**Цели работы:**

1. Создание объектов баз данных
2. Модифицирование объектов баз данных
3. 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.

**Пример 1. Код для создания простой базы данных**

USE master;

CREATE DATABASE sampleNew;

Код, приведенный в примере 1, создает базу данных, которая называется sampleNew. Результат выполнения представлен на рисунке 1.

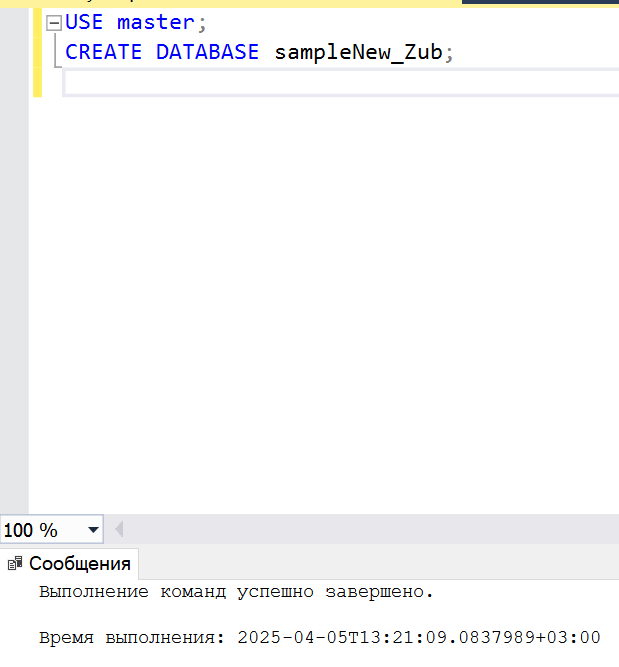


Рисунок 1 - создание БД sampleNew

**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 kb, tb и mb, означающих килобайты, терабайты и мегабайты соответственно. По умолчанию используется единица размера mb, т. е. мегабайты.). Результат выполнения представлен на

