ШАГ
компьютерная
АКАДЕМИЯ

Программирование
и администрирование
СУБД

Microsoft®
SQL Server

Урок №1

Работа с таблицами и представлениями в MS SQL Server

Содержание

1. Пакетные запросы и сценарии

Сценарий – это набор операторов, которые хранятся отдельным файлом, который может запускаться на выполнение и использоваться повторно.

В T-SQL также выделяют понятие пакета. Пакетный запрос (пакет) – это последовательность операторов T-SQL, интерпретируемых сервером вместе, то есть как одна логическая единица. Операторы, которые являются составной частью пакета, ссылаются на сервер как единое целое. Чтобы разделить сценарий на несколько пакетов, используется оператор GO. Оператор GO вызывает компиляцию всех операторов от начала сценария или предыдущего оператора GO (в зависимости от того, что ближе), после чего полученный план выполнения передается на сервер независимо от всех других пакетов. К примеру:

```
--1 команда. Делаем базу данных master активной с помощью
 оператора use
 use master
--2 команда. Выводим на экран всю информацию из системной
 таблицы sysobjects
 select *
 from sysobjects
- Посылаем на сервер пакет из двух команд для обработки
 Go
```

Пакеты подчиняются следующим правилам:

1. Все операторы пакета компилируются как единое целое.
2. Если в пакете существует синтаксическая ошибка, – отменяется выполнение всего пакета.
3. Если во время выполнения пакета в одном из операторов происходит ошибка, то этот оператор пропускается и продолжается выполнение других операторов. Например, если пакет содержит три оператора CREATE TABLE и во втором операторе происходит ошибка, то SQL Server создаст только первую и третью таблицу.
4. В пределах одного пакета нельзя сначала менять поля таблицы, а затем использовать эти новые поля.
5. Операторы SET выполняются сразу, кроме случаев, когда установлены опции QUOTED_IDENTIFIER и ANSI_NULLS.
6. В один пакет нельзя совместно помещать следующие операторы:
   - CREATE RULE;
   - CREATE TRIGGER;
   - CREATE PROCEDURE;
   - CREATE DEFAULT;
   - CREATE VIEW.

2. Типы данных в MS SQL Server. Расчетные поля. Поля с ограничениями (check)

MS SQL Server, как уже было сказано, является реляционной базой данных, и поэтому все ее данные хранятся в таблицах, состоящих из записей и полей. Каждое поле таблицы имеет имя и содержит данные только одного типа. Тип данных позволяет указать на то, какие именно данные можно хранить в каждом отдельном поле и ограничивать диапазон их значений, что позволит воспрепятствовать введению неверных данных. В MS SQL Server 2008 выделяют следующие типы данных:

2.1. Строчные типы данных

Char (к-во_символов) – строка фиксированной длины, количество символов которого не может превышать установленного значения.

Если в поле будет содержаться строка длиной, например, 5 символов, а тип для хранения данных был определен как char (20), то на диске для него все равно выделено 20 байт, а не используемые байт наполняются пробелами (space). Если строка, вводится будет более 20 символов, он будет обрезан до необходимой длины.

Максимальная длина – 8000 байт (8000 знаков).

Varchar (к-во_символов) * – строка переменной длины, максимальное количество символов которой определяется параметром.

Если в поле будет содержаться строка длиной, например, 5 символов, а тип для хранения данных был определен как varchar (20), то на диске для него будет выделено 5 байт. Если строка, вводится будет более 20 символов, он будет обрезан до необходимой длины.

Максимальная длина – 8000 байт (8000 знаков). При использовании в качестве параметра значения "MAX" (varchar (MAX)) максимальная длина хранимых данных достигает 2 Гб (до 1.073.741.824 символов)

Nchar (к-во_символов), nvarchar (к-во_символов)* – аналогичные типам char и varchar, но предназначены для хранения данных в формате Unicode (2 байта на символ).

Максимальное количество необходимых байт – 8000, то есть максимальное количество символов – 4000. При использовании в качестве параметра значения "MAX" (nvarchar (MAX)) максимальная длина хранимых данных достигает 2 Гб (до 536 870 912 символов)

2.2. Целочисленные типы данных

Tinyint – размер – 1 байт.
Диапазон значений: 0..255
Smallint – размер – 2 байта.
Диапазон значений: -32767 .. + 32676
Int – размер – 4 байта.
Диапазон значений: -2147483647 .. + 2147483647
Bigint – размер – 8 байт.

Диапазон значений: -9223372036854775808 .. +9223372036854775808

2.3. Типы данных для хранения действительных чисел

Real (к-во_разрядов) – размер – 4 байта.

Максимальная разрядность – 7.

