ИННОВАЦИОННО-ОБРАЗОВАТЕЛЬНАЯ ПРОГРАММА
Саратовского
государственного университета
им.Н.Г.Чернышевского

Факультет нелинейных процессов
Кафедра электроники, колебаний и волн
Кафедра нелинейной физики

А.А Короновский, О.И. Москаленко, П.В. Попов

КРАТКОЕ РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ ЯЗЫКА SQL. СОЗДАНИЕ БАЗ ДАННЫХ И ТАБЛИЦ, ИЗМЕНЕНИЯ ТАБЛИЦ, ОГРАНИЧЕНИЯ ЦЕЛОСТНОСТИ ДАННЫХ, ВВОД ДАННЫХ В ТАБЛИЦЫ

Учебно-методическое пособие по курсу
«Высокоуровневые методы программирования
и информационные системы. Принципы построения
и функционирования баз данных»

Саратов – 2008

Содержание

Введение

В современном мире все построено на информационных и телекоммуникационных системах. Во всех сферах человеческой деятельности (торговле, медицине, образовании, промышленности и т.д.) первостепенную роль в организационных процессах играют средства накопления, обработки и передачи данных. Базы данных и программные сферы их обработки лежат в основе многих сфер человеческой деятельности, определяя развитие общественных процессов. Малейший сбой в функционировании той или иной базы данных приведет к возникновению больших проблем для населения.

Для создания приложений, управляющих информационными потоками, существует множество развитых средств, ориентированных на современные языки программирования. В то же самое время, все взаимодействие с базами данных в любом случае сводится к применению структурированного языка запросов SQL (Structured Query Language).

Цикл методических пособий по учебному курсу "Высокоуровневые методы программирования и информационные системы. Принципы построения и функционирования баз данных" представляет собой основы использования языка SQL на примере работы MS SQL SERVER. Данное методическое пособие — первое в этом цикле. В нем демонстрируется использование клиентской программы Microsoft Query Analyzer для создания несложной базы данных и работы с ней. В этом пособии дано подробное описание процесса создания и использования базы данных (создание таблиц и их изменение, ограничения целостности данных, ввод данных в таблицы, удаление и изменение введенных данных), приведены методические указания, подобран набор индивидуальных заданий для студентов и сформулированы вопросы для самопроверки. Так как это учебное пособие, многие детали при изложении теоретических вопросов пришлось опустить. Дополнительную информацию о рассмотренных вопросах можно получить из соответствующих разделов справки или рекомендуемой литературы.

1 Краткие теоретические сведения

1.1 Microsoft Query Analyzer

Query Analyzer предназначен для выполнения запросов и анализа их исполнения. Для работы с SQL Query Analyzer необходимо сначала подсоединиться к серверу. Для этого обычно требуется ввести имя пользователя и пароль. Необходимо также указать имя хоста (тот компьютер, на котором запущен сервер). Параметры соединения (а именно — соответствующие имя хоста, имя пользователя и пароль) можно узнать у администратора.

После установки соединения откроется окно SQL Query Analyzer. Общий вид окна приведен на рис. 1. Окно Query Analyzer разделено