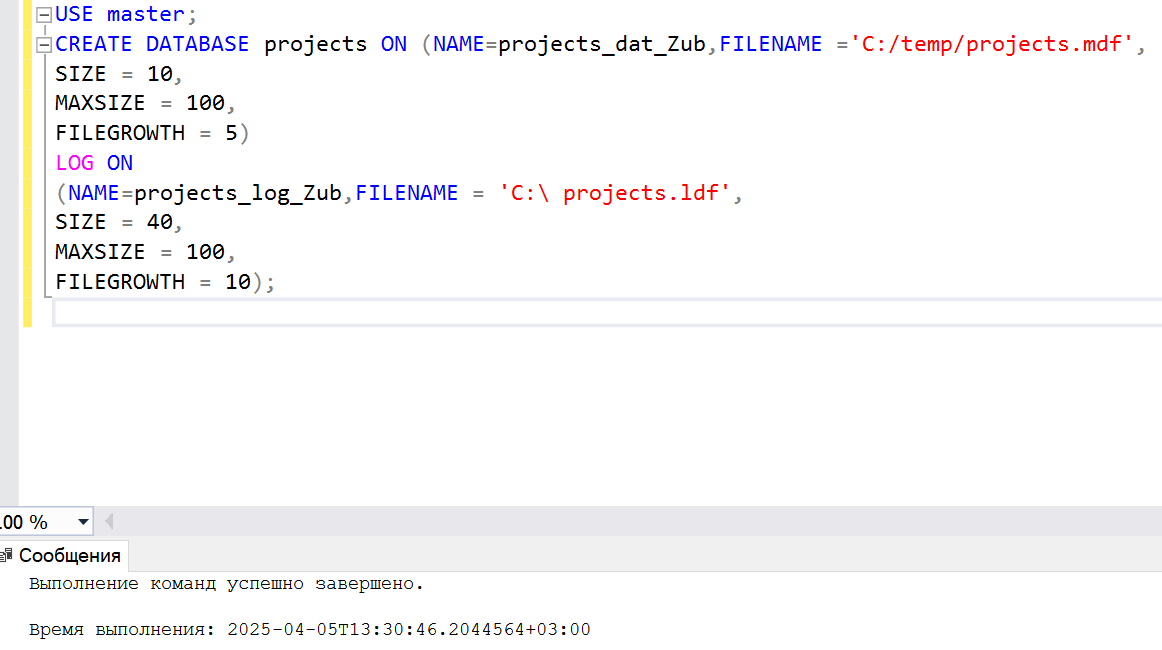


Рисунок 2 - Создание БД с передачей измененными параметрами по умолчанию

**Пример 3. Создание моментального снимка базы данных sampleNew\_Zub**

USE master;

CREATE DATABASE sample\_snapshot ON (NAME = ’sample\_Data’ ,

FILENAME = ’C:\temp\snapshot\_DB.mdf’)

AS SNAPSHOT OF sampleNew\_Zub;

Моментальный снимок существующей базы данных — это доступная только для чтения копия базы данных-источника, которая отражает состояние этой базы данных на момент копирования. Результат создания снапшота базы данных представлена на рисунке 3.



