МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт кибербезопасности и цифровых технологий

Кафедра КБ-2 «Информационно-аналитические системы кибербезопасности»

Кафедра защиты информации (КБ-2)
Методические рекомендации для практической работы по дисциплине «Безопасность систем баз данных»
10.05.03 «Создание автоматизированных систем в защищенном исполнении»

Москва 2024

Лабораторная работа 3 Язык описания данных

- **Создание объектов баз данных**
- **Модифицирование объектов баз данных**
- **Удаление объектов баз данных**

В этой работе рассматриваются все инструкции Transact-SQL, связанные с языком описания данных DDL (Data Definition Language). Инструкции языка DDL разбиты на три группы, которые рассматриваются последовательно. В первую группу входят инструкции для создания объектов, вторая содержит инструкции для модифицирования структуры объектов, а третья состоит из инструкций для удаления объектов.

Создание объектов баз данных

В организации базы данных задействуется большое число различных объектов. Все объекты базы данных являются либо физическими, либо логическими. *Физические объекты* связаны с организацией данных на физических устройствах (дисках). Физическими объектами компонента Database Engine являются файлы и файловые группы. *Логические объекты* являются пользовательскими представлениями базы данных. В качестве примера логических объектов можно назвать таблицы, столбцы и представления (виртуальные таблицы).
Объектом базы данных, который требуется создать в первую очередь, является сама база данных. Компонент Database Engine управляет как системными, так и пользовательскими базами данных. Пользовательские базы данных могут создаваться авторизованными пользователями, тогда как системные базы данных создаются при установке системы базы данных.
В этой работе рассматривается создание, изменение и удаление пользовательских баз данных.

Создание базы данных

Для создания базы данных используется два основных метода. В первом методе задействуется обозреватель объектов среды SQL Server Management Studio (см. *предыдущую работу*), а во втором применяется инструкция языка Transact-SQL create database. Далее приводится общая форма этой инструкции, а затем подробно рассматриваются ее составляющие:

```
CREATE DATABASE db_name
        [ON [PRIMARY] {file_specl},.]
        [LOG ON {file_spec2},.]
                [COLLATE collation_name]
    [ FOR {ATTACH ATTACH_REBUILD_LOG}]
```

ПРИМЕЧАНИЕ

Синтаксис инструкций языка Transact-SQL представляется соответственно соглашениям. По этим соглашениям необязательные элементы инструкций приводятся в квадратных скобках []. В фигурных скобках {} с последующим троеточием представлены элементы, которые можно повторять любое количество раз.

Параметр *db_name* — это *имя базы данных.* Имя базы данных может содержать максимум 128 символов. (Правила для идентификаторов, рассмотренные в *предыдущей работе,* применяются к именам баз данных.) Одна система может управлять до 32 767 базами данных.
Все базы данных хранятся в файлах, которые могут быть указаны явно администратором или предоставлены неявно системой. Если инструкция CREATE DATABASE содержит параметр ON, все файлы базы данных указываются явно.

ПРИМЕЧАНИЕ

Компонент Database Engine хранит файлы данных на диске. Каждый файл содержит данные одной базы данных. Эти файлы можно организовать в файловые группы. Файловые группы предоставляют возможность распределять данные по разным приводам дисков и выполнять резервное копирование и восстановление частей базы данных. Это полезная функциональность для очень больших баз данных.

Параметр *fiie_specl* представляет спецификацию файла и сам может содержать дополнительные опции, такие как логическое имя файла, физическое имя и размер. Параметр primary

указывает первый (и наиболее важный) файл, который содержит системные таблицы и другую важную внутреннюю информацию о базе данных. Если параметр primary отсутствует, то в качестве первичного файла используется первый файл, указанный в спецификации.

Учетная запись компонента Database Engine, применяемая для создания базы данных, называется *владельцем базы данных.* База данных может иметь только одного владельца, который всегда соответствует учетной записи. Учетная запись, принадлежащая владельцу базы данных, имеет специальное имя dbo. Это имя всегда используется в отношении базы данных, которой владеет пользователь.

Опция log on параметра dbo определяет один или более файлов в качестве физического хранилища журнала транзакций базы данных. Если опция log on отсутствует, то журнал транзакций базы данных все равно будет создан, поскольку каждая база данных должна иметь, по крайней мере, один журнал транзакций. (Компонент Database Engine ведет учет всем изменениям, которые он выполняет с базой данных. Система сохраняет все эти записи, в особенности значения до и после транзакции, в одном или более файлов, которые называются журналами транзакций. Для каждой базы данных системы ведется ее собственный журнал транзакций.

В опции COLLATE указывается порядок сортировки по умолчанию для базы данных. Если опция COLLATE не указана, базе данных присваивается порядок сортировки по умолчанию базы данных model, совершенно такой же, как и порядок сортировки по умолчанию системы баз данных.

В опции FOR ATTACH указывается, что база данных создается за счет подключения существующего набора файлов. При использовании этой опции требуется явно указать первый первичный файл. В опции FOR ATTACH_REBUILD_LOG указывается, что база данных создается методом присоединения существующего набора файлов операционной системы. (Предмет присоединения и отсоединения базы данных рассматривается далее.)

Компонент Database Engine создает новую базу данных по шаблону образцовой базы данных model. Свойства базы данных model можно настраивать для удовлетворения персональных концепций системного администратора.

ПРИМЕЧАНИЕ

Если определенный объект базы данных должен присутствовать в каждой пользовательской базе данных, то этот объект следует сначала создать в базе данных model.

В примере 1 показан код для создания простой базы данных, без указания дополнительных подробностей. Чтобы исполнить этот код, введите его в редактор запросов среды Management Studio и нажмите клавишу <F5>.

Пример 1. Код для создания простой базы данных

```
USE master;
CREATE DATABASE sampleNew;
```

Код, приведенный в примере 1, создает базу данных, которая называется sampleNew. Такая сокращенная форма инструкции CREATE DATABASE возможна благодаря тому, что почти все ее параметры имеют значения по умолчанию. По умолчанию система создает два файла. Файл данных имеет логическое имя sample и исходный размер 2 Мбайта. А файл журнала транзакций имеет логическое имя sampie_iog и исходный размер 1 Мбайт. (Значения размеров обоих файлов, а также другие свойства новой базы данных зависят от соответствующих спецификаций базы данных model.)

В примере 2 показано создание базы данных с явным указанием файлов базы данных и журнала транзакций.

Пример 2. Создание базы данных с явным указанием файлов

```
USE master;
CREATE DATABASE projects ON
(NAME=projects_dat,
        FILENAME = 'C:\ projects.mdf',
        SIZE = 10,
        MAXSIZE = 100,
        FILEGROWTH = 5)
LOG ON
```

```
(NAME=projects_log,
    FILENAME = 'C:\ projects.ldf',
    SIZE = 40,
    MAXSIZE = 100,
    FILEGROWTH = 10);
```

Созданная в листинге 2 база данных называется project. Поскольку опция PRIMARY не указана, то первичным файлом предполагается первый файл. Этот файл имеет логическое имя projects_dat и он сохраняется в дисковом файле projects.mdf. Исходный размер этого файла 10 Мбайт. При необходимости, система выделяет этому файлу дополнительное дисковое пространство в приращениях по 5 Мбайт. Если не указать опцию MAXSIZE или если этой опции присвоено значение UNLIMITED, то максимальный размер файла может увеличиваться и будет ограничиваться только размером всего дискового пространства. (Единицу размера файла можно указывать с помощью суффиксов КВ, ТВ и МВ, означающих килобайты, терабайты и мегабайты соответственно. По умолчанию используется единица размера МВ, т. е. мегабайты.)

Кроме файла данных создается файл журнала транзакций, который имеет логическое имя projects_log и физическое имя projects.ldf. Все опции спецификации файла журнала транзакций имеют такие же имена и значения, как и соответствующие опции для спецификации файла данных.

В языке Transact-SQL можно указать конкретный контекст базы данных (т. е. какую базу данных использовать в качестве текущей) с помощью инструкции USE.
(Альтернативный способ — выбрать имя требуемой базы данных в раскрывающемся списке **Database** (Базы данных) в панели инструментов среды SQL Server Management Studio.)

Системный администратор может назначить пользователю текущую базу данных по умолчанию с помощью инструкции CREATE LOGIN или инструкции ALTER LOGIN. В таком случае пользователям не нужно выполнять инструкцию use, если только они не хотят использовать другую базу данных.

Создание моментального снимка базы данных

Кроме создания новой базы данных, инструкцию CREATE DATABASE можно применить для получения моментального снимка существующей базы данных (база данных-источник). Моментальный снимок базы данных является согласованной с точки зрения завершенных транзакций копией исходной базы данных на момент создания моментального снимка. Далее показан синтаксис инструкции для создания моментального снимка базы данных:

```
CREATE DATABASE database_snapshot_name ON (NAME =
    logical_file_name,
        FILENAME = 'os_file_name[r])' [,...n ]
        AS SNAPSHOT OF source_database_name
```

Таким образом, чтобы создать моментальный снимок базы данных, в инструкцию create database нужно вставить предложение as snapshot of.

В примере 3 иллюстрируется создание моментального снимка базы данных sample и сохранения его в папке C:\temp. (Прежде чем выполнять этот пример, нужно создать данный каталог.