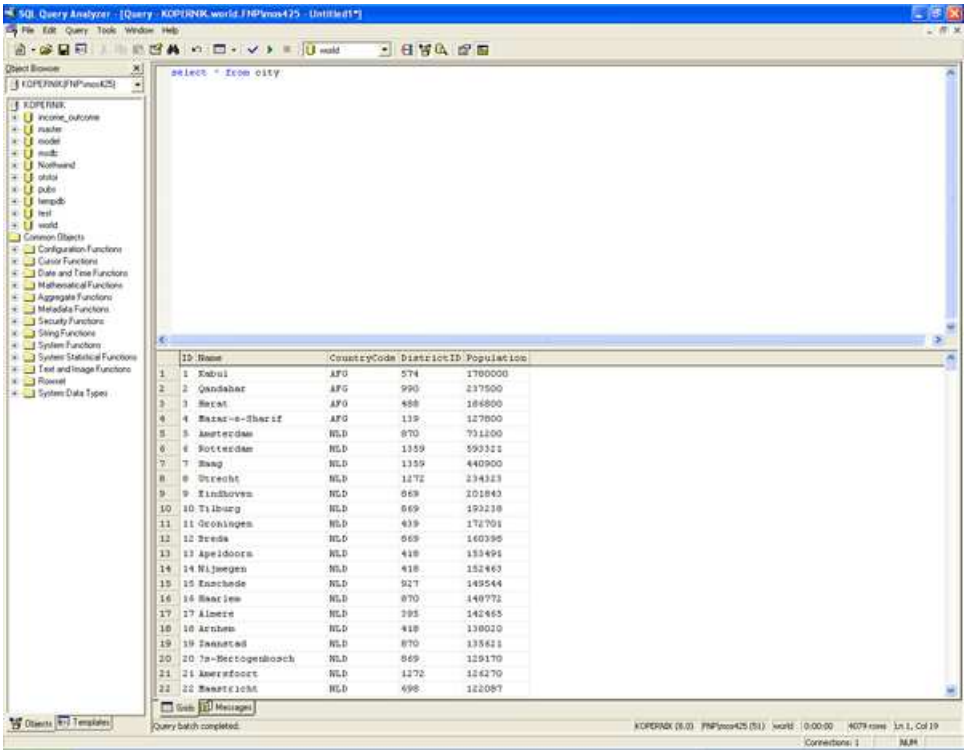

Рис. 1: Общий вид окна Microsoft Query Analyzer

на три части. В левой части выводится так называемый браузер объектов (**Object Browser**), с помощью которого можно посмотреть список всех объектов, расположенных в любой базе данных сервера, а также перечень встроенных функций и системных типов данных. Включение/отключение браузера объектов производится нажатием клавиши

<F8> или при помощи меню **Tools** (команда **Object Browser**, а затем команда **Show/hide**).

Верхняя часть окна SQL Query Analyzer предназначена для написания самих запросов. Для их выполнения необходимо выбрать команду **Execute Query** (зеленый треугольник) или нажать клавишу <F5>. При вводе ключевых слов в запросе можно использовать любой регистр символов. Результат выполнения запроса будет отображен в нижней части окна SQL Query Analyzer (см. рис. 1). В MS SQL SERVER существует возможность вывода информации в виде текста и в виде таблице. Во втором случае эта часть окна будет содержать две вкладки, **Grids** и **Messages**.

1.2 Создание структуры базы данных

1.2.1 Создание базы данных

Для создания базы данных в SQL используется оператор `CREATE DATABASE`. Его базовый синтаксис следующий:

```
CREATE DATABASE db_name
```

Здесь **db_name** — название базы данных. Оно должно быть уникальными внутри сервера и может состоять не более чем из 128 допустимых символов. Полный синтаксис оператора `CREATE DATABASE` можно найти в разделе **Transact-SQL Reference** справки по MS SQL SERVER.

▶ **Пример 1.** Рассмотрим создание базы данных **test**. Для этого нужно подать команду:

```
CREATE DATABASE test
```

В окне Messages будет выведен текст:

```
The CREATE DATABASE process is allocating 0.75 MB on disk
'test'.
The CREATE DATABASE process is allocating 0.49 MB on disk
'test_log'.
```

Вывод такого сообщения говорит о том, что база данных **test** создана.

Для удаления базы данных нужно подать команду

```
DROP DATABASE db_name
```

1.2.2 Выбор активной базы данных

Узнать, какие базы данных существуют в настоящее время на сервере, можно при помощи хранимой процедуры *sp_databases*.

Для того чтобы работать с конкретной базой данных надо сделать ее активной. Для этого служит команда `USE`:

```
USE db_name
```

▶ **Пример 2.**

```
USE test
```

1.2.3 Создание таблиц

Для создания таблиц используется оператор `CREATE TABLE`.