Рисунок 3 - создание снапшота БД

**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);

Выполнение скрипта представлено на

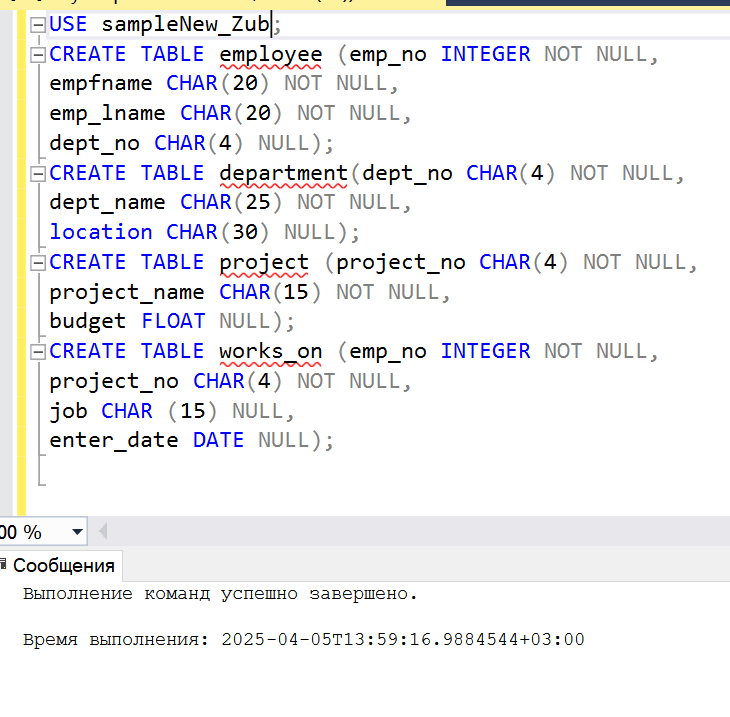


Рисунок 4 - Создание структуры БД sampleNew\_Zub

**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. Прмер выполнения представлен на рисунке 5.

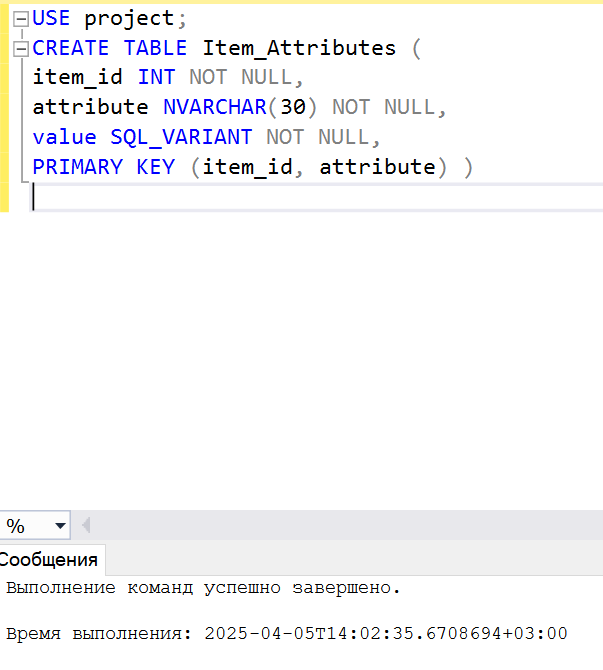


Рисунок 5 Создание таблицы, содержащей столбец типа sql\_variant

**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));

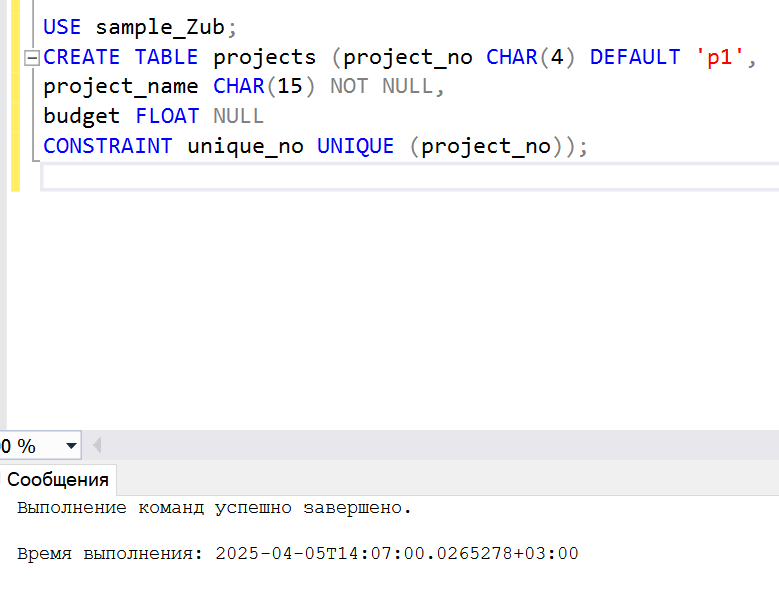


Рисунок 6 - Применение предложения unique

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. Результат выполнения скрипта представлен на рисунке 6.