Пример 3. Создание моментального снимка базы данных `Adventure Works`

```
USE master;
CREATE DATABASE sample_snapshot ON (NAME =
    'sample_Data' ,
        FILENAME = 'C:\temp\snapshot_DB.mdf')
        AS SNAPSHOT OF AdventureWorks;
```

Моментальный снимок существующей базы данных — это доступная только для чтения копия базы данных-источника, которая отражает состояние этой базы данных на момент копирования. (Таким образом, можно создавать множественные моментальные снимки существующей базы данных.) Файл моментального снимка (в примере 3 это файл C:\temp\snapshot_DB.mdf) содержит только измененные данные базы данных-источника. Поэтому в коде для создания моментального снимка необходимо указывать логическое имя каждого файла

данных базы данных- источника, а также соответствующие физические имена (см. пример 4).

Поскольку моментальный снимок содержит только измененные данные, то для каждого снимка требуется лишь небольшая доля дискового пространства, требуемого для соответствующей базы данных-источника.

ПРИМЕЧАНИЕ

Моментальные снимки баз данных можно создавать только на дисках с файловой системой *NTFS* (New Technology File System — файловая система новой технологии), т. к. только эта файловая система поддерживает технологию разреженных файлов, применяемую для хранения моментальных снимков.

Моментальные снимки баз данных обычно применяются в качестве механизма предохранения данных от искажения.

Присоединение и отсоединение баз данных

Все данные базы данных можно отсоединить, а потом снова присоединить к этому же или другому серверу базы данных. Эта функциональность используется при перемещении базы данных.

Для отсоединения базы данных от сервера баз используется системная процедура sp_detach_db. (Отсоединяемая база данных должна находиться в однопользовательском режиме.)

Для присоединения базы данных используется инструкция CREATE DATABASE с предложением FOR ATTACH. Для присоединяемой базы данных должны быть доступными все требуемые файлы. Если какой-либо файл данных имеет путь, отличающийся от исходного пути, то для этого файла необходимо указать текущий путь.

Инструкция *CREATE TABLE:* базовая форма

Инструкция CREATE TABLE создает новую таблицу базы данных со всеми соответствующими столбцами требуемого типа данных. Далее приводится базовая форма инструкции CREATE TABLE:

CREATE TABLE *table_name*
(col_name1 type1 [NOT NULL | NULL)
[{, *col_name2 type2* [NOT NULL | NULL]} ...])

ПРИМЕЧАНИЕ

Кроме основных таблиц, существуют также некоторые специальные виды таблиц, такие как временные таблицы и представления.

Параметр *table_name* — имя создаваемой базовой таблицы. Максимальное количество таблиц, которое может содержать одна база данных, ограничивается количеством объектов базы данных, число которых не может быть более 2 миллиардов, включая таблицы, представления, хранимые процедуры, триггеры и ограничения. В параметрах *col_name1, col_name2, ...* указываются имена столбцов таблицы, а в параметрах *type1, type2, ...* — типы данных соответствующих столбцов (см. *предыдущую работу*).

ПРИМЕЧАНИЕ

Имя объекта базы данных может обычно состоять из четырех частей, в форме:

[server_name.[db_name.[schema_name.]]]object_name

Здесь *object_name* — это имя объекта базы данных, *schema_name* — имя схемы, к которой принадлежит объект, а *server_name* и *db_name* — имена сервера и базы данных, к которым принадлежит объект. Имена таблиц, сгруппированные с именем схемы, должны быть однозначными в рамках базы данных. Подобным образом имена столбцов должны быть однозначными в рамках таблицы.

Рассмотрим теперь ограничение, связанное с присутствием или отсутствием значений NULL в столбце. Если для столбца не указано, что значения NULL разрешены (NOT NULL), то данный столбец не может содержать значения NULL, и при попытке вставить такое значение система возвратит сообщение об ошибке.

Как уже упоминалось, объект базы данных (в данном случае таблица) всегда создается в схеме базы данных. Пользователь может создавать таблицы только в такой схеме, для которой у него есть полномочия на выполнение инструкции alter. Любой пользователь с ролью sysadmin,

db_ddladmin или db_owner может создавать таблицы в любой схеме.

Создатель таблицы не обязательно должен быть ее владельцем. Это означает, что один пользователь может создавать таблицы, которые принадлежат другим пользователям. Подобным образом таблица, создаваемая с помощью инструкции CREATE TABLE, не обязательно должна принадлежать к текущей базе данных, если в префиксе имени таблицы указать другую (существующую) базу данных и имя схемы.

Схема, к которой принадлежит таблица, может иметь два возможных имени по умолчанию. Если таблица указывается без явного имени схемы, то система выполняет поиск имени таблицы в соответствующей схеме по умолчанию. Если имя объекта найти в схеме по умолчанию не удается, то система выполняет поиск в схеме dbo.

ПРИМЕЧАНИЕ

Имена таблиц всегда следует указывать вместе с именем соответствующей схемы. Это позволит избежать возможных неопределенностей.

Временные таблицы — это специальный тип базовой таблицы. Она сохраняются в базе данных tempdb и автоматически удаляются в конце сессии.

В примере 4 показано создание всех таблиц базы данных sample. (База данных sampleNew должна быть установлена в качестве текущей базы данных.)

Пример 4. Создание всех таблиц базы данных sampleNew

```
USE sampleNew;
CREATE TABLE employee (emp_no INTEGER NOT NULL,
                       empfname CHAR(20) NOT NULL,
                       emp_lname CHAR(20) NOT NULL,
                       dept_no CHAR(4) NULL);
CREATE TABLE department(dept_no CHAR(4) NOT NULL,
                       dept_name CHAR(25) NOT NULL,
                       location CHAR(30) NULL);
CREATE TABLE project (project_no CHAR(4) NOT NULL,
                       project_name CHAR(15) NOT NULL,
                       budget FLOAT NULL);
CREATE TABLE works_on (emp_no INTEGER NOT NULL,
                       project_no CHAR(4) NOT NULL,
                       job CHAR (15) NULL,
                       enter_date DATE NULL);
```

Кроме типа данных и свойства содержать значения NULL, в спецификации столбца можно указать следующие параметры:

- предложение DEFAULT;
- свойство IDENTITY.

Предложение DEFAULT в спецификации столбца указывает значение столбца по умолчанию, т. е. когда в таблицу вставляется новая строка, ячейка этого столбца будет содержать указанное значение, которое останется в ячейке, если в нее не будет введено другое значение. В качестве значения по умолчанию можно использовать константу, например, одну из системных функций, таких как, USER, CURRENT_USER, SESSION_USER, SYSTEM_USER, CURRENT_TIMESTAMP и NULL.

Столбец идентификаторов, создаваемый указанием свойства identity, может иметь только целочисленные значения, которые системой присваиваются обычно неявно. Каждое следующее значение, вставляемое в такой столбец, вычисляется, увеличивая последнее, вставленное в этот столбец, значение. Поэтому определение столбца со свойством identity содержит (явно или неявно) начальное значение и шаг инкремента.

В заключение этого раздела, в примере 5, показано создание таблицы, содержащей столбец типа SQL_VARIANT.

Пример 5 Создание таблицы, содержащей столбец типа sql_variant

```
USE project;
```

```
CREATE TABLE Item_Attributes (
        item_id INT NOT NULL,
        attribute NVARCHAR(30) NOT NULL,
        value SQL_VARIANT NOT NULL,
        PRIMARY KEY (item_id, attribute) )
```

В примере 5 создается таблица, содержащая столбец value, который имеет тип sql_variant. Как рассматривалось в *предыдущей работе*, тип данных sql_variant можно использовать для хранения значений разных типов одновременно, таких как числовые значения, строки и даты. Обратите внимание на то, что в примере 5 столбцу присваивается тип данных SQL_VARIANT по той причине, что значения разных атрибутов могут быть разных типов данных. Например, для атрибута размера тип данных значения attribute будет целочисленным, а для атрибута имени — строковым.

Инструкция *CREATE TABLE* и ограничения декларативной целостности

Одной из самых важных особенностей, которую должна предоставлять СУБД, является способ обеспечения целостности данных. Ограничения, которые используются для проверки данных при их модификации или вставке, называются *ограничениями для обеспечения целостности* (integrity constraints). Обеспечение целостности данных может осуществляться пользователем в прикладной программе или же системой управления базами данных. Наиболее важными преимуществами предоставления ограничений целостности системой управления базами данных являются следующие:

- повышается надежность данных;
- сокращается время на программирование;
- упрощается техническое обслуживание.

Определение ограничений для обеспечения целостности посредством СУБД повышает надежность данных, поскольку устраняется возможность, что программист прикладного приложения может забыть реализовать их. Если ограничения целостности предоставляются прикладными программами, то *все* приложения, затрагиваемые этими ограничениями, должны содержать соответствующий код. Если код отсутствует хоть в одном приложении, то целостность данных будет поставлена под сомнение.

Если ограничения для обеспечения целостности не предоставляются системой управления базами данных, то их необходимо определить в каждой программе приложения, которая использует данные, включенные в это ограничение. В противоположность этому, если ограничения для обеспечения целостности предоставляются системой управления базами данных, то их требуется определить только один раз. Кроме этого, код для ограничений, предоставляемых приложениями, обычно более сложный, чем в случае таких же ограничений, предоставляемых СУБД.

Если ограничения для обеспечения целостности предоставляются СУБД, то в случае изменений ограничений, соответствующие изменения в коде необходимо реализовать только один раз — в системе управления базами данных. А если ограничения предоставляются приложениями, то модификацию для отражения изменений в ограничениях необходимо выполнить в каждом из этих приложений.

Системами управления базами данных предоставляются два типа ограничений для обеспечения целостности:

- декларативные ограничения для обеспечения целостности;
- процедурные ограничения для обеспечения целостности, реализуемые посредством триггеров.

