обработка исключительных ситуаций, операторы обработки ошибок;

Несмотря на то, что триггеры и хранимые процедуры служат для разных целей и используются по-разному, они используют одинаковый язык. Существуют следующие ограничения:

- контекстные переменные используются только в триггерах;
- входные и выходные параметры, а также возвращающие результат операторы **SUSPEND** и **EXIT** могут быть использованы только в хранимых процедурах.

Элементы языка хранимых процедур и триггеров приведены в Таблице 2.

Таблица 2 - Элементы языка

BEGIN...END	определяет блок операторов (операторные скобки). Скобка после END не нужна.
variable = expression	оператор присваивания
/* comment_text */	многострочный комментарий
EXCEPTION exception_name	вызывает исключительную ситуацию с именем exception_name, если она не обрабатывается оператором WHEN
EXECUTE PROCEDURE proc_name [var [, var ...]] [RETURNING_VALUES var [, var ...]]	Вызывает хранимую процедуру с именем proc_name, с указанными входными и выходными параметрами. Параметры процедуры должны быть локальными переменными.
FOR select_statement DO compound_statement	повторяет выполнение блока кода следующего за DO для каждой строки, возвращенной оператором select_statement
select_statement.	select_statement - обычный запрос на выборку, за исключением того, что он обязательно должен содержать часть INTO, и данная часть должна идти на последнем месте.
compound_statement	или одиночный оператор языка, или блок операторов заключенных в операторные скобки.
IF (condition) THEN compound_statement [ELSE compound_statement]	условный оператор condition - условие, выражение булевой трехзвенной логики (TRUE, FALSE, UNKNOWN). Обычно два выражения как операнды оператора сравнения.
NEW.column	контекстная переменная, содержащая новое значение столбца с именем column при выполнении операций INSERT или UPDATE.
OLD.column	контекстная переменная, содержащая старое значение столбца с именем column при выполнении операций INSERT или UPDATE.
POST_EVENT event_name	отсылает сообщение с именем event_name.

Продолжение таблицы 2

WHILE (condition) DO compound_statement	цикл с предусловием.
WHEN {error [, error .] ANY} DO compound_statement	Оператор обработки исключительных ситуаций и ошибок. В случае если имеет место одна из ошибок перечисленных в операторе WHEN, вызывается compound_statement. Оператор WHEN должен быть последним оператором перед END тела триггера. Error: EXCEPTION exception_name, SQLCODE errcode or GDSCODE number. Ошибкой может быть - возбуждение исключительной ситуации с именем exception_name, равенство константы SQLCODE значению errcode, равенство константы GDSCODE значению number. ANY – обрабатывает любую ошибку, если она имела место.

2.2. Использование триггеров**2.2.1. Синтаксические ошибки в триггерах**

Сообщение об ошибке выглядит следующим образом (пример):

```
Statement failed, SQLCODE = -104
Dynamic SQL Error
-SQL error code = -104
-Token unknown - line 4, char 9
-tmp
```

Номер строки считается от начала оператора CREATE TRIGGER, а не от начала всех операторов. Символы считаются от левой границы. Неизвестный элемент (token) либо непосредственно является источником ошибки, либо источник ошибки находится справа от него. В случае сомнений необходимо исследовать всю строку на наличие ошибок.

2.2.2. Контекстные переменные NEW и OLD

Контекстная переменная OLD содержит текущие или старые значения элементов строки, которая обновляется или удаляется. OLD не используется для операций добавления (INSERT). Контекстная переменная NEW содержит новые значения столбцов строки, которые будут присвоены после операции добавления или обновления. NEW не имеет смысла для операции удаления. Контекстные переменные, как правило, используются для сравнения нового и старого значения столбца до его изменения и после.

Синтаксис контекстных переменных:

NEW.column

OLD.column

где column - имя столбца.

Контекстные переменные могут использоваться так же, как и простые локальные переменные.

Новое значение для столбца может быть получено только до операции. В триггере, который вызывается после операции INSERT, присвоение переменной NEW не имеет смысла. При проведении операции сначала вызываются триггеры BEFORE, они "видят" старое значение строки, затем производится действие, после чего вызываются триггеры AFTER, которые "видят" новое значение строки. Триггеры, описанные как BEFORE и обращающиеся к NEW не имеют смысла.