Вывести список уже имеющихся таблиц в текущей базе данных можно с помощью хранимой процедуры *sp_tables*. С помощью следующего запроса можно вывести информацию об имеющихся *пользовательских* таблицах:

```
sp_tables @table_name = '[^sys]%'
```

Базовый синтаксис оператора `CREATE TABLE` следующий:

```
CREATE TABLE table_name
    ( { < column_definition >
        | < table_constraint > }
        | [ { PRIMARY KEY | UNIQUE } ]
        [ ,...n ]
    )

< column_definition > ::= { column_name data_type }
    [ DEFAULT constant_expression
        | IDENTITY [ (seed,increment) ] ]
    [ < column_constraint > ] [ ...n ]

< column_constraint > ::= [ CONSTRAINT constraint_name ]
    { [ NULL | NOT NULL ]
        | [ { PRIMARY KEY | UNIQUE } ]
        | [ [ FOREIGN KEY ]
            REFERENCES ref_table [ ( ref_column ) ]
```

```
            [ ON DELETE { CASCADE | NO ACTION } ]
            [ ON UPDATE { CASCADE | NO ACTION } ]
          ]
    }

< table_constraint > ::= [ CONSTRAINT constraint_name ]
    { [ { PRIMARY KEY | UNIQUE }
        { ( column [ ,...n ] ) }
      ]
        | FOREIGN KEY
          [ ( column [ ,...n ] ) ]
          REFERENCES ref_table [ ( ref_column [ ,...n ] ) ]
          [ ON DELETE { CASCADE | NO ACTION } ]
          [ ON UPDATE { CASCADE | NO ACTION } ]
    }
```

Полный синтаксис оператора `CREATE TABLE` можно найти в разделе **Transact-SQL Reference** справки по MS SQL SERVER.

В MS SQL SERVER каждый столбец, локальная переменная, выражение и параметр имеет определенный тип данных. MS SQL SERVER поддерживает набор системных типов данных, которые определяют все типы данных, которые могут быть использованы MS SQL SERVER. Список наиболее часто используемых из них приведен ниже.

- Символьные типы данных — содержат буквы, цифры и специальные символы.

  CHAR или CHAR(n) — символьные строки фиксированной длины. Длина строки определяется параметром n. CHAR без параметра соответсвует CHAR(1). Максимальное значение n = 8000;

  VARCHAR(n) — символьная строка переменной длины. Максимальная длина строки составляет 8000 символов;

  TEXT — символьная строка переменной длины с максимальной длиной 2147483647 символов.

- Целые типы данных — поддерживают только целые числа (дробные части и десятичные точки не допускаются). Над этими типами разрешается выполнять арифметические операции и применять к ним агрегирующие функции (определение максимального, минимального, среднего и суммарного значения столбца реляционной таблицы).

BIGINT — целое в интервале -2^{63} до $2^{63}-1$;

INTEGER или INT — целое число в интервале значений от -2147483647 до 2147483648;

SMALLINT — короткое целое, интервал значений от -32767 до 32768;

TINYINT — целые числа в интервале 0 до 255.

- Вещественные типы данных — описывают числа с дробной частью.

  FLOAT и REAL — числа с плавающей точкой (диапазон от $-1.79 \cdot 10^{308}$ до $1.79 \cdot 10^{308}$ и от $-3.4 \cdot 10^{38}$ до $3.4 \cdot 10^{38}$, соответственно);

  DECIMAL(p)и NUMERIC(p) — типы данных аналогичные FLOAT с числом значащих цифр p (диапазон от $-10^{38}+1$ до $10^{38}+1$).

  DECIMAL(p,n) и NUMERIC(p,n) — аналогично предыдущему, p — общее количество десятичных цифр, n — количество цифр после десятичной запятой.

- Денежные типы данных — описывают денежные величины.

  MONEY — значения денежного типа в диапазоне -2^{63} до 2^63-1 с точностью до десяти тысячных денежной единицы;

  SMALLMONEY — значения денежного типа в диапазоне -214748.3648 до 214748.3647 с точностью до десяти тысячных денежной единицы;

- Дата и время — используются для хранения даты, времени и их комбинаций.

  DATETIME — тип данных для хранения моментов времени с 1 января 1753 до 31 декабря 9999 с точностью до трех сотых секунды или 3.33 милисекунд;

  SMALLDATETIME — аналогично DATETIME с 1 января 1900 до 6 июня 2079 с точностью до 1 минуты.

- Двоичные типы данных — позволяют хранить данные любого объема в двоичном коде (оцифрованные изображения, исполняемые файлы и т.д.).

BINARY — двоичное число фиксированной длины до 8000 байт;

VARBINARY — двоичное число переменной длины до 8000 байт;

IMAGE — двоичное число переменной длины до $2^{31} - 1$ байт.