Декларативные ограничения определяются с помощью инструкций языка DDL CREATE TABLE и ALTER TABLE. Эти ограничения могут быть уровня столбцов или уровня таблицы. Ограничения уровня столбцов определяются наряду с типом данных и другими свойствами столбца в объявлении столбца, тогда как ограничения уровня таблицы всегда определяются в конце инструкции CREATE TABLE или ALTER TABLE после определения всех столбцов.

ПРИМЕЧАНИЕ

Между ограничениями уровня столбцов и ограничениями уровня таблицы есть лишь одно различие: ограничения уровня столбцов можно применять только к одному столбцу, в то время как ограничения уровня таблицы могут охватывать больше, чем один столбец таблицы.

Каждому декларативному ограничению присваивается имя. Это имя может быть присвоено явно посредством использования опции CONSTRAINT в инструкции CREATE TABLE ИЛИ ALTER TABLE. Если опция CONSTRAINT не указывается, то имя ограничению присваивается неявно компонентом Database Engine.

ПРИМЕЧАНИЕ

Настоятельно рекомендуется использовать явные имена ограничений, поскольку это может значительно улучшить поиск этих ограничений.

Декларативные ограничения можно сгруппировать в следующие категории:

- предложение DEFAULT;
- предложение UNIQUE;
- предложение PRIMARY KEY;
- предложение CHECK;
- ссылочная целостность и предложение FOREIGN KEY.

Использование предложения DEFAULT для определения ограничения по умолчанию было показано в этой работе ранее (см. также пример 6). Все другие ограничения рассматриваются далее.

Предложение *UNIQUE*

Иногда несколько столбцов или группа столбцов таблицы имеет уникальные значения, что позволяет использовать их в качестве первичного ключа. Столбцы или группы столбцов, которые можно использовать в качестве первичного ключа, называются *потенциальными ключами* (candidate key). Каждый потенциальный ключ определяется, используя предложение unique в инструкции create table или alter table. Синтаксис предложения unique следующий:

[CONSTRAINT *c_name]*

UNIQUE [CLUSTERED NONCLUSTERED1 *({colname1},...)*

Опция constraint в предложении unique присваивает явное имя потенциальному ключу. Опция clustered или nonclustered связана с тем обстоятельством, что компонент Database Engine создает индекс для каждого потенциального ключа таблицы. Этот индекс может быть кластеризованным, когда физический порядок строк определяется посредством индексированного порядка значений столбца. Если порядок строк не указывается, индекс является некластеризованным. По умолчанию применяется опция nonclustered. Параметр *colname1* обозначает имя столбца, который создает потенциальный ключ. (Потенциальный ключ может иметь до 16 столбцов.)

Применение предложения UNIQUE показано в примере 6. (Прежде чем выполнять этот пример, в базе данных sample нужно удалить таблицу projects, используя для этого инструкцию DROP TABLE projects.)

Пример 6. Применение предложения unique

```
USE sample;
CREATE TABLE projects (project_no CHAR(4) DEFAULT 'p1',
                       project_name CHAR(15) NOT NULL,
                       budget FLOAT NULL
                       CONSTRAINT unique_no UNIQUE (project_no));
```

Каждое значение столбца project_no таблицы projects является уникальным, включая значение NULL. (Точно так же, как и для любого другого значения с ограничением UNIQUE, если значения NULL разрешены для соответствующего столбца, этот столбец может содержать не более одной строки со значением NULL.) Попытка вставить в столбец project_no уже имеющееся в нем значение будет неуспешной, т. к. система не примет его. Явное имя ограничения, определяемого в примере 6, — unique_no.

Предложение *PRIMARY KEY*

Первичным ключом таблицы является столбец или группа столбцов, значения которого разные в каждой строке. Каждый первичный ключ определяется, используя предложение primary key в инструкции create table или alter table. Синтаксис предложения primary key следующий:

[CONSTRAINT *c_name]*

PRIMARY KEY [CLUSTERED | NONCLUSTERED] ({*col_name1*},...)

Все параметры предложения PRIMARY KEY имеют такие же значения, как и соответствующие одноименные параметры предложения UNIQUE. Но в отличие от столбца UNIQUE, столбец PRIMARY KEY не разрешает значений NULL и имеет значение по умолчанию CLUSTERED.

В примере 7 показано объявление первичного ключа для таблицы employee базы данных sample.

ПРИМЕЧАНИЕ

Прежде чем выполнять этот пример, в базе данных sampleNew нужно удалить таблицу employee, используя для этого инструкцию DROP TABLE employee.

Пример 7. Определение первичного ключа

```
USE sample;
CREATE TABLE employee (emp_no INTEGER NOT NULL,
                       emp_fname CHAR(20) NOT NULL, emp_lname
                       CHAR(20) NOT NULL, dept_no CHAR(4) NULL,
                       CONSTRAINT prim_empl PRIMARY KEY (emp_no));
```

В результате выполнения кода в примере 7 снова создается таблица employee, в которой определен первичный ключ. Первичный ключ таблицы определяется посредством декларативного ограничения для обеспечения целостности с именем prim_empl. Это ограничение для обеспечения целостности является ограничением уровня таблицы, поскольку оно указывается после определения всех столбцов таблицы employee.

Пример 8 эквивалентный примеру 7, за исключением того, что первичный ключ таблицы employee определяется как ограничение уровня столбца.

ПРИМЕЧАНИЕ

Опять же, прежде чем выполнять этот пример, в базе данных sampleNew нужно удалить таблицу employee, используя для этого инструкцию DROP TABLE employee.

Пример 8. Определение ограничения уровня столбца

```
USE sampleNew;
CREATE TABLE employee
    (emp_no INTEGER NOT NULL CONSTRAINT prim_empl PRIMARY KEY,
    emp_fname CHAR(20) NOT NULL,
    emp_lname CHAR(20) NOT NULL,
    dept_no CHAR(4) NULL);
```

В примере 8 предложение primary key принадлежит к объявлению соответствующего столбца, наряду с объявлением его типа данных и свойства содержать значения null. По этой причине это ограничение называется *ограничением на уровне столбца.*

Предложение *CHECK*

Проверочное ограничение (check constraint) определяет условия для вставляемых в столбец данных. Каждая вставляемая в таблицу строка или каждое значение, которым обновляется значение столбца, должно отвечать этим условиям. Проверочные ограничения устанавливаются посредством предложения CHECK, определяемого в инструкции CREATE TABLE или ALTER TABLE. Синтаксис предложения CHECK следующий:

```
[CONSTRAINT c_name]
    CHECK [NOT FOR REPLICATION] expression
```

Параметр *expression* должен иметь логическое значение (true или false) и может ссылаться на любые столбцы в текущей таблице (или только на текущий столбец, если определен как ограничение уровня столбца), но не на другие таблицы. Предложение check не применяется принудительно при репликации данных, если присутствует параметр not for replication. (При репликации база данных, или ее часть, хранится в нескольких местах. С помощью репликации можно повысить уровень доступности данных.)

В примере 9 показано применение предложения CHECK.

Пример 9. Применение предложения check

```
USE sampleNew;
CREATE TABLE customer
    (cust_no INTEGER NOT NULL,
    cust_group CHAR(3) NULL,
    CONSTRAINT ch_cust_gr CHECK (cust_group IN ('c1', 'c2', 'c10')));
```

Создаваемая в примере 9 таблица customer включает столбец cust_group, содержащий соответствующее проверочное ограничение. При вставке нового значения, отличающегося от значений в наборе ('cl', 'c2', 'c10'), или при попытке изменения существующего значения на значение, отличающегося от этих значений, система управления базой данных возвращает сообщение об ошибке.

Предложение *FOREIGN KEY*

Внешним ключом (foreign key) называется столбец (или группа столбцов таблицы), содержащий значения, совпадающие со значениями первичного ключа в этой же или другой таблице. Внешний ключ определяется с помощью предложения foreign key в комбинации с предложением references. Синтаксис предложения foreign key следующий:

```
[CONSTRAINT c_name]
    [[FOREIGN KEY] ({col_name1},.)]
    REFERENCES table_name ({col_name2},.)
    [ON DELETE (NO ACTION) | CASCADE | SET NULL | SET DEFAULT}]
    [ON UPDATE (NO ACTION) | CASCADE | SET NULL | SET DEFAULT}]
```

Предложение FOREIGN KEY явно определяет все столбцы, входящие во внешний ключ. В предложении REFERENCES указывается имя таблицы, содержащей столбцы, создающие соответствующий первичный ключ. Количество столбцов и их тип дан ных в предложении FOREIGN KEY должны совпадать с количеством соответствующих столбцов и их типом данных в предложении REFERENCES (и, конечно же, они должны совпадать с количеством столбцов и типами данных в первичном ключе таблицы, на которую они ссылаются).

Таблица, содержащая внешний ключ, называется *ссылающейся (или дочерней) таблицей* (referencing table), а таблица, содержащая соответствующий первичный ключ, называется *ссылочной* (referenced table) или *родительской* (parent table) таблицей. В примере 10 показано объявление внешнего ключа для таблицы works_on базы данных sampleNew.

ПРИМЕЧАНИЕ

Прежде чем выполнять этот пример, в базе данных sampleNew нужно удалить таблицу works_on, используя для этого инструкцию DROP TABLE works_on.