**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. Реализация в SSMS представлена на рисунке 7.

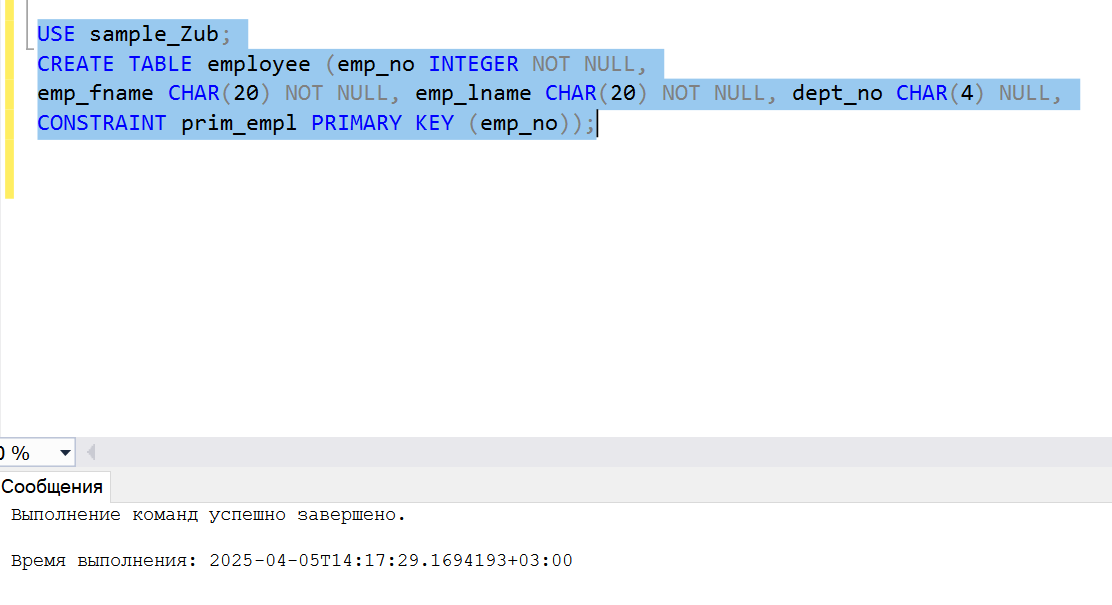


Рисунок 7 - Определение первичного ключа

**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. Добавление ограничения представлено на рисунке 8.

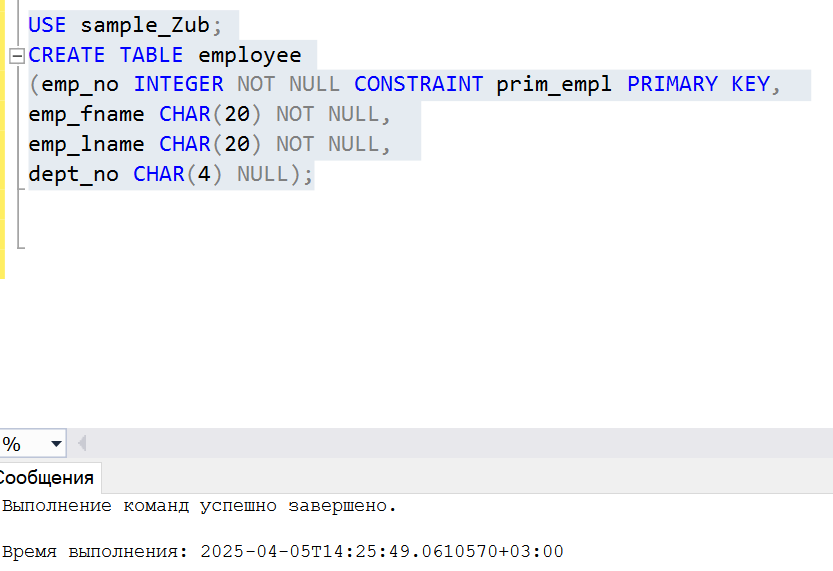


Рисунок 8 - Определение ограничения уровня столбца