Для всех типов данных имеется общее значение `NULL` — "не определено". Это значение имеет каждый элемент столбца до тех пор, пока в него не будут введены данные. При создании таблицы можно явно указать, могут ли элементы того или иного столбца иметь значения `NULL` (это не допустимо, например, для столбца, являющего первичным ключом).

Для того, чтобы лучше понять, каким образом работает оператор `CREATE TABLE`, рассмотрим несколько примеров. Создадим в базе данных **test** несколько таблиц.

▶ **Пример 3.** Простая таблица с двумя полями:

```
CREATE TABLE table1(
id int,
name varchar(50) )
```

В этой таблице не определен первичный ключ.

▶ **Пример 4.** Если первичным ключом является единичное поле, то можно сразу указать факт наличия ключа:

```
CREATE TABLE table2 (
id int PRIMARY KEY,
name varchar(50) )
```

Также в определении поля таблицы можно указывать дополнительные опции, такие как

- значение по-умолчанию (`DEFAULT`)
- допустимость/не допустимость(по-умолчанию) значения `NULL` (`NULL | NOT NULL`)
- флаг первичного ключа (`PRIMARY KEY`)
- флаг внешнего ключа (`FOREIGN KEY`)

- флаг идентифицирующего значения (`IDENTITY(начальное значение, инкремент)`)
- и др.

▶ **Пример 5.** Создадим тестовую таблицу, атрибуты которой содержит все вышеперичесленные опции:

```
CREATE TABLE table3 (
id int PRIMARY KEY IDENTITY(1,1),
name varchar(50) NOT NULL,
title varchar(50) NOT NULL DEFAULT 'Vice President',
referenced_id int NOT NULL FOREIGN KEY REFERENCES table2 (id))
```

Если таблица содержит составные ключи (первичные или внешние), то этот факт указывается отдельно.

▶ **Пример 6.** Создание таблицы с первичным ключом из двух полей можно осуществить следующим образом:

```
CREATE TABLE table4 (
id1 int NOT NULL IDENTITY(1,1),
id2 char(3) NOT NULL,
name varchar(10),
PRIMARY KEY (id1,id2))
```

1.2.4 Изменение структуры существующей таблицы

В реляционных СУБД существует возможность изменять структуру существующих таблиц (возможно даже таблиц с данными). Для изменения структуры существующей таблицы используется оператор `ALTER TABLE`.

Полный синтаксис оператора `ALTER TABLE` можно найти в разделе **Transact-SQL Reference** справки по MS SQL SERVER. В этом методическом пособии будут рассмотрены только случаи изменения типа данных столбца в таблице, добавление в таблицу нового столбца, добавление в таблицу первичного и/или внешнего ключей (если они не были определены на этапе создания таблицы).

Изменить тип данных существующего столбца таблицы можно следующим образом:

```
ALTER TABLE table_name ALTER COLUMN column_name new_data_type
[(precision [, scale])] [ NULL | NOT NULL ]
```

Существует ряд ограничений на изменение типа столбца (например, нельзя изменить тип данных со строкового на числовой или если изменяемый столбец является частью первичного ключа). Также при преобразовании типов может произойти потеря данных.

Чтобы добавить в таблицу новый столбец необходимо пользоваться следующим синтаксисом оператора `ALTER TABLE`:

```
ALTER TABLE table_name ADD <column_definition> [,...n]
```

Здесь <*column_definition*> — определение столбца, аналогичное таковому в операторе `CREATE TABLE`.

▶ **Пример 7.** Добавим столбец к таблице **table1**, созданной ранее:

```
ALTER TABLE table1
ADD description varchar(50) NULL DEFAULT ''''
```

Для того чтобы добавить первичный или внешний ключ надо использовать следующую запись оператора `ALTER TABLE`:

```
ALTER TABLE table_name ADD PRIMARY KEY (column_name [,...n])
```

или

```
ALTER TABLE table_name ADD FOREIGN KEY (column_name [,...n])
REFERENCES ref_table(ref_column [,...n])
```

▶ **Пример 8.** Сделаем поле **id** таблицы **table1** первичным ключом:

```
ALTER TABLE table1
ADD PRIMARY KEY (id)
```

▶ **Пример 9.** Теперь сделаем поле **name** этой же таблицы внешним ключом поля **name** таблицы **table2**:

```
ALTER TABLE table1
ADD FOREIGN KEY (name) REFERENCES table2 (name)
```

В поле **Messages** появится сообщение об ошибке. Сервер выдал ошибку о том, что он не может создать внешний ключ на поле родительской таблицы, которое само не является первичным или потенциальным ключом.