Пример 10. Объявление внешнего ключа

```
USE sampleNew;
CREATE TABLE works_on (emp_no INTEGER NOT NULL,
                       project_no CHAR(4) NOT NULL,
                       job CHAR (15) NULL,
                       enter_date DATE NULL,
    CONSTRAINT prim_works PRIMARY KEY(emp_no, project_no),
    CONSTRAINT foreign_works FOREIGN KEY(emp_no) REFERENCES employee (emp_no));
```

Таблица works_on в примере 10 задается с двумя декларативными ограничениями для обеспечения целостности: prim_works и foreign_works. Оба ограничения являются уровня таблицы, где первое указывает первичный ключ, а второе — внешний ключ таблицы works_on. Кроме этого, ограничение foreign_works определяет таблицу employee, как ссылочную таблицу, а ее столбец emp_no, как соответствующий первичный ключ столбца с таким же именем в таблице works_on.

Предложение FOREIGN KEY можно пропустить, если внешний ключ определяется, как

ограничение уровня таблицы, поскольку столбец, к которому применяется ограничение, является неявным "списком" столбцов внешнего ключа, и ключевого слова REFERENCES достаточно для указания того, какого типа является это ограничение. Таблица может содержать самое большее 63 ограничения FOREIGN KEY.

Определение внешних ключей в таблицах базы данных налагает определение другого важного ограничения для обеспечения целостности: ссылочной целостности, которая рассматривается следующей.

Ссылочная целостность

Ссылочная целостность (referential integrity) обеспечивает выполнение правил для вставок и обновлений таблиц, содержащих внешний ключ и соответствующее ограничение первичного ключа. В примерах 7 и 10 задаются два таких ограничения: prim_empl и foreign_works. Предложение REFERENCES в примере 10 определяет таблицу employee в качестве ссылочной (родительской) таблицы.

Если для двух таблиц указана ссылочная целостность, модифицирование значений в первичном ключе и соответствующем внешнем ключе будет не всегда возможным. В последующих разделах рассматривается, когда это возможно, а когда нет.

Возможные проблемы со ссылочной целостностью

Модификация значений внешнего или первичного ключа может создавать проблемы в четырех случаях. Все эти случаи будут продемонстрированы с использованием базы данных sampleNew. В первых двух случаях затрагиваются модификации ссылающейся таблицы, а в последних двух — родительской.

Случай 1

Вставка новой строки в таблицу works_on с номером сотрудника 11111.
Соответствующая инструкция Transact-SQL выглядит таким образом:

```
USE sampleNew;
INSERT INTO works_on (emp_no,...)
    VALUES (11111,...);
```

При вставке новой строки в дочернюю таблицу works_on используется новый номер сотрудника emp_no, для которого нет совпадающего сотрудника (и номера) в родительской таблице employee. Если для обеих таблиц определена ссылочная целостность, как это сделано в примерах 7 и 10, то компонент Database Engine не допустит вставки новой строки с таким номером emp_no.

Случай 2

Изменение номера сотрудника 10102 во всех строка таблицы works_on на номер 11111.
Соответствующая инструкция Transact-SQL выглядит таким образом:

```
USE sampleNew;
UPDATE works_on
    SET emp_no = 11111 WHERE emp_no = 10102;
```

В данном случае существующее значение внешнего ключа в ссылающейся таблице works_on заменяется новым значением, для которого нет совпадающего значения в родительской таблице employee. Если для обеих таблиц определена ссылочная целостность, как это сделано в примерах 7 и 10, то система управления базой данных не допустит модификацию строки с таким номером emp_no в таблице works_on.

Случай 3

Замена значения `10102` номера сотрудника emp_no на значение `22222` в таблице employee. Соответствующая инструкция Transact-SQL будет выглядеть таким образом:

```
USE sampleNew;
UPDATE employee
    SET emp_no = 22222 WHERE emp_no = 10102;
```

В данном случае предпринимается попытка заменить существующее значение 10102 номера сотрудника emp_no значением 22222 только в родительской таблице employee, не меняя соответствующие значения emp_no в ссылающейся таблице works_on. Система не разрешает выполнения этой операции. Ссылочная целостность не допускает существования в ссылающейся таблице (таблице, для которой предложением foreign key определен внешний ключ) таких значений, для которых в родительской таблице (таблице, для которой предложением primary key определен первичный ключ) не существует соответствующего значения. В противном случае такие строки в ссылающейся таблице были бы "сиротами". Если бы описанная выше модификация таблицы employee была разрешена, тогда строки в таблице works_on со значением emp_no равным 10102 были бы сиротами. Поэтому система и не разрешает выполнения такой модификации.

Случай 4

Удаление строки в таблице employee со значением emp_no равным 10102.

Этот случай похожий на *случай 3*. В случае выполнения этой операции, из таблицы employee была бы удалена строка со значением emp_no, для которого существуют совпадающие значения в ссылающейся (дочерней) таблице works_on. В примере 11 приводится код для определения всех ограничений первичного и внешнего ключей для таблиц базы данных sampleNew. Для выполнения этого кода, если таблицы employee, department, project и works_on уже существуют, то их нужно удалить с помощью инструкции drop table *table_name (имятаблицы*).

Пример 11. Определение всех ограничений первичного и внешнего ключей для таблиц базы данных `sampleNew`

```
USE sampleNew;
CREATE TABLE department(dept_no CHAR(4) NOT NULL,
                        dept_name CHAR(25) NOT NULL,
                        location CHAR(30) NULL,
                CONSTRAINT prim_dept PRIMARY KEY (dept_no));
CREATE TABLE employee (emp_no INTEGER NOT NULL,
                        emp_fname CHAR(20) NOT NULL,
                        emp_lname CHAR(20) NOT NULL,
                        dept_no CHAR(4) NULL,
                CONSTRAINT prim_emp PRIMARY KEY (emp_no),
                CONSTRAINT foreign_emp FOREIGN KEY(dept_no) REFERENCES department(dept_no));
CREATE TABLE project (project_no CHAR(4) NOT NULL,
                        project_name CHAR(15) NOT NULL,
                        budget FLOAT NULL,
                CONSTRAINT prim_proj PRIMARY KEY (project_no));
CREATE TABLE works_on (emp_no INTEGER NOT NULL,
                        project_no CHAR(4) NOT NULL,
                        job CHAR (15) NULL,
                        enter_date DATE NULL,
                CONSTRAINT prim_works PRIMARY KEY(emp_no, project_no),
                CONSTRAINT foreign1_works FOREIGN KEY(emp_no) REFERENCES employee(emp no),
                CONSTRAINT foreign2_works FOREIGN KEY(project_no) REFERENCES project(project_no));
```

Опции *ON DELETE* и *ON UPDATE*

Компонент Database Engine на попытку удаления и модифицирования первичного ключа может реагировать по-разному. Если попытаться обновить значения внешнего ключа, то все эти обновления будут несогласованы с соответствующим первичным ключом (см. *случай 1* и *2* в предыдущем разделе), база данных откажется выполнять эти обновления и выведет сообщение об ошибке, наподобие следующего:

Server: Msg 547, Level 16, State 1, Line 1 UPDATE statement conflicted with COLUMN FOREIGN KEY constraint 'FKemployee'. The conflict occurred in database 'sampleNew', table 'employee', column 'dept no'. The statement has been terminated.

(Сообщение 547, уровень 16, состояние 0, строка 1. Конфликт инструкции UPDATE с ограничением FOREIGN KEY "foreign1_works". Конфликт произошел в базе данных "sampleNew", таблица "dbo.employee", столбец 'emp_no'. Выполнение данной инструкции было прервано.)

Но при попытке внести обновления в значения первичного ключа, вызывающие несогласованность в соответствующем внешнем ключе (см. *случай 3* и *4* в предыдущем разделе), система базы данных может реагировать достаточно гибко. В целом, существует четыре опции, определяющих то, как система базы данных может реагировать.

- NO ACTION. Модифицируются (обновляются или удаляются) только те значения в родительской таблице, для которых нет соответствующих значений во внешнем ключе дочерней (ссылающейся) таблицы.
- CASCADE. Разрешается модификация (обновление или удаление) любых значений в родительской таблице. При обновлении значения первичного ключа в родительской таблице или при удалении всей строки, содержащей данное значение, в дочерней (ссылающейся) таблице обновляются (т. е. удаляются) все строки с соответствующими значениями внешнего ключа.
- SET NULL. Разрешается модификация (обновление или удаление) любых значений в родительской таблице. Если обновление значения в родительской таблице вызывает несогласованность в дочерней таблице, система базы данных присваивает внешнему ключу всех соответствующих строк в дочерней таблице значение NULL. То же самое происходит и в случае удаления строки в родительской таблице, вызывающего несогласованность в дочерней таблице. Таким образом, все несогласованности данных пропускаются.
- SET DEFAULT. Аналогично опции SET NULL, но с одним исключением: всем внешним ключам, соответствующим модифицируемому первичному ключу, присваивается значение по умолчанию. Само собой разумеется, что после модификации первичный ключ родительской таблицы все равно должен содержать значение по умолчанию.

ПРИМЕЧАНИЕ

В языке Transact-SQL поддерживаются первые две из этих опций. Использование опций ON DELETE и ON UPDATE показано в примере 12.

Пример 12. Применение опций on delete и on update

```
USE sampleNew;
CREATE TABLE works_on1
    (emp_no INTEGER NOT NULL,
    project_no CHAR(4) NOT NULL,
    job CHAR (15) NULL,
    enter_date DATE NULL,
    CONSTRAINT prim_works1 PRIMARY KEY(emp_no, project_no),
    CONSTRAINT foreign1_works1 FOREIGN KEY(emp_no)
            REFERENCES employee(emp_no) ON DELETE CASCADE,
    CONSTRAINT foreign2_works1 FOREIGN KEY(project_no)
            REFERENCES proiact(project_no) ON UPDATE CASCADE);
```

В примере 12 создается таблица works_on1 с использованием опций ON DELETE CASCADE и ON UPDATE CASCADE. Если таблицу works_on1 загрузить значениями из табл. 1.4, каждое удаление строки в таблице employee будет вызывать каскадное удаление всех строк в таблице works_on1, которые имеют значения внешнего ключа, соответствующие значениям первичного ключа строк, удаляемых в таблице employee. Подобным образом каждое обновление значения столбца project_no таблицы project будет вызывать такое же обновление всех соответствующих значений столбца project_no таблицы works_on1.