Например, при типе данных real (5), в нем можно хранить значения типа 1.2345, 123.45, но не более определенной разрядности, то есть 5.

Данный тип поддерживается только для совместимости со стандартом SQL-92 и заменен на float.

Float (к-во_разрядов) – размер – 4 или 8 байт.

Максимальная разрядность – 38.

По умолчанию устанавливается точность15 разрядов.

Numeric (общее_к-во_разрядов, сколько_для_ дробной_части) – Сохраняет действительные числа с максимальным количеством разрядов не более 38. По умолчанию разрядность устанавливается 18 и 0. При введении 0 для десятичных разрядов числовые значения приводятся в целочисленные.

Например, при типе данных numeric (7,2) можно хранить значения типа 123,45 и 12345,67, но не 1234567,89.

Если количество разрядов после запятой больше, чем указано, то при сохранении будет осуществлено математическое сокращения. В случае превышения разрядности мантиссы будет выведена ошибка при сохранении данных.

Decimal (общее_к-во_разрядов, сколько_для_ дробной_части) – аналог numeric.

2.4. Типы данных для хранения даты и времени

Date – размер – 3 байта.

Диапазон дат: 0001/01 / 01..9999 / 12/31

Дата представлена в формате YYYY-MM-DD

Новый тип данных в SQL Server 2008

Time [(точность_в_секундах)] – размер – 5 байт.

Диапазон времени: 00: 00: 00..23: 59: 59.9 (7)

Дата представлена в формате HH: MM: SS [.nnnnnnn]

Обозначения n * указывает на доли секунды (0..9999999).

Новый тип данных в SQL Server 2008

Smalldatetime – размер – 4 байта.

Диапазон дат: 1900/01 / 01..2079 / 06/06.

Дата представлена в формате YYYY-MM-DD HH: MM: SS

Datetime – размер – 8 байт.

Диапазон дат: 1753/01 / 01..9999 / 12/03

Диапазон времени: 00: 00: 00..23: 59: 59.0997

Дата представлена в формате YYYY-MM-DD HH: MM: SS.MS

Datetime2[(точность_в_секундах)] – размер – 6 байт для представления точности меньше 3 цифр, 7 байт – для точности в 3 и 4 цифры. Для представления других значений точности необходимо 8 байт.

Диапазон дат: 0001/01 / 01..9999 / 12/31.

Диапазон времени: 00: 00: 00..23: 59: 59.9 (7)

Дата представлена в формате YYYY-MM-DD HH: MM: SS.MS

Новый тип данных в SQL Server 2008

Datetimeoffset [(точность_в_секундах)] – дата и время с учетом часового пояса в 24-часовом формате.

Необязательный параметр p указывает на точность в секундах.

Размер – 10 байт.

Диапазон дат: 0001/01 / 01..9999 / 12/31

Диапазон времени: 00: 00: 0 (7) .. 23: 59: 59.9 (7)

Дата представлена в формате YYYY-MM-DD HH: MM: SS [.nnnnnnn] [{+ | -} hh: mm]

Обозначения n * указывает на доли секунды (0..9999999). Обозначение hh может принимать значения от -14 до +14. Обозначение mm принимает значения от 00 до 59.

Новый тип данных в SQL Server 2008

2.5. Типы данных для хранения денежных величин

Smallmoney – размер – 4 байта.

Диапазон значений: -214 748,3647 .. + 214 748,3647.

Money – размер – 8 байт.

Диапазон значений: -922 337 203 685 477,5808 .. + 922337203685 477,5808.

2.6. Бинарные типы данных

Binary (к-во_байт), **varbinary** (к-во_байт)* – предназначены для хранения информации в битном виде, то есть данные, которые в них хранятся, представляют собой последовательность 0 и 1, представленных в шестнадцатеричной системе счисления и организованных в пары. При вводе данных в бинарном виде к ним добавляется префикс 0x. Например, для создания в поле битового значения 10, введите 0x10.

Принцип работы аналогичен типам char и varchar.

Максимальный размер – 8000 байт.

При использовании в качестве параметра типа varbinary значения "MAX" (varbinary (MAX)) максимальная длина хранимых данных достигает 2 Гб

Filestream – новый тип данных в SQL Server 2008, обеспечивает сохранение больших объемов двоичных данных в файловой системе NTFS, при этом они остаются частью БД с поддержкой целостности транзакций. Это позволяет размещать двоичные данные за пределами БД и при этом обеспечивать доступ к ним.

2.7. Дополнительные типы данных

Bit – размер: 1 бит

Двоичный тип данных, который используется для сохранения булевых значений: 0 (false) или 1 (true). Поля, которые имеют тип данных bit могут содержать NULL значения и не могут быть проиндексированы. При создании в таблице нескольких полей данного типа SQL Server объединяет их в группы по 8-ми полей в каждой, то есть суммарно по 1 байта.