**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. Пр менение параметра показано на рисунке 9.

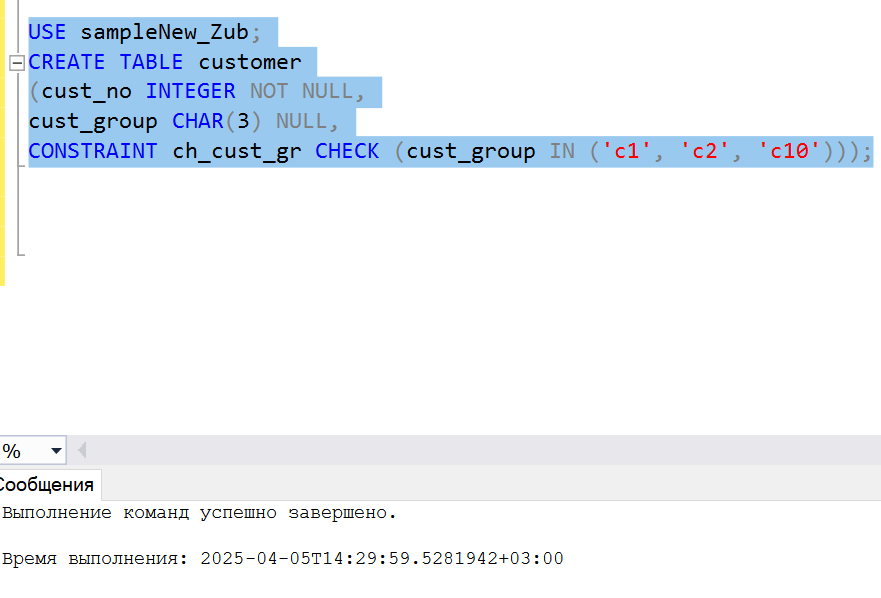


Рисунок 9 -Применение предложения check

**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. Реализация скрипта показана на рисунке 10.

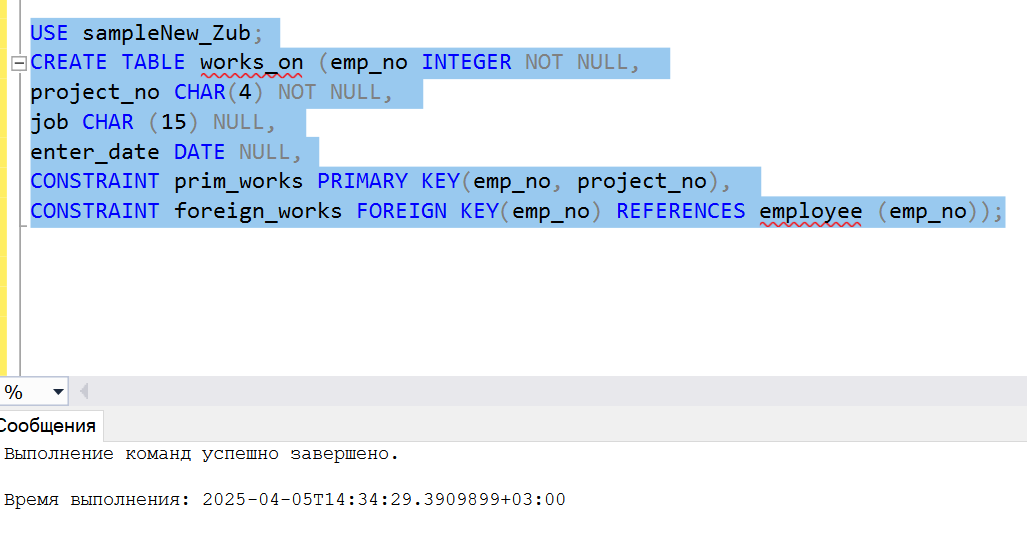


Рисунок 10 - Объявление внешнего ключа

**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));

Результат работы скрипта представлен на рисунке 11.