Создание других объектов баз данных

Кроме основных таблиц, которые существуют как самостоятельные сущности, реляционная база данных также содержит *представления* (view), которые являются виртуальными таблицами. Данные базовой таблицы существуют физически, т. е. сохранены на диске, тогда как представление извлекается из одной или нескольких базовых таблиц. Представление на основе одной или нескольких существующих таблиц баз данных (или представлений) создается с помощью

инструкции create view и инструкции select, которая является неотъемлемой частью инструкции create view. Поскольку создание представления всегда содержит запрос, инструкция create view принадлежит к языку манипуляции данными (DML), а не к языку описания данных (DDL).

Инструкция create index создает новый *индекс* для указанной таблицы. Индексы в основном применяются для обеспечения эффективного доступа к данным, хранящимся на диске. Наличие индекса может значительно улучшить доступ к данным.

Еще одним дополнительным объектом базы являются *хранимые процедуры* (stored procedure), которые создаются посредством инструкции create procedure. *Хранимая процедура* — это последовательность инструкций Transact-SQL, созданная посредством языка SQL и процедурных расширений.

Триггером (trigger) называется объект базы данных, который задает определенное действие в ответ на определенное событие. Это означает, что когда для предопределенной таблицы происходит определенное событие (модификации, вставка или удаление данных), компонент Database Engine автоматически запускает одно или несколько дополнительных действий.

Синоним (synonym) — это локальный объект базы данных, который предоставляет связь между самим собой и другим объектом, управляемым одним и тем же или связанным сервером баз данных. Синонимы объектов создаются посредством инструкции create synonym, применение которой показано в примере 13.

Пример 13. Создание синонима объекта базы данных

```
USE ABC;
CREATE SYNONYM prod
                FOR ABC.Production.Product;
```

В примере 13 создается синоним таблицы Product в схеме Production базы данных ABC. Этот синоним можно потом использовать в инструкциях языка DML, таких как SELECT, INSERT, UPDATE и DELETE.

ПРИМЕЧАНИЕ

Синонимы в основном используются во избежание необходимости применять длинные имена в инструкциях DML. Как уже упоминалось, имя объекта базы данных может состоять из четырех частей. Использование синонима, состоящего из одной части, для объекта с именем, состоящим из трех или четырех частей, позволяет сэкономить время на вводе имени такого объекта.

Схема (schema) — это объект базы данных, содержащий инструкции для создания таблиц, представлений и пользовательских разрешений. Схему можно рассматривать, как конструкцию, в которой собраны вместе несколько таблиц, соответствующие представления и пользовательские разрешения.

ПРИМЕЧАНИЕ

В Database Engine применяется такое же понятие схемы, как и в стандарте ANSI SQL. В стандарте SQL схема определяется как коллекция объектов базы данных, имеющая одного владельца и формирующая одно пространство имен. *Пространство имен* (namespace) — это набор объектов с однозначными именами. Например, две таблицы могут иметь одно и то же имя только в том случае, если они находятся в разных схемах. Схема является очень важным концептом в модели безопасности компонента Database Engine.

Ограничения для обеспечения целостности и домены

Домен (domain) — это набор всех возможных разрешенных значений, которые могут содержать столбцы таблицы. Почти во всех системах управления базами данных для определения таких возможных значений столбца используются такие типы данных, как int, char и date. Такого метода принудительного обеспечения "целостности домена" недостаточно, как можно увидеть в следующем примере.

В таблице person есть столбец zip, в котором указывается индекс города, в котором проживает данное лицо. Тип данных этого столбца можно определить как SMALLINT или CHAR (5). Определение типа данных столбца как SMALLINT будет неточным, потому что этот тип данных содержит все положительные и отрицательные целые числа в диапазоне от -2^{15} до 2^{15}. Объявление с использованием типа данных CHAR (5) будет еще менее точным, поскольку в таком случае можно

будет использовать все буквенно-цифровые и специальные символы. Поэтому для точного определения данных столбца индексов требуется диапазон положительных значений от 00601 до 99950.

Более точно целостность домена можно принудительно обеспечить с помощью ограничений CHECK (определяемые в инструкции CREATE TABLE или ALTER TABLE), благодаря их гибкости и тому, что они всегда принудительно применяются при вставке или модифицировании данных столбца.

Язык Transact-SQL поддерживает домены посредством создания псевдонимов типов данных с помощью инструкции CREATE TYPE. Рассмотрению псевдонимов типов данных и типов данных среды CLR посвящены следующие два раздела.

Псевдонимы типов данных

Псевдоним типа данных (alias data type) — это специальный, который определяется пользователем при использовании существующих базовых типов данных. Такой тип данных можно использовать в инструкции create table для определения одного или большего количества столбцов таблицы.

Для создания псевдонимного типа данных обычно применяется инструкция CREATE TYPE. Далее показан синтаксис этой инструкции для определения псевдо- нимного типа данных:

```
CREATE TYPE [type_schema_name.] type_name
{[FROM base_type[(precision[ , scale ])] [NULL | NOT NULL]]
    | [EXTERNAL NAME assembly_name [.class_name]]}
```

Использование инструкции CREATE TYPE для создания псевдонимного типа данных показано в примере 14.

Пример 14. Создание псевдонимного типа данных С ПОМОЩЬЮ инструкции CREATE TYPE

```
USE sampleNew;
CREATE TYPE zip
    FROM SMALLINT NOT NULL;
```

В примере 14 создается псевдонимный тип данных zip на основе стандартного типа данных SMALLINT. Теперь этот определенный пользователем тип данных можно присвоить столбцу таблицы, как показано в примере 15.

ПРИМЕЧАНИЕ

Прежде чем выполнять этот пример, в базе данных AdventureWorks нужно удалить таблицу customer, используя для этого инструкцию DROP TABLE customer.

Пример 15. Определение столбца псевдонимным типом данных

```
USE sampleNew;
CREATE TABLE customer
    (cust_no INT NOT NULL,
    cust_name CHAR(20) NOT NULL,
    city CHAR(20),
    zip_code ZIP,
CHECK (zip_code BETWEEN 601 AND 99950));
```

В примере 15 тип данных столбца zip_code таблицы customer определяется псевдонимным типом данных zip. Допустимые значения этого столбца требуется ограничить диапазоном целочисленных значений от 601 до 99950. Как можно видеть в примере 15, это ограничение можно наложить с помощью предложения CHECK.

ПРИМЕЧАНИЕ

Обычно компонент Database Engine неявно преобразовывает разные типы данных совместимых столбцов. Это также относится и к псевдонимным типам данных.

Начиная с версии SQL Server 2008, стали поддерживаться определяемые пользователем табличные типы. В примере 16 показано создание такого типа с помощью инструкции CREATE TYPE.

Пример 16. Создание табличного типа

```
USE sampleNew;
CREATE TYPE person_table_t AS TABLE
    (name VARCHAR(30),
    salary DECIMAL(8,2));
```

Создаваемый в примере 16 определяемый пользователем табличный тип данных person_table_t имеет два столбца: name и salary. Основное синтаксическое отличие табличных типов от псевдонимных состоит в наличии предложения as table, как это можно видеть в примере 15. Определяемые пользователем табличные типы обычно применяются с возвращающими табличные значения параметрами.

Типы данных CLR

Инструкцию CREATE TYPE можно также применить для создания определяемых пользователем типов данных с использованием .NET. В этом случае, реализация определяемого пользователем типа определяется в классе сборки в среде CLR. Это означает, что для реализации нового типа данных можно использовать один из языков .NET, такой как C# или Visual Basic.

Модифицирование объектов баз данных

Язык Transact-SQL поддерживает модифицирование структуры следующих объектов базы данных, среди прочих:

- базы данных;
- таблицы;
- хранимые процедуры;
- представления;
- схемы;
- триггеры.

В последующих двух разделах описывается изменение первых двух объектов из этого списка: баз данных и таблиц.

Все, что выделено красным – делать не надо. Надо только ознакомиться.

Изменение базы данных

Для изменения физической структуры базы данных используется инструкция ALTER DATABASE. Язык Transact-SQL позволяет выполнять следующие действия по изменению свойств базы данных:

- добавлять и удалять один или несколько файлов базы данных;
- добавлять и удалять один или несколько файлов журнала;
- добавлять и удалять файловые группы;
- изменять свойства файлов или файловых групп;
- устанавливать параметры базы данных;
- изменять имя базы данных с помощью хранимой процедуры sp_rename (рассматривается в *пункте "Изменение таблиц "* далее в этой рабоет).

Эти разные типы модификаций базы данных рассматриваются в последующих далее подразделах. В этом разделе мы также используем инструкцию ALTER DATABASE, чтобы показать, как можно сохранять данные типа FILESTREAM в файлах и файловых группах, а также для объяснения концепции автономной базы данных.

Добавление и удаление файлов базы данных, файлов журналов и файловых групп

Добавление или удаление файлов базы данных осуществляется посредством инструкции ALTER DATABASE. Операция добавления нового или удаления существующего файла указывается предложением ADD FILE и REMOVE FILE соответственно. Кроме этого, новый файл можно определить в существующую файловую группу посредством параметра TO FILEGROUP.
В примере 17 показано добавление нового файла базы данных в базу данных projects.