Uniqueidentifier (RowGUID) – размер – 16 байт

При репликации данных каждое поле реплицированной таблицы должно иметь уникальный идентификатор, для которого желательно использовать тип данных uniqueidentifier, что имеет свойство ROWGUIDCOL. При подключении данного свойства полю предоставляется GUID (Global Unique Identifier – глобально уникальный идентификатор). GUID определяется двумя способами:

- с помощью функции NEWID;
- путем приведения строчной константы до 16 системы счисления в формате xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx. Например, 8FE17A24-B1AA-C790-23DA-2749A3E09AA2

В типе данных uniqueidentifier можно использовать символы =, <> IS NULL и IS NOT NULL

Text, image – эти типы данных считаются устаревшими и вместо них рекомендуется использовать типы varchar (max) и varbinary (max) соответственно (см. Примечание). Использовался для сохранения в поле данных размером более 8000 байт. Эти типы данных нужны при сохранении объектов BLOB (Binary Large Object – большой объект двоичных данных), поскольку они объявляют поля, содержащие до 2Гб битной и текстовой информации.

При объявлении типа данных text и image в строку добавляется 16-битный указатель, который определяет страницу данных размером 8 Кб, которая содержит дополнительные данные. Если дополнительные данные превышают 8 Кб, тогда создаются указательными на дополнительные страницы Вашего BLOB.

Сохранение и использование текстовых данных и рисунков существенно снижает производительность базы данных, поскольку большие объемы данных медленно обрабатываются при выполнении запросов на вставку, обновление или удаление. С целью ускорения выполнения операций с большими типами данных используется команда WRITETEXT.

Ntext – аналогично типа данных text, но данные хранятся в формате Unicode (2 байта на символ), поэтому

при максимуме данных 2 Гб, можно помещать символов в два раза меньше.

Вместо данного типа рекомендуется использовать тип nvarchar (max).

Sql_variant – позволяет сохранить значение любого типа, кроме типов данных ntext и timestamp. Этот тип используется для сохранения значений полей в переменных, в качестве значений параметров, для представления результатов выполнения пользовательских функций.

Этот тип данных ОПАСНЫЙ (!), поскольку он позволяет объявлять поле или переменную без явного указания типа данных, которые будут в нем храниться. После этого sql_variant автоматически приводится к типу данных, добавляемых в поле. В связи с этим, может возникнуть ошибка в случае повторного внесения в данное поле данных, которые имеют отличный тип.

Table – используется для сохранения набора данных, которые будут использоваться в будущем. Объявления данного типа аналогично созданию временных таблиц.

Geography – представляет собой пространственный тип данных на основе Microsoft .NET Framework, в котором используется сферическая модель нашей планеты. В этом типе сохраняются точки, линии, многоугольники и наборы координат долготы и широты.

Появление данного типа данных в SQL Server 2008 обусловлено использованием во многих приложениях картографических функций.

Geometry – пространственный тип данных SQL Server 2008 на основе Microsoft .NET Framework, основным назначение которого, в отличие от типа GEOGRAPHY, есть

навигация и картография на сферической модели Земли. Данный тип соответствует спецификации Open Geospatial Consortium (OGC) и основан на плоской модели.

Hierarchyid – тип данных SQL Server 2008 переменной длины, который предназначен для хранения данных, которые имеют древовидную структуру, например, организационные схемы. Он реализован как пользовательский тип CLR, содержащий несколько встроенных методов для создания узлов иерархии и гибкого манипулирования ими.

Cursor – размер – 1 байт

Содержит ссылки на курсор, который используется для выполнения операций. Данный тип нельзя использовать в таблицах.

Xml – сохранение XML-документов размером до 2 Гб. Задаваемые параметры позволяют сохранять в полях только документы, соответствующей структуры.

2.8. Автоматически обновляемые типы данных

Rowversion (старое название- **timestamp**) – Размер – 8 байт.

При добавлении записи в таблицу с полем данного типа, в него автоматически добавляется новое значение времени. При обновлении таблицы значение поля rowversion также обновляются.

Новое значение представляет собой набор автоматически сгенерированных чисел в двоичном формате и поэтому его данные сохраняются в виде binary (8) со свойством NOT NULL или varbinary (8) со свойством NULL. В одной таблице может существовать несколько полей данного типа.

Сгенерированные значения удобно использовать для отслеживания порядка добавления элементов в таблицу.