Например, рассмотрим триггер:

```

SET TERM !! ;
CREATE TRIGGER SAVE_SALARY_CHANGE FOR EMPLOYEE
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!!
SET TERM ; !!

```

Триггер вызывается после обновления таблицы EMPLOYEE, и сравнивает старую и новую сумму продаж. Если сумма изменилась, триггер производит запись в таблицу SALARY_HISTORY (история продаж).

2.2.3. Модификация триггеров

Для модификации триггеров используется оператор ALTER TRIGGER. С помощью ALTER TRIGGER можно менять тело триггера, заголовок триггера, а также триггер целиком.

Для того чтобы изменить триггер, автоматически создаваемый СУБД для проверки условия CHECK, следует использовать оператор ALTER TABLE.

Синтаксис ALTER TRIGGER:

```

ALTER TRIGGER name
[ACTIVE | INACTIVE]
[{BEFORE | AFTER} {DELETE | INSERT | UPDATE}]
[POSITION number]
AS <trigger_body>

```

Синтаксис **ALTER TRIGGER** похож на **CREATE TRIGGER** за исключением:

- **CREATE** заменено на **ALTER**
- Пропущена часть **FOR <имя таблицы>**, так как нельзя менять таблицу, с которой ассоциирован триггер.
- оператор должен содержать только те параметры, которые должны быть изменены, ниже приводятся исключения.

2.2.3.1. Модификация заголовка триггера

При модификации заголовка триггера **ALTER TRIGGER** требует, чтобы была изменена хотя бы одна позиция после имени триггера. Все характеристики триггера, пропущенные в **ALTER TRIGGER** сохраняют свои старые значения.

Следующий оператор делает триггер не активным.

ALTER TRIGGER SAVE_SALARY_CHANGE INACTIVE;

Если меняется момент вызова триггера (**BEFORE**, **AFTER**), то обязательно надо указать операцию (**UPDATE**, **INSERT**, **DELETE**).

Следующий оператор делает триггер активным, и меняет момент его запуска на **BEFORE UPDATE**:

**ALTER TRIGGER SAVE_SALARY_CHANGE
ACTIVE
BEFORE UPDATE;**

2.2.3.2. Модификация тела триггера

Заменить можно только тело триггера целиком. При этом новое описание полностью заменяет ранее существующее. При изменении тела триггера оператор может не содержать заголовочной информации за исключением имени триггера.

Последовательность действий для модификации тела триггера:

- Извлечь описание триггера;
- заменить **CREATE** на **ALTER**, удалить всю заголовочную информацию после имени триггера до ключевого слова **AS**;
- модифицировать тело триггера, запустить оператор.

2.2.3.3. Удаление триггера

Триггер можно удалить командой **DROP TRIGGER**. Синтаксис оператора:

DROP TRIGGER <имя триггера>

2.3. Триггеры и транзакции

Триггеры работают в контексте транзакции вызвавшего их приложения. В случае если транзакция приложения будет отменена, будут отменены и все действия триггера.

2.4. Триггеры и сообщения о событиях

Триггеры могут быть использованы для того, чтобы уведомить приложение о том, что произошло некоторое событие. Для примера рассмотрим триггер POST_NEW_ORDER, который генерирует событие с именем "NEW_ORDER" в случае если добавляется запись в таблицу SALES.

```
SET TERM !! ;
CREATE TRIGGER POST_NEW_ORDER FOR SALES
AFTER INSERT AS
BEGIN
    POST_EVENT 'NEW_ORDER';
END !!
SET TERM;
```

В общем случае триггер может использовать переменную в качестве имени события:

```
POST_EVENT :EVENT_NAME;
```

2.5. Использование триггеров для обновления пользовательских представлений

Пользовательские представления, основанные на соединении нескольких таблиц, как правило, не обновляемы, и могут служить только для чтения. Тем не менее, можно написать триггеры для операций добавления, обновления и удаления таким образом, что они будут правильно модифицировать базовые таблицы, из которых "собран" VIEW. Таким образом, можно редактировать не редактируемые пользовательские представления.