Пример 17. Добавление нового файла в базу данных

```
USE master;
GO
```

```
ALTER DATABASE projects ADD FILE
(NAME=projects_dat1,
    FILENAME = 'C:\projects1.mdf',
    SIZE = 10,
    MAXSIZE = 100, FILEGROWTH = 5);
```

В примере 17 инструкция ALTER DATABASE добавляет новый файл с логическим именем projects_dat1. Здесь же указан начальный размер файла 10 Мбайт и автоувеличение по 5 Мбайт до максимального размера 100 Мбайт. Файлы журналов добавляются так же, как и файлы баз данных. Единственным отличием является то, что вместо предложения ADD FILE используется предложение ADD LOG FILE.
Удаления файлов (как файлов базы данных, так и файлов журнала) из базы данных осуществляется посредством предложения REMOVE FILE. Удаляемый файл должен быть пустым.
Новая файловая группа создается посредством предложения CREATE FILEGROUP, а существующая удаляется с помощью предложения DELETE FILEGROUP. Как и удаляемый файл, удаляемая файловая группа также должна быть пустой.

Изменение свойств файлов и файловых групп

С помощью предложения MODIFY FILE можно выполнять следующие действия по изменению свойств файла:

- изменять логическое имя файла, используя параметр NEWNAME;
- увеличивать значение свойства SIZE;
- изменять значение свойств FILENAME, MAXSIZSE и FILEGROWTH;
- отмечать файл как OFFLINE.

Подобным образом с помощью предложения MODIFY FILEGROUP можно выполнять следующие действия по изменению свойств файловой группы:

- изменять логическое имя файловой группы, используя параметр NAME;
- помечать файловую группу, как файловую группу по умолчанию, используя для этого параметр DEFEAULT;
- помечать файловую группу как позволяющую осуществлять доступ только для чтения или для чтения и записи, используя для этого параметр READ_ONLY или READ_WRITE соответственно.

Установка опций базы данных

Для установки различных опций базы данных используется предложение SET инструкции ALTER DATABASE. Некоторым опциям можно присвоить только значения ON или OFF, но для большинства из них предоставляется выбор из списка возможных значений. Каждый параметр базы данных имеет значение по умолчанию, которое устанавливается в базе данных model. Поэтому значения определенных опций по умолчанию можно модифицировать, изменив соответствующим образом базу данных model.
Все опции, значения которых можно изменять, можно разбить на несколько групп, наиболее важными из которых являются следующие:

- опции состояния;
- опции автоматических действий;
- опции SQL.

Опции состояния управляют следующими возможностями:

- доступом пользователей к базе данным (это опции SINGLE_USER, RESTRICTED_USER и MULTI_USER);
- статусом базы данных (это опции ONLINE, OFFLINE и EMERGENCY);
- режимом чтения и записи (опции READ_ONLY и READ_WRITE).

Опции автоматических операций управляют, среди прочего, остановом базы данных (опция AUTO_CLOSE) и способом создания статистики индексов (опции AUTO_CREATE_STATISTICS и AUTO_UPDATE_STATISTICS).
Опции SQL управляют соответствием базы данных и ее объектов стандарту ANSI. Значения всех операторов SQL можно узнать посредством функции DATABASEPROPERTYEX, а редактировать с помощью инструкции ALTER DATABASE.
Опции восстановления FULL, BULK-LUGGED и SIMPLE управляют процессом восстановления базы данных.

Хранение данных типа *FILESTREAM*

В предыдущей работе мы рассмотрели данные типа FILESTREAM и причины, по которым их используют. В этом разделе мы рассмотрим, как данные типа FILESTREAM можно сохранять в базе данных. Чтобы данные FILESTREAM можно было сохранять в базе данных, система должна быть должным образом инициирована. В следующем подразделе объясняется, как инициировать операционную систему и экземпляр базы данных для хранения данных типа FILESTREAM.

Инициирование хранилища *FILESTREAM*

Хранилище данных типа FILESTREAM требуется инициировать на двух уровнях:

- для операционной системы Windows;
- для конкретного экземпляра сервера базы данных.

Инициирование хранилища данных типа filestream на уровне системы осуществляется с помощью диспетчера конфигурации SQL Server. Чтобы запустить диспетчер конфигурации, выполните следующую последовательность команд по умолчанию **Пуск | Все программы | Microsoft SQL Server 2012 | Configuration Tools** (Средства настройки | Диспетчер конфигурации SQL Server Configuration Manager). В открывшемся окне **Sql Server Configuration Manager** (Sql Server Configuration Manger) щелкните правой кнопкой пункт **SQL Server Services** (Службы SQL Server) и в появившемся контекстном меню выберите команду **Open** (Открыть). В правой панели щелкните правой кнопкой экземпляр, для которого требуется разрешить хранилище filestream, и в контекстном меню выберите команду **Properties** (Свойства). В открывшемся диалоговом окне **SQL Server Properties** (Свойства | SQL Server) выберите вкладку **FILESTREAM** (FILESTREAM) (рис. 1).

Чтобы иметь возможность только читать данные типа filestream, установите флажок **Enable FILESTREAM for Transact-SQL access** (Разрешить FILESTREAM при доступе через Transact-SQL). Чтобы кроме чтения можно было также записывать данные, установите дополнительно флажок **Enable FILESTREAM for file I/O streaming access** (Разрешить использование FILESTREAM при доступе файлового ввода/вывода). Введите имя общей папки Windows в одноименное поле. Общая папка Windows используется для чтения и записи данных filestream, используя интерфейс API Win32. Если для возвращения пути для filestream blob использовать имя, то это будет имя общей папки Windows.

Диспетчер конфигурации SQL Server создаст на системе хоста новую общую папку с указанным именем. Чтобы применить изменения, нажмите кнопку **ОК.**

ПРИМЕЧАНИЕ

Чтобы разрешить хранилище FILESTREAM, необходимо быть администратором Windows локальной системы и обладать правами администратора (syssdmin). Чтобы изменения вступили в силу, необходимо перезапустить экземпляр сервера базы данных.

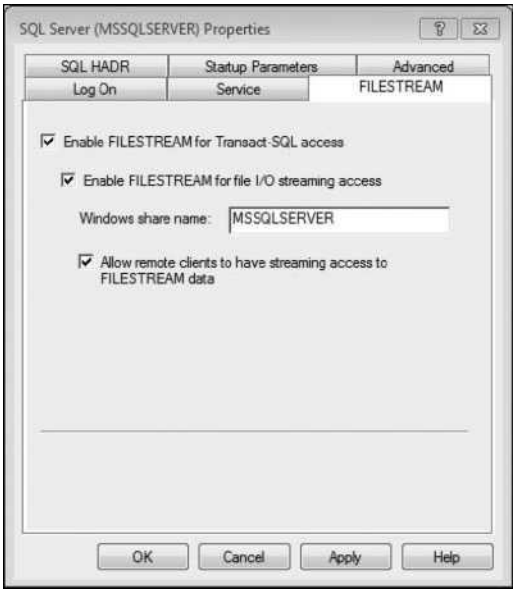

Рис. 1. Диалоговое окно SQL Server Properties, вкладка FILESTREAM

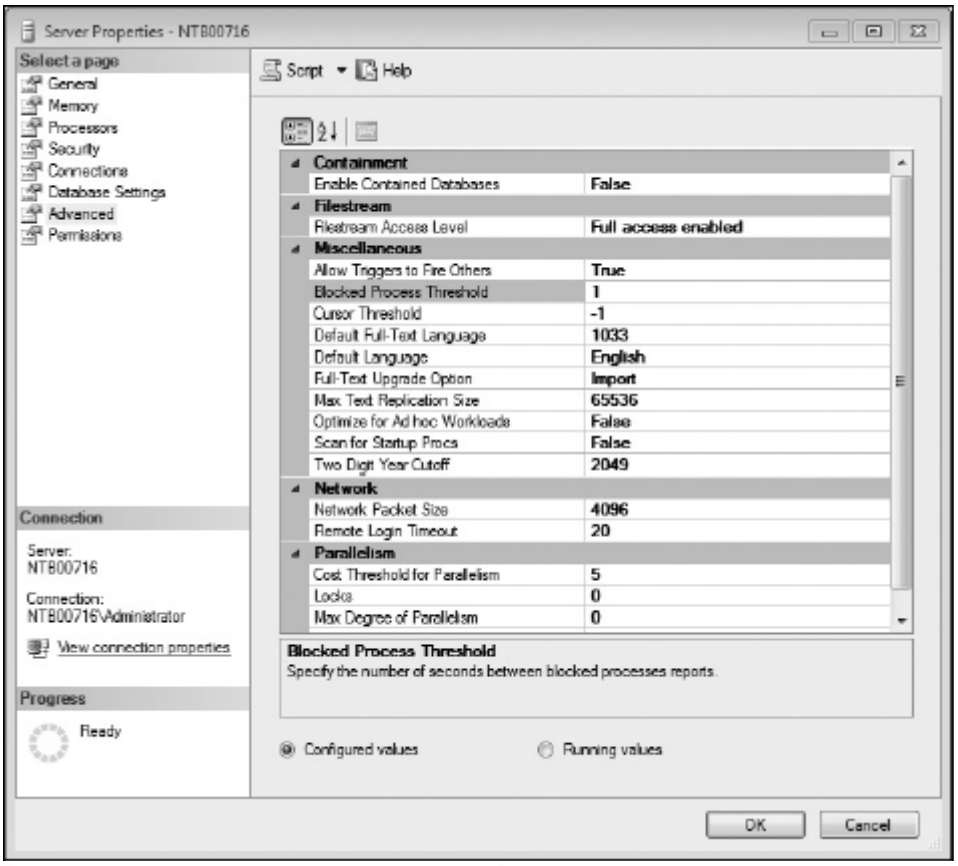

Рис. 2. Диалоговое окно Server Properties с уровнем доступа FILESTREAM, установленным в Full Access Enabled

Следующим шагом будет разрешить хранилище filestream для конкретного экземпляра. Мы рассмотрим, как выполнить эту задачу с помощью среды SQL Server Management Studio. (Для этого можно также воспользоваться хранимой системной процедурой sp_configure с параметром filestream access level.) Щелкните правой кнопкой требуемый экземпляр в обозревателе объектов и в появившемся контекстном меню выберите пункт **Properties** (Свойства), в левой панели открывшегося диалогового окна **Server Properties** (Свойства сервера) выберите пункт **Advanced** (Дополнительно) (рис. 2), после чего в правой панели из выпадающего списка выберите **Filestream Access Level** (Уровень доступа FILESTREAM) одну из следующих опций:

- **Disabled** (Отключено) — хранилище filestream не разрешено;
- **Transact-SQL Access Enabled** (Включен доступ с помощью Transact-SQL) — к данным filestream можно обращаться посредством инструкций T-SQL;
- **Full Access Enabled** (Включен полный доступ). К данным filestream можно обращаться как посредством инструкций T-SQL, так и через интерфейс API Win32.