В случае ссылки на первичный ключ в дочерней таблице, или на колонку с уникальными значениями (в которой добавлено ограничение UNIQUE), подобной ошибки не возникло бы.

Для того чтобы удалить из таблицы существующую колонку или какое-либо ограничение (первичный или внешний ключ, индекс и т.д.) надо дать команду:

```
ALTER TABLE table_name DROP {[CONSTRAINT] constraint_name |
COLUMN column} [,...n]
```

Здесь constraint_name — имя ограничения. Для первичного ключа оно обычно начинается с букв PK, для внешнего ключа — с букв FK (такие имена по-умолчанию даются системой, но можно их задать вручную на этапе создания ограничений). Список всех ограничений можно посмотреть, запустив хранимую процедуру *sp_helpconstraint*. Она имеет один входной параметр — имя таблицы:

```
sp_helpconstraint table_name
```

Она выведет список всех имеющихся в таблице ограничений. Искомые имена ограничений будет состоять из первых двух букв, описывающих тип ограничения, а затем через подчеркивание — описание таблиц и колонок, которые это ограничение связывает.

▶ **Пример 10.** Удалим первичный ключ из таблицы **table1**:

```
ALTER TABLE table1 DROP CONSTRAINT
PK__table1__07020F21
```

Если теперь запустить процедуру *sp_helpconstraint* для этой таблицы, то первичного ключа в ней уже не будет. Точно также удаляются и внешние ключи.

1.3 Ввод, удаление и модификация данных

1.3.1 Ввод данных в таблицы

Для ввода данных в существующую таблицу в MS SQL SERVER используется оператор INSERT следующего базового синтаксиса:

```
INSERT [INTO] table_name [(column_list)]
    {
        { VALUES
            ( { DEFAULT | NULL | expression } [ ,...n] )
```

```
    }
}
| DEFAULT VALUES
```

Здесь сокращения означают следующее:

- table_name — имя таблицы;
- (column_list) — список столбцов таблицы из одного или более элементов, разделенных запятыми, в которые необходимо вставить данные. Список может отсутсвовать, но тогда придется вносить значения в каждый столбец таблицы (причем очередность столбцов такая же, как при создании таблицы). Если существующий столбец таблицы не входит в column_list, то СУБД должна суметь заполнить его значение самостоятельно. Иначе будет выдано сообщение об ошибке. СУБД сама может заполнить следующие столбцы:

  столбцы, имеющие свойство `IDENTITY`. По умолчанию непосредственный ввод данных в такие столбцы запрещен;

  столбцы, имеющие значение по умолчанию (`DEFAULT`);

  столбцы, имеющие тип данных timestamp. В них будет записано текущее время;

  столбцы с атрибутом `NULL`. Будет записано значение `NULL`;

- expression — корректное выражение SQL, возвращающее подходящее для столбца значение.

▶ **Пример 11.** Чтобы проиллюстрировать, каким образом работает оператор INSERT, добавим данные в таблицу **table4**. Значение **id1** вручную прописать нельзя (запрещает свойство `IDENTITY`), поэтому нужно добавлять данные только в остальные два столбца:

```
INSERT INTO table4 VALUES ('key1', 'first value')
(1 row affected)

INSERT INTO table4 VALUES ('key2', 'second value')
(1 row affected)

INSERT INTO table4 VALUES ('key3', 'third value')
(1 row affected)
```

```
INSERT INTO table4 VALUES ('key3', 'third value')
(1 row affected)

INSERT INTO table4 VALUES ('key3', 'third value')
(1 row affected)
```

? **Вопрос 1.** *Подумайте, почему последних три добавления данных прошли успешно, хотя данные были добавлены абсолютно идентичные.*

? **Вопрос 2.** *Посмотрите на тип данных столбцов, и подумайте, какие данные реально внесены в таблицу.*