Следующие операторы создают две таблицы, создают пользовательское представление и три триггера для модификации представления:

```
CREATE TABLE Table1 (
    ColA INTEGER NOT NULL,
    ColB VARCHAR(20),
    CONSTRAINT pk_table PRIMARY KEY(ColA)
);
```

```
CREATE TABLE Table2 (  
    ColA INTEGER NOT NULL,  
    ColC VARCHAR(20),  
    CONSTRAINT fk_table2 FOREIGN KEY (ColA)  
        REFERENCES Table1(ColA)  
);
```

```
CREATE VIEW TableView AS  
    SELECT Table1.ColA, Table1.ColB, Table2.ColC  
    FROM Table1, Table2  
    WHERE Table1.ColA = Table2.ColA;
```

Создание триггера, который позволит производить удаление данных в таблицах Table1, Table2:

```
CREATE TRIGGER TableView_Delete FOR TableView BEFORE DELETE  
AS  
    BEGIN  
        DELETE FROM Table1  
            WHERE ColA = OLD.ColA;  
        DELETE FROM Table2  
            WHERE ColA = OLD.ColA;  
    END;
```

Создание триггера, который позволит производить обновление данных в таблицах Table1, Table2:

```
CREATE TRIGGER TableView_Update FOR TableView  
BEFORE UPDATE AS  
    BEGIN  
        UPDATE Table1  
            SET ColB = NEW.ColB  
            WHERE ColA = OLD.ColA;  
        UPDATE Table2  
            SET ColC = NEW.ColC  
            WHERE ColA = OLD.ColA;  
    END;
```

Создание триггера, который позволит производить выполнять ввод данных в таблицах Table1, Table2:

```
CREATE TRIGGER TableView_Insert FOR TableView BEFORE INSERT AS  
BEGIN  
    INSERT INTO Table1 values (NEW.ColA,NEW.ColB);  
    INSERT INTO Table2 values (NEW.ColA,NEW.ColC);  
END;
```

2.6. Использование исключений в триггерах

Исключение - это именованное сообщение об ошибке, которое может быть вызвано из триггера или хранимой процедуры. Исключения создаются с помощью оператора **CREATE EXCEPTION**, модифицируется с помощью **ALTER EXCEPTION**, и удаляются с помощью **DROP EXCEPTION**.

Исключения - это объекты базы данных, они хранятся в системной таблице и могут использоваться любой процедурой или триггером.

Если исключение вызывается в хранимой процедуре или триггере, исключение возвращает строковое сообщение вызывающей программе и завершает выполнение блока операторов, если исключение не обрабатывается с помощью оператора **WHEN** ниже в тексте процедуры или триггера.

Например, если количество неудачных попыток зайти в систему (каждая попытка записывается в таблицу) превышает 5, можно выводить предупреждающее сообщение.

2.6.1. Генерация исключения

Для генерации исключения в триггере нужно воспользоваться следующим оператором:

EXCEPTION name;

здесь name - это имя существующего исключения.

Пример создания исключения:

CREATE EXCEPTION RAISE_TOO_HIGH 'New salary exceeds old
by more than 50%. Cannot update record.';

Пример вызова исключения из триггера:

SET TERM !! ;

CREATE TRIGGER SAVE_SALARY_CHANGE
FOR EMPLOYEE

AFTER UPDATE AS

DECLARE VARIABLE PCNT_RAISE;

BEGIN

PCNT_RAISE=(NEW.SALARY-OLD.SALARY)*100/ OLD.SALARY;

IF (OLD.SALARY <> NEW.SALARY)

THEN IF (PCNT_RAISE > 50)

THEN EXCEPTION RAISE_TOO_HIGH;

ELSE BEGIN

INSERT INTO SALARY_HISTORY (EMP_NO, HANGE_DATE,
UPDATER_ID, OLD_SALARY, PERCENT_CHANGE)

VALUES (OLD.EMP_NO, 'NOW', USER, OLD.SALARY,

PCNT_RAISE);

END

END !!

SET TERM ; !!

3. Порядок выполнения лабораторной работы

- 1 В соответствии с вариантом задания создать триггеры.
- 2 Внести изменения в указанные таблицы, используя триггеры (не менее 5 строк). Просмотреть полученный результат.
- 3 Модифицировать созданные триггеры.
- 4 Написать отчет.

4. Содержание отчета

1. Отчет состоит из титульного листа, цели работы, описания процесса выполнения работы и вывода.
2. Отчет должен содержать: исходные данные, этапы работы по созданию представлений и триггеров, исходные и модифицированные таблицы, выборку информации из таблицы на каждом шаге работы.