Добавление файла в файловую группу

Разрешив хранилище FILESTREAM для требуемого экземпляра, можно сначала создать файловую группу для данных FILESTREAM (посредством инструкции ALTER DATABASE), а затем добавить файл в эту файловую группу, как это показано в примере 18. (Конечно же, эту задачу также можно было бы выполнить с помощью инструкции CREATE DATABASE.)

ПРИМЕЧАНИЕ

Прежде чем выполнять инструкции, приведенные в примере 18, измените имя файла в предложении FILENAME.

Пример 18. Добавление файла в файловую группу

```
USE sampleNew;
ALTER DATABASE sampleNew
    ADD FILEGROUP Employee_FSGroup CONTAINS FILESTREAM;
GO
ALTER DATABASE sampleNew
```

```
ADD FILE (NAME= employee_FS,
FILENAME = 'C:\DUSAN\emp_FS')
TO FILEGROUP Employee_FSGroup
```

Первая инструкция ALTER DATABASE в примере 18 добавляет в базу данных sampleNew новую файловую группу Employee_FSGroup. Параметр CONTAINS FILESTREAM этой инструкции указывает системе, что данная файловая группа будет содержать только данные FILESTREAM. Вторая инструкция ALTER DATABASE добавляет в созданную файловую группу новый файл.

Теперь можно создавать таблицы, содержащие столбцы с типом данных FILESTREAM. Создание такой таблицы показано в примере 19.

Пример 19. Создание таблицы, содержащей столбец filestream :

```
CREATE TABLE employee_info
    (id UNIQUEIDENTIFIER ROWGUIDCOL NOT NULL UNIQUE,
    filestream_data VARBINARY(MAX) FILESTREAM NULL)
```

В примере 19 таблица employee_info содержит столбец filestream_data, тип данных которого должен быть VARBINARY(max). Определение такого столбца включает атрибут FILESTREAM, указывающий, что данные столбца сохраняются в файловой группе FILESTREAM. Для всех таблиц, в которых хранятся данные типа FILESTREAM, требуется наличие свойств UNIQUE ROWGUILDCOL. Поэтому таблица employee_info содержит столбец id, определенный с использованием этих двух атрибутов.

Автономные базы данных

Одна из значительных проблем с базами данных SQL Server состоит в том, что они трудно поддаются экспортированию и импортированию. Как рассматривалось ранее, базы данных можно присоединять и отсоединять, но при этом утрачиваются важные части и свойства присоединенных баз данных. (Основной проблемой в таких случаях является безопасность базы данных, в общем, и учетные записи, в частности, в которых после перемещения обычно отсутствует часть информации или содержится неправильная информация.)

Разработчики Microsoft планируют решить эти проблемы посредством использования *автономных баз данных* (contained databases). Автономная база данных содержит все параметры и данные, необходимые для определения базы данных, и изолирована от экземпляра Database Engine, на котором она установлена. Иными словами, база данных данного типа не имеет конфигурационных зависимостей от экземпляра и ее можно с легкостью перемещать с одного экземпляра SQL Server на другой.

По большому счету, что касается автономности, существует три вида баз данных:

- полностью автономные базы данных;
- частично автономные базы данных;
- неавтономные базы данных.

Полностью автономными являются такие базы данных, объекты которые не могут перемещаться через границы приложения. (Граница приложения определяет область видимости приложения. Например, пользовательские функции находятся в границах приложения, в то время как функции, связанные с экземплярами сервера, находятся вне границ приложения.)

Частично автономные базы данных позволяют объектам пересекать границы приложения, в то время как неавтономные базы данных вообще не поддерживают концепции границы приложения.

ПРИМЕЧАНИЕ

В SQL Server 2012 поддерживаются частично автономные базы данных. В будущих версиях SQL Server также будет поддерживаться полная автономность. Базы данных предшествующих версий SQL Server являются неавтономными.

Рассмотрим, как создать частично автономную базу данных в SQL Server 2012. Если существующая база данных my_sample является неавтономной (созданная, например, посредством инструкции CREATE DATABASE), с помощью инструкции ALTER DATABASE ее можно преобразовать в частично автономную, как это показано в примере 20.

Пример 20. Преобразование неавтономной базы данных в частично автономную

```
EXEC sp_configure 'show advanced options', 1;
RECONFIGURE WITH OVERRIDE;
EXEC sp_configure 'contained database authentication', 1; RECONFIGURE WITH
OVERRIDE;
ALTER DATABASE my_sampleNew SET CONTAINMENT = PARTIAL;
EXEC sp_configure 'show advanced options', 0;
RECONFIGURE WITH OVERRIDE;
```

Инструкция ALTER DATABASE изменяет состояние автономности базы данных my_sampie с неавтономного на частично автономное. Это означает, что теперь система базы данных позволяет создавать как автономные, так неавтономные объекты для базы данных my_sampie. Все другие инструкции в примере 20 являются вспомогательными для инструкции ALTER DATABASE.

ПРИМЕЧАНИЕ

Функция sp_configure является системной процедурой, с помощью которой можно, среди прочего, изменить дополнительные параметры конфигурации, такие как contained database authentication. Чтобы изменить дополнительные параметры конфигурации, сначала нужно присвоить параметру show advanced options значение 1, а потом переконфигурировать систему (инструкция reconfigure). В конце кода примера 20 этому параметру опять присваивается его значение по умолчанию — 0.

Теперь для базы данных sp_smaple можно создать пользователя, не привязанного к учетной записи.

Изменение таблиц

Для модифицирования схемы таблицы применяется инструкция ALTER TABLE. Язык Transact-SQL позволяет осуществлять следующие виды изменений таблиц:

- добавлять и удалять столбцы;
- изменять свойства столбцов;
- добавлять и удалять ограничения для обеспечения целостности;
- разрешать или отключать ограничения;
- переименовывать таблицы и другие объекты базы данных.

Эти типы изменений рассматриваются в последующих далее разделах.

Добавление и удаление столбцов

Чтобы добавить новый столбец в существующую таблицу, в инструкции ALTER TABLE используется предложение ADD. В одной инструкции ALTER TABLE можно добавить только один столбец. Применение предложения ADD показано в примере 21.

Пример 21. Добавление нового столбца в таблицу

```
USE sampleNew;
ALTER TABLE employee
    ADD telephone_no CHAR(12) NULL;
```

В примере 21 инструкция ALTER TABLE добавляет в таблицу employee столбец telephone_no. Компонент Database Engine заполняет новый столбец значениями NULL или IDENTITY или указанными значениями по умолчанию. По этой причине новый столбец должен или поддерживать свойство содержать значения NULL, или для него должно быть указано значение по умолчанию.

ПРИМЕЧАНИЕ

Новый столбец нельзя вставить в таблицу в какой-либо конкретной позиции. Столбец, добавляемый предложением ADD, всегда вставляется в конец таблицы.

Столбцы из таблицы удаляются посредством предложения DROP COLUMN. Применение этого предложения показано в примере 22.

Пример 22. Удаление столбца таблицы

```
USE sampleNew;
ALTER TABLE employee
    DROP COLUMN telephone_no;
```

В примере 22 инструкция ALTER TABLE удаляет в таблице employee столбец telephone_no, который был добавлен в эту таблицу предложением ADD в примере 21.

Изменение свойств столбцов

Для изменения свойств существующего столбца применяется предложение ALTER COLUMN инструкции ALTER TABLE. Изменению поддаются следующие свойства столбца:

- тип данных;
- свойство столбца принимать значения NULL.

Применение предложения ALTER COLUMN показано в примере 23.

Пример 23. Изменение свойств столбца

```
USE sampleNew;
ALTER TABLE department
    ALTER COLUMN location CHAR(25) NOT NULL;
```

В примере 23 инструкция ALTER TABLE изменяет начальные свойства (CHAR(30), значения NULL разрешены) столбца location таблицы department на новые (CHAR(25) , значения NULL не разрешены — NOT NULL).

Добавление и удаления ограничений для обеспечения целостности

Для добавления в таблицу новых ограничений для обеспечения целостности используется параметр ADD CONSTRAINT инструкции ALTER TABLE. В примере 24 показано использование параметра ADD CONSTRAINT для добавления проверочного ограничения.