Чтобы посмотреть данные, внесенные предыдущими операторами в таблицу **table4**, нужно подать команду:

```
SELECT * FROM table4

 id1 id2   name
 --- --- ----------
  1 key first colu
  2 key second val
  3 key third valu
  4 key third valu
  5 key third valu
(5 rows affected)
```

Для выборки данных из таблицы был использован оператор `SELECT`. Его синтаксис будет рассмотрен во втором методическом пособии по основам SQL "Выборка данных в SQL, объединение таблиц, создание представлений и хранимых процедур". Сейчас скажем только, что примененная форма выбирает все записи из таблицы.

Следующие несколько строчек также добавляют данные в таблицу

```
INSERT INTO table4 (id2, name) VALUES ('k1', 'val1')
INSERT INTO table4 (name, id2) VALUES ('val2', 'k2')
INSERT INTO table4 (id2, name) VALUES ('k3', 'val3')
INSERT INTO table4 (id2, name) VALUES ('k2', 'val3')
```

Обратите внимание на 2ю и 3ю строки.

Теперь в таблице **table4** содержатся следующие записи

```
SELECT * FROM table4
 id1 id2   name
 --- --- ----------
```

```
  1 key first colu
  2 key second val
  3 key third valu
  4 key third valu
  5 key third valu
  6 k1  val1
  7 k2  val2
  8 k3  val3
  9 k2  val3
(9 rows affected)
```

Теперь рассмотрим, как работает оператор `INSERT` с внешними ключами. Для этого будем использовать таблицы **table2** и **table3**. Для начала внесем данные в таблицу **table2**:

```
INSERT INTO table2 VALUES (1, 'John Doe')
INSERT INTO table2 VALUES (3, 'Jane Doe')
INSERT INTO table2 VALUES (8, 'John Travolta')
INSERT INTO table2 VALUES (10, 'John Doe Doe')
```

Внесенные данные:

```
SELECT * FROM table2
 id     name
 --- ---------------
  1   John Doe
  3   Jane Doe
  8   John Travolta
  10  John Doe Doe
(4 rows affected)
```

Теперь попробуем внести данные в зависимую от **table2** таблицу **table3**:

```
INSERT INTO table3 VALUES ('Wife', DEFAULT, 1)
(1 row affected)
INSERT INTO table3 VALUES ('Wife', DEFAULT, 2)
Msg 547, Level 16, State 0, Server ********, Line 1 INSERT statement
conflicted with COLUMN FOREIGN KEY constraint
'FK__table3__refer__03317E3D'. The conflict occurred in database
'test', table 'table2', column 'id'. The statement has been
terminated. (0 rows affected)
INSERT INTO table3 VALUES ('Wife', 'Nobody', 3)
(1 row affected)
```

? **Вопрос 3.** *Разберитесь, почему второй оператор вызвал ошибку.*

Внесенные данные:

```
SELECT * FROM table3
 id    name        title         referenced_id
 --- -------- ----------------- -------------
  1   Wife       Vice President         1
  3   Wife       Nobody                 3
(2 rows affected)
```

1.3.2 Удаление данных из таблицы

Для удаления существующих данных из таблицы (удаления строк) служит оператор `DELETE`, имеющий следующий базовый синтаксис:

```
DELETE [FROM] table_name [WHERE condition]
```

Здесь необязательным является условие, по которому удаляется только часть записей. Если условие не указывать, то будут удалены все записи из таблицы.

Синтаксис условия condition достаточно прозрачен, и его подробное описание можно найти в справке. Здесь будет рассмотрено несколько примеров использования оператора `DELETE`.

▶ **Пример 12.** Удалим из таблицы **table4** данные, соответствующие **id1**=6:

```
DELETE FROM table4 WHERE id1=6
(1 row affected)
```

Команда

```
DELETE FROM table4 WHERE id1>6
```

удаляет из таблицы все записи, удовлетворяющие условию **id1**>6 (в данном случае три строки таблицы).