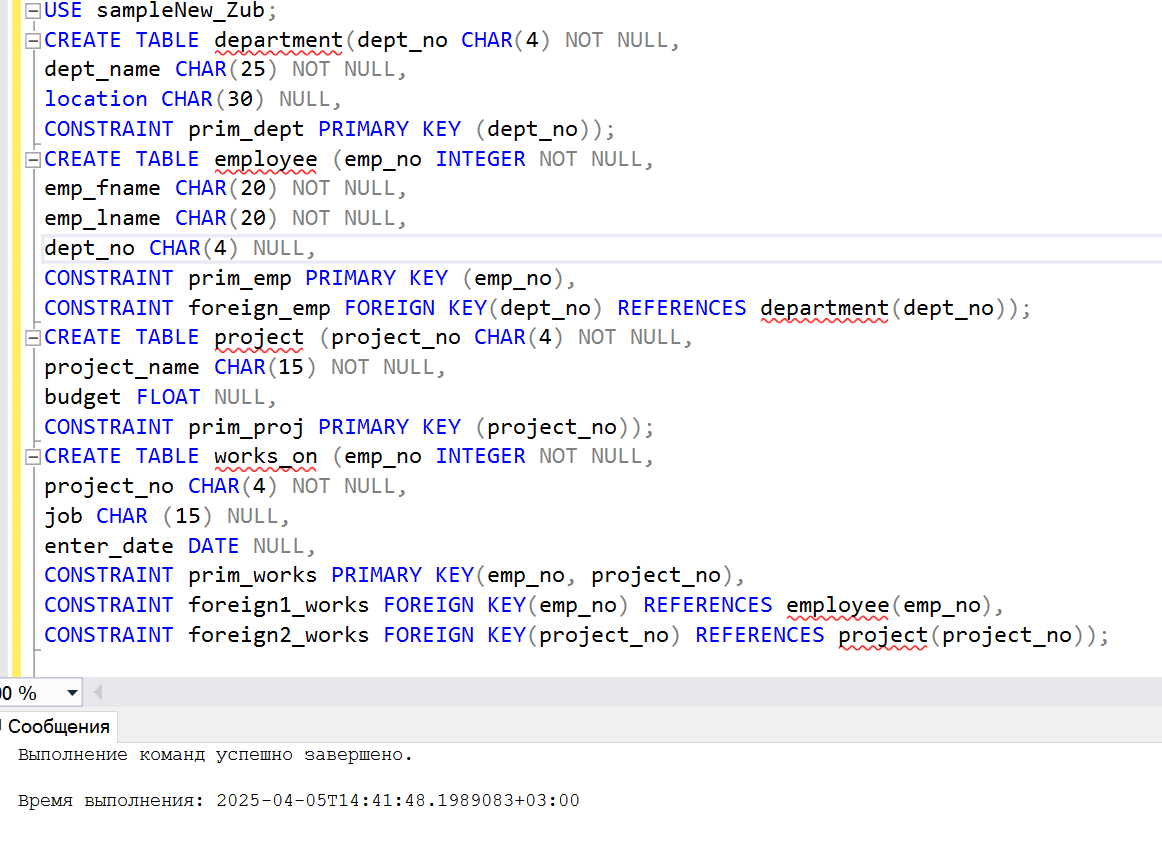


Рисунок 11 - Определение всех ограничений первичного и внешнего ключей для таблиц базы данных sampleNew\_Zub

**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 projact(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. Результат отображен на рисунке*12*.