Пример 24. Добавление проверочного ограничения

```
USE sampleNew;
CREATE TABLE sales
    (order_no INTEGER NOT NULL,
    order_date DATE NOT NULL,
    ship_date DATE NOT NULL);
ALTER TABLE sales
        ADD CONSTRAINT order_check CHECK(order_date <= ship_date);
```

В примере 24 инструкцией CREATE TABLE сначала создается таблица sales, содержащая два столбца с типом данных DATE: order_date и ship_date. Далее, инструкция ALTER TABLE определяет ограничение для обеспечения целостности order_check, которое сравнивает значения обоих этих столбцов и выводит сообщение об ошибке, если дата отправки ship_date более ранняя, чем дата заказа order_date.

В примере 25 показано использование инструкции ALTER TABLE для определения первичного ключа таблицы.

Пример 25. Определение первичного ключа таблицы

```
USE sampleNew;
ALTER TABLE sales
ADD CONSTRAINT primaryk_sales PRIMARY KEY(order_no);
```

В примере 25 столбец order_no определен, как первичный ключ таблицы sales.

Ограничения для обеспечения целостности можно удалить посредством предложения DROP CONSTRAINT инструкции ALTER TABLE, как это показано в примере 26.

Пример 26. Удаление ограничений для обеспечения целостности

```
USE sampleNew;
ALTER TABLE sales
    DROP CONSTRAINT order check;
```

В примере 26 инструкция ALTER TABLE удаляет проверочное ограничение order_check, установленное в примере 24.

ПРИМЕЧАНИЕ

Определения существующих ограничений нельзя модифицировать. Чтобы изменить ограничение, его сначала нужно удалить, а потом создать новое, содержащее требуемые модификации.

Разрешение и запрещение ограничений

Как упоминалось ранее, ограничение для обеспечения целостности всегда имеет имя, которое может быть объявленным или явно посредством опции CONSTRAINT, или неявно посредством системы. Имена всех ограничений таблицы (объявленных как явно, так и неявно) можно просмотреть с помощью системной процедуры sp_helpconstraint.

В последующих операциях вставки или обновлений значений в соответствующий столбец ограничение по умолчанию обеспечивается принудительно. Кроме этого, при объявлении ограничения все существующие значения соответствующего столбца проверяются на удовлетворение условий ограничения. Начальная проверка не выполняется, если ограничение создается с параметром WITH NOCHECK. В таком случае ограничение будет проверяться только при последующих операциях вставки и обновлений значений соответствующего столбца. (Оба параметра — WITH CHECK и WITH NOCHECK — можно применять только с ограничениями проверки целостности CHECK и проверки внешнего ключа FOREIGN KEY.)

В примере 27 показано, как отключить все существующие ограничения таблицы.

Пример 27. Отключение ограничений таблицы

```
USE sampleNew;
ALTER TABLE sales NOCHECK CONSTRAINT ALL;
```

В примере 27 все ограничения таблицы sales отключаются посредством ключевого слова ALL.

ПРИМЕЧАНИЕ

Применять опцию NOCHECK не рекомендуется, поскольку любые подавленные нарушения условий ограничения могут вызвать ошибки при будущих обновлениях.

Переименование таблиц и других объектов баз данных

Для изменения имени существующей таблицы (и любых других объектов базы данных, таких как база данных, представление или хранимая процедура) применяется системная процедура sp_rename. В примерах 28—29 показано использование этой системной процедуры.

Пример 28. Переименование таблицы

```
USE sampleNew;
EXEC sp_rename @objname = department, @newname = subdivision
```

В примере 28 таблице department присваивается новое имя subdivision.

Пример 29. Переименование столбца таблицы

```
USE sampleNew;
EXEC sp_rename @objname = 'sales.order_no', Snewname = ordernumber
```

В примере 29 столбцу order_no таблицы sales присваивается новое имя ordernumber. При переименовании столбца таблицы имя этого столбца требуется указывать в виде: *table_name. column_name* (т. е. *имя_таблицы.имя_столбца).*

ПРИМЕЧАНИЕ

Использовать системную процедуру sp_rename настоятельно не рекомендуется, поскольку изменение имен объектов может повлиять на другие объекты базы данных, которые

ссылаются на них. Вместо этого следует удалить объект и воссоздать его с новым именем.

Удаление объектов баз данных

Все инструкции Transact-SQL для удаления объектов базы данных имеют следующий общий вид:
DROP *object_type object_name*
Для каждой инструкции create *object* для создания объекта имеется соответствующая инструкция drop *object* для удаления. Инструкция для удаления одной или нескольких баз данных имеет следующий вид:
DROP DATABASE *database1* {, . . . }
Эта инструкция безвозвратно удаляет базу данных из системы баз данных.
Для удаления одной или нескольких таблиц применяется следующая инструкция:
DROP TABLE *table_name1* {, . . . }
При удалении таблицы удаляются все ее данные, индексы и триггеры. Но представления, созданные по удаленной таблице, не удаляются. Таблицу может удалить только пользователь, имеющий соответствующие разрешения.
Кроме объектов DATABASE и TABLE, в параметре objects инструкции DROP можно указывать, среди прочих, следующие объекты:

- TYPE (тип);
- SYNONYM (синоним);
- PROCEDURE (процедура);
- INDEX (индекс);
- VIEW (представление);
- TRIGGER (триггер);
- SCHEMA (схема).

Инструкции drop type и drop synonym удаляют тип и синоним соответственно.

Резюме

Язык Transact-SQL поддерживает большое число различных инструкций описания данных для создания, изменения и удаления объектов баз данных. Используя инструкции create *object* и drop *object* можно создавать и удалять, соответственно, следующие объекты баз данных:

- базы данных;
- таблицы;
- схемы;
- представления;
- триггеры;
- хранимые процедуры;
- индексы.

Структуру любого из перечисленных в предшествующем списке объектов баз данных можно изменить с помощью инструкции alter *object*. Обратите внимание на то обстоятельство, что единой стандартной инструкцией в данном списке является инструкция ALTER TABLE. Все другие инструкции типа ALTER *object* являются расширениями стандарта SQL для языка Transact-SQL.

Упражнения

Упражнение 1

Используя инструкцию CREATE DATABASE, создайте новую базу данных test_db, задав явные спецификации для файлов базы данных и журнала транзакций. Файл базы данных с логическим именем test_db_dat сохраняется в физическом файле C:\tmp\test_db.mdf, его начальный размер — 5 Мбайт, автоувеличение по 8%, максимальный размер не ограничен. Файл журнала транзакций с логическим именем test_db_log сохраняется в физическом файле C:\tmp\test_db_log.ldf, его начальный размер — 2 Мбайта, автоувеличение по 500 Кбайт, максимальный размер 10 Мбайт.

Упражнение 2

Используя инструкцию ALTER DATABASE, добавьте новый файл журнала в базу данных test_db. Файл сохраняется в физическом файле C:\tmp\emp_log.ldf, его начальный размер — 2 Мбайта, автоувеличение по 2 Мбайта, максимальный размер не ограничен.

Упражнение 3

Используя инструкцию ALTER DATABASE, измените начальный размер файла базы данных

test_db на 10 Мбайт.

Упражнение 4

В примере 4 для некоторых столбцов четырех созданных таблиц запрещены значения NULL. Для каких из этих столбцов это определение является обязательным, а для каких нет?

Упражнение 5

Почему в примере 4 тип данных для столбцов dept_no и project_no определен как CHAR, а не как один из целочисленных типов?

Упражнение 6

Создайте таблицы customers и orders, содержащие перечисленные в следующей таблице столбцы. Не объявляйте соответствующие первичный и внешние ключи.

customers	**orders**
customerid char(5) not null	orderid integer not null
companyname varchar(40) not null	customerid char(5) not null
contactname char(30) null	orderdate date null
address varchar(60) null	shippeddate date null
city char(15) null	freight money null
phone char(24) null	shipname varchar(40) null
fax char(24) null	shipaddress varchar(60) null
	quantity integer null

Упражнение 7

Используя инструкцию ALTER TABLE, добавьте в таблицу orders новый столбец shipregion. Столбец должен иметь целочисленный тип данных и разрешать значения NULL.

Упражнение 8

Используя инструкцию ALTER TABLE, измените тип данных столбца shipregion с целочисленного на буквенно-цифровой длиной 8 символов. Столбец может содержать значения NULL.

Упражнение 9

Удалите созданный ранее столбец shipregion.

Упражнение 10

Дайте точное описание происходящему при удалении таблицы с помощью инструкции DROP TABLE.

Упражнение 11

Создайте заново таблицы customers и orders, усовершенствовав их определение всеми ограничениями первичных и внешних ключей.

Упражнение 12

Используя среду SQL Server Management Studio, попробуйте вставить следующую новую строку в таблицу orders:

(10, 'ord0l', getdate(), getdate(), 100.0, 'Windstar', 'Ocean', 1).

Почему система отказывается вставлять эту строку в таблицу?

Упражнение 13

Используя инструкцию ALTER TABLE, определите значение по умолчанию столбца orderdate таблицы orders в виде текущей даты и времени системы.

Упражнение 14

Используя инструкцию ALTER TABLE, создайте ограничение для обеспечения целостности, ограничивающее допустимые значения столбца quantity таблицы orders диапазоном значений от 1 до

30.

Упражнение 15

Отобразите все ограничения для обеспечения целостности таблицы orders.

Упражнение 16

Попытайтесь удалить первичный ключ таблицы customers. Почему это не удается?

Упражнение 17

Удалите ограничение для обеспечения целостности prim_empl, определенное в примере 7.

Упражнение 18

В таблице customers измените имя столбца city на town.