Возможны также следующие варианты команды `DELETE`:

- удаление данных в заданном интервале значений **id1**

  ```
  DELETE FROM table4 WHERE id1<5 AND id1>3
  ```

- удаление всех данных из таблицы

```
DELETE FROM table4
```

Теперь попробуем удалить данные из таблицы **table2**:

```
DELETE FROM table2
```

Сервер выдаст сообщение об ошибке:

```
Msg 547, Level 16, State 0, Server ******, Line 1 DELETE statement
conflicted with COLUMN REFERENCE constraint
'FK__table3__refer__03317E3D'. The conflict occurred in database
'test', table 'table3', column 'referenced_id'. The statement has
been terminated.
```

Из-за того, что на данные в таблице ссылается внешний ключ в таблице **table3** данные удалить не удается. Причем, оператор `DELETE` аварийно завершается уже на первой строчке (с **id**=1).

Поэтому перед удалением данных из родительской таблицы нужно сначала удалить данные из дочерней таблицы, которые ссылаются на данные в родительской таблице, а затем удалить данные из последней.

```
DELETE FROM table3

DELETE FROM table2
```

1.3.3 Изменение данных в таблице

Оператор `UPDATE` изменяет имеющиеся данные в таблице. Команда имеет следующий синтаксис

```
UPDATE table_name
SET {column_name = {expression | DEFAULT | NULL }
        | @variable = expression
        | @variable = column = expression } [ ,...n ]
[WHERE condition];
```

С помощью одного оператора могут быть заданы значения для любого количества столбцов. Однако в одном и том же операторе `UPDATE` можно вносить изменения в каждый столбец указанной таблицы только один раз. При отсутствии условия WHERE будут обновлены все строки таблицы.

Если столбец допускает NULL-значение, то его можно указать в явном виде. Кроме того, можно заменить имеющееся значение на значение по умолчанию (DEFAULT) для данного столбца.

Ссылка на выражение "expression" может относиться к текущим значениям в изменяемой таблице. Например, можно увеличить все **id** в таблице **table1** на 1 с помощью следующего оператора:

```
UPDATE table1 SET id=id+1
```

Разрешается также значения одних столбцов присваивать другим столбцам, естественно, если типы данных этих столбцов являются совместимыми.

Если требуется изменять данные в зависимости от содержимого некоторого столбца, можно воспользоваться выражением CASE. Предположим, нужно положить значение в поле **description** таблицы **table1** равным true для всех **id**<5 и равным false для всех остальных **id**.

```
UPDATE table1 SET description = CASE WHEN id<5 THEN 'TRUE' ELSE
'FALSE' END
```

Необходимо сказать несколько слов о столбцах, имеющих свойство IDENTITY. Если столбец **id** в таблице **table3** определен как IDENTITY(1,1), то оператор

```
UPDATE table3 SET ide=5 WHERE id=4
```

выполнен не будет, так как идентифицированное поле не допускает обновления, и будет выдано соответствующее сообщение об ошибке.

2 Методические указания

Каждый студент выполняет индивидуальное задание. При выполнении практических заданий студенту следует сначала внимательно изучить теорию, описанную в разделе 1 данного методического пособия (в случае необходимости студент может использовать дополнительную литературу, список которой приведен ниже в разделе 4), ответить на контрольные вопросы и лишь затем приниматься за работу с MS SQL SERVER. Следует обратить внимание на тот факт, что преподаватели *не дают* индивидуальных консультаций по поводу выполнения того или иного задания, а только проверяет корректность написанных студентом запросов и предлагают пути устранения возникших ошибок. Отчет происходит в индивидуальном порядке.

Для получения зачета по теме “Создание баз данных и таблиц, изменения таблиц, ограничения целостности данных, ввод данных в таблицы” студент должен выполнить все задания, приведенные в разделе 3, и продемонстрировать полученные результаты преподавателю с использованием выбранной СУБД. Знания и умения студента должны удовлетворять всем приведенным ниже требованиям.