5. Задание на работу

Таблица 3 – Задание на работу

Вариант	Задание
1	Автоматическая генерация идентификатора публикации. Создать обновляемые VIEW "СписокСтатей" и "СписокЖурналов"
2	Автоматическая генерация полей "Номер преподавателя", "Номер курса", "Номер студента". Не позволять студенту посещать более 5 курсов (выдавать сообщение).
3	Автоматическая генерация полей "Номер пациента", "Номер патента хирурга", "Номер операции". Не позволять назначать более пяти процедур на операцию.
4	Автоматическая генерация поля "Номер". Создать обновляемые VIEW "СписокДетей", "СписокРодителей" (представление объединяет таблицу "Личность" с таблицей "Ребенок" или "Родитель")
5	Автоматическая генерация полей "Номер фирмы", "Номер типа помещения", создать обновляемое представление "ПомещениеТип" (в таблицу "Помещение" вместо номера типа помещения подставить наименование типа помещения).
6	Автоматическая генерация полей "Код курса", "Номер служащего", "Код профессии", "Номер ставки", "Код работы". Не позволять служащему работать более чем на двух работах.
7	Автоматическая генерация полей "Номер служащего", "Номер события", "Номер языка", "Тип образования". Создать редактируемое представление "Список отпусков", объединив таблицы "Трудовая деятельность" и "Отпуск"

Продолжение таблицы 3

8	Автоматическая генерация полей "Номер специальности", "Номер факультета". Не позволять преподавателю читать более пяти курсов.
9	Автоматическая генерация поля "Номер предмета", создать редактируемое представление "Экзамены", состоящие из полей "Код специальности", "Наименование ВУЗа", "ФИО абитуриента", "Наименование предмета", "Оценка".
10	Автоматическая генерация полей "Номер поезда", "Номер станции". Создать редактируемое представление "Расписание", состоящие из полей "Дата рейса",
11	Автоматическая генерация полей "Номер факультета", "Номер предмета". Не разрешать студенту сдавать экзамены на двух факультетах.
12	Автоматическая генерация полей "Номер персоны", "Номер дела", "Номер псевдонима". Создать обновляемое представление "Псевдонимы", состоящее из полей "Номер персоны", "ФИО", "Псевдоним".
13	Автоматическая генерация поля "Номер соревнования". Не допускать проведения в год более пяти соревнований.
14	Автоматическая генерация полей "Номер зала", "Номер концерта", "Номер фирмы", "Номер артиста". Создать обновляемое представление "Программа концерта", состоящее из полей "Дата концерта", "Время концерта", "Номер зала", "ФИО артиста", "Номер по порядку в концерте", "Содержание выступления".
15	Автоматическая генерация полей "Код фирмы", "Номер номенклатуры". Создать обновляемое представление "Родственники", состоящее из полей "Табельный номер сотрудника", "ФИО сотрудника", "ФИО ребенка", "Год рождения ребенка".
16	Автоматическая генерация полей "Номер фирмы", "Номер водителя", "Номер груза". Не позволять водителю делать более пяти перевозок в день.
17	Автоматическая генерация полей "Номер гостиницы", "Номер фирмы". Создать обновляемое представление "Номера гостиницы", состоящее из полей "Название гостиницы", "Адрес", "Разряд", "Порядковый номер номера", "Количество комнат", "Разряд номера".
18	Автоматическая генерация полей "Номер диагноза", "Номер курса", "Номер прибора", "Номер препарата", "Номер метода", "Номер ограничения". Не позволять на курс лечения назначать более пяти препаратов.
19	Автоматическая генерация полей "Номер инженера", "Номер ВЦ", "Номер пакета", "Номер комплектующего". Создать редактируемое представление "Машины инженера", состоящее из всех не ключевых полей таблиц "Инженер" и "Машина".

Продолжение таблицы 3

20	Автоматическая генерация полей "Номер имущества", "Номер помещения", "Номер газеты". Не разрешать партии владеть более чем пятью газетами.
----	--

6. Контрольные вопросы

1. Что такое "триггер"?
2. Из каких частей состоит триггер?
3. Какая информация содержится в заголовочной части триггера?
4. Как сделать триггер временно неактивным? Как удалить триггер?
5. Для чего используются триггеры?
6. Назовите элементы языка хранимых процедур и триггеров.