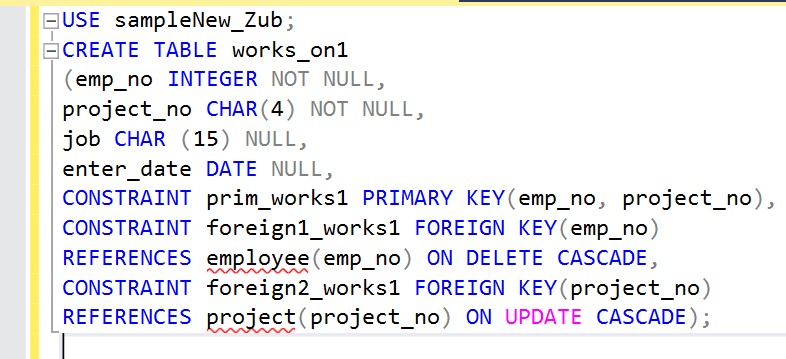


Рисунок 12 - Применение опций on delete и on update

**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. Результат представлен на рисунке 13.

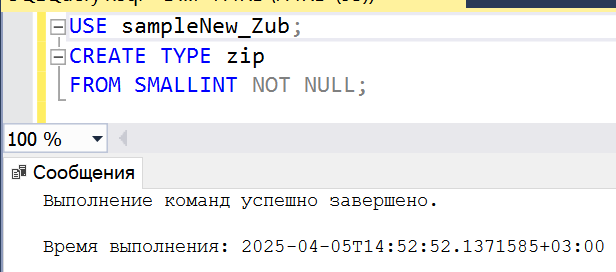


Рисунок 13 - Создание псевдонимного типа данных С ПОМОЩЬЮ инструкции CREATE TYPE

**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. Результат работы скрипта показан на рисунке 14.

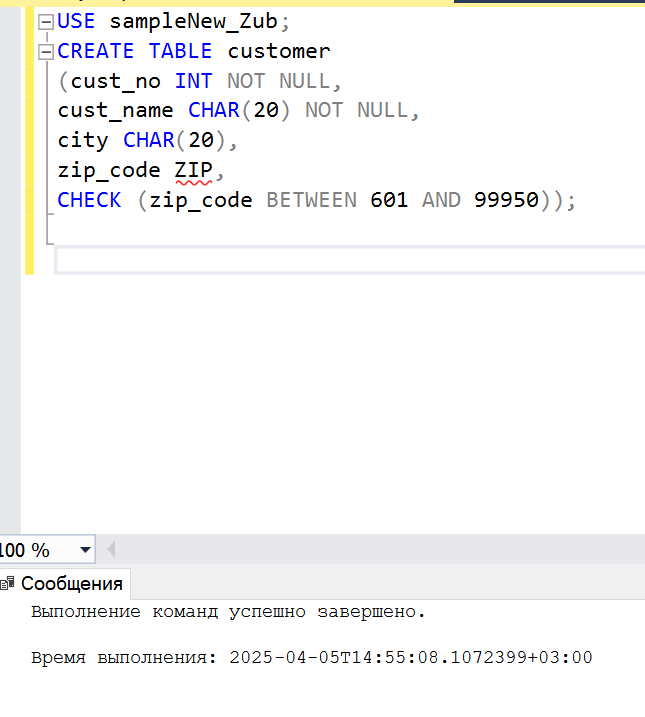


Рисунок 14 - Определение столбца псевдонимным типом данных

**Пример 16. Создание табличного типа**

USE sampleNew;

CREATE TYPE person\_table\_t AS TABLE

(name VARCHAR(30),

salary DECIMAL(8,2));

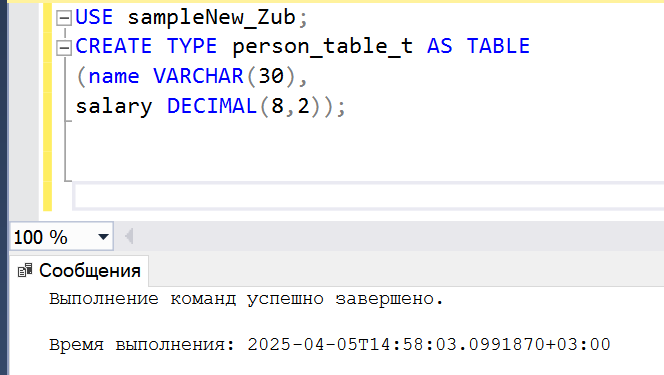


Рисунок 15 - Создание табличного типа

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. Результат отображен на рисунке 15.

**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. Результат отображен на рисунке 16.

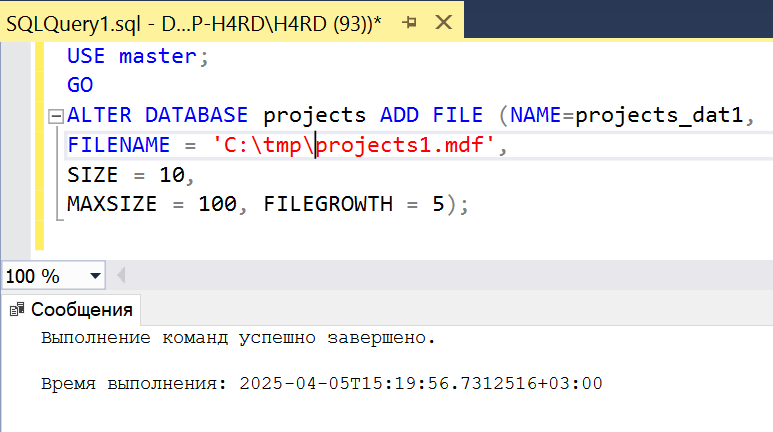


Рисунок 16 - Добавление нового файла в базу данных

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.

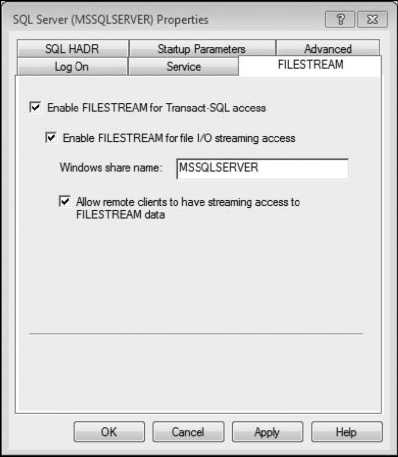


Рис. 1. Диалоговое окно SQL Server Properties, вкладка FILESTREAM



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); ♦ view (представление);

synonym (синоним); ♦ trigger (триггер);

procedure (процедура); ♦ schema (схема).

index (индекс);

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.