Требования к умениям студентов

После прохождения практических занятий по курсу “Высокоуровневые методы программирования и информационные системы. Принципы построения и функционирования баз данных ” по теме “Создание баз данных и таблиц, изменения таблиц, ограничения целостности данных, ввод данных в таблицы” студент должен научиться манипулировать базами данных посредством Transact-SQL: уметь создавать и удалять базу данных, делать ее активной, работать с таблицами. Студент должен знать основные команды Transact-SQL и их базовый синтаксис: уметь получать информацию об имеющихся в базе данных пользовательских и системных таблицах, создавать, удалять и изменять таблицы, определять в них первичные и внешние ключи как на этапе создания, так и при изменении структуры, знать основные типы данных, поддерживаемые SQL, а также дополнительные опции атрибутов, уметь изменять тип заданного столбца, добавлять в таблицу новый столбец, знать основные ограничения на то или иное изменение и уметь от них избавиться. Кроме того, он должен уметь вводить данные в таблицы, знать основные трудности, которые могут возникнуть при вводе данных в случае наличия тех или иных опций атрибутов, а также вносить изменения во введенные данные и удалять их в случае возникновения необходимости как полностью, так и частично. Он должен уметь получать информацию о введенных данных посредством простой выборки всех данных из одной таблицы.

3 Задания

Выбор задания для студента осуществляется преподавателем на первом занятии при ознакомлении с требованиями, предъявляемыми к студентам и методикой проведения семинаров, а также при раздаче заданий по теме “Нормализация универсальных отношений”. Студентам предлагается выполнить приведенные ниже задания для нормализованной им базы данных. *Студент получает допуск к выполнению заданий по теме “Создание баз данных и таблиц, изменения таблиц, ограничения целост-*

ности данных, ввод данных в таблицы" только после сдачи промежуточного зачета по теме "Нормализация универсальных отношений".

1. Создайте, используя операторы `CREATE DATABASE`, `CREATE TABLE`, базу данных, нормализованную Вами при выполнении задания по теме "Нормализация универсальных отношений". Определите в таблицах первичные ключи. Свяжите таблицы внешними ключами

2. Введите данные в таблицы Вашей базы данных. Разберите операторы удаления `DELETE` и модифкации `UPDATE` данных. Попробуйте удалить/изменить данные из таблиц, на которые ссылаются другие таблицы.

3. Изучите оператор `ALTER TABLE`. Измените несколько таблиц в базе данных. Удалите внешние ключи. Измените первичные ключи в таблицах. Добавьте внешние ключи. Рассмотрите случаи различной работы операторов `DELETE`, `INSERT` и `UPDATE` при работе с зависимыми данными (при различной настройке работы внешних ключей).

4 Рекомендуемая литература

1. Астахова И.Ф., Толстобров А.П., Мельников В.М. SQL в примерах и задачах. Учебное пособие — Мн.: Новое знание, 2002.
2. Баженова И.Ю. Основы проектирования приложений баз данных. Изд-во: Интернет-университет информационных технологий, 2006.
3. Дж. Боуман, С. Эмерсон, М. Дарновски. Практическое руководство по SQL. М., 2001.
4. Гарсиа М.Ф. и др. Microsoft SQL Server. Справочник администратора. ЭКОМ: Москва, 2002.
5. Кузнецов С.Д. Основы баз данных. Изд-во: Интернет-университет информа-ционных технологий, 2005.
6. Мамаев Е. MS SQL Server 2000. Изд.: БХВ-Петербург, 2005.
7. Марков А.С., Лисовский К.Ю. Базы данных. Введение в теорию и методологию. М.: Финансы и статистика, 2006.
8. Моисеенко С. SQL. Задачи и решения. Спб.: Питер, 2006.
9. Уилтон П., Колби Дж. Язык запросов SQL для начинающих. М.: Диалектика, 2006.
10. Фленов М.Е. Transact–SQL. СПб: БХВ–Петербург, 2006.
11. Харрингтон Дж.Л. Проектирование реляционных баз данных. Изд. “Лорри”, 2006.
12. Хендерсон К. Профессиональное руководство по Transact–SQL. Изд. Питер, 2006.
13. М.Дж. Хернанденс, Дж. Л. Вьескас. SQL запросы для простых смертных. Изд. “Лори”, 2003.
14. Хоторн Р. Разработка баз данных Microsoft SQL Server 2000 на примерах. М.: Издат. дом “Вильямс”, 2001.
15. Шпенник М., Слендж О. Руководство администратора баз данных Microsoft SQL Server 2000. М.: Издат. дом “Вильямс”, 2001.
16. Раздел Transact-SQL Reference справки по MS SQL SERVER 2000.