Примечание! * *Типы данных text и ntext предназначены для хранения больших массивов текстовых данных, а image – двоичных. Но все же ряд операций для полей данного типа запрещены, например, к ним нельзя применять оператор равенства или конкатенации, их можно использовать во многих системных функциях. В связи с этими ограничениями, еще в SQL Server 2005 появились типы данных varchar (max), nvarchar (max) и varbinary (max). Типы varchar (max) и nvarchar (max) объединяют возможности типов text / ntext и varchar / nvarchar, могут хранить данные до 2 Гб и не имеют ограничений по использованию с различными операциями и функциями. Что качается типа varbinary (max), то он может хранить данные такого же объема, как и image (до 2 Гб) и может использоваться во всех операциях и функциях, где допустимые типы данных binary / varbinary.*

Стоит также отметить, что благодаря интеграции CLR и SQL Server (начиная с версии SQL Server 2005) можно создавать собственные пользовательские типы данных CLR. Для этого следует осуществить следующие шаги:

1. Создать класс на одном из языков программирования Microsoft .NET, который соответствует спецификации пользовательских типов, например, C#.
2. Написанный класс скомпилировать в динамично подключаемую библиотеку (DLL).
3. Зарегистрировать библиотеку в экземпляре SQL Server. Это может сделать только член серверной роли sysadmin.

4. В базе данных включить поддержку типов данных CLR с помощью утилиты Surface Area Configuration. При отключении CLR все поля с пользовательскими типами данных CLR будут недоступны.

К сожалению, изучение создания и использования пользовательских типов данных CLR выходит за рамки нашего курса. Более подробно о типах данных CLR смотрите документацию SQL Server.

3. Домены и их создание

В SQL Server существует возможность создавать собственные типы данных, которые основаны на базовых типах и включают ряд ограничений. Для чего? Как вы знаете, существует определенный набор типов данных, которые определяют, какие данные будут храниться в отдельных полях таблицы. Но, когда необходимо установить более жесткие условия на ввод данных, характеристик базового типа данных может быть недостаточно. Например, необходимо задать условие на введение корректного возраста работника (положительное значение) или установить условие на ввод значения в определенном диапазоне и тому подобное. Для установления таких ограничений используют пользовательские типы данных, которые называют доменами.

Домен – это тип, определенный пользователем для удобства применения определенных ограничений или совокупности параметров базовых типов. В стандарте SQL2 указывается, что домен реализован как часть базы данных. Согласно этому стандарту, домен является именуемой совокупностью значений и используется в БД как дополнительный тип данных. После создания домена на него можно ссылаться как на обычный тип данных.

Как уже было сказано, домены привязаны к конкретной базе данных, но если существует необходимость использования его во всех базах данных в рамках текущего сервера, следует включить его в базу данных model.

В SQL Server 2008 было снято 8 килобайтное ограничения для пользовательских типов (User Defined Type, UDT), то есть для доменов, которое существовало в ранних версиях. Это позволило значительно расширить возможности пользователей.

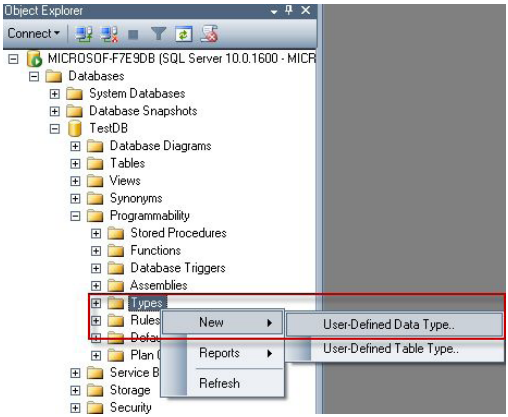
Существует **три способа создания доменов**:

- с помощью Визарда студии;
- с помощью системных хранимых процедур;
- с помощью команд SQL.

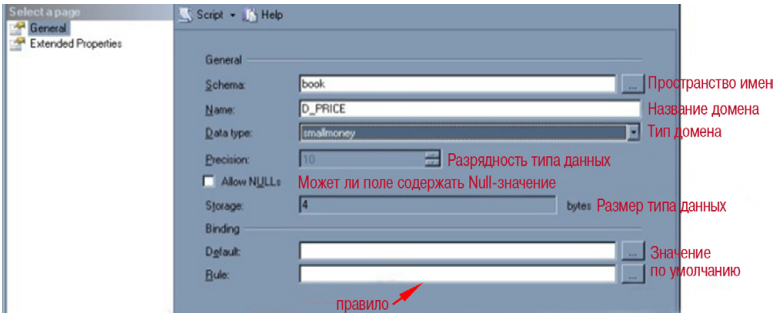
3.1. Средствами MS SQL Server

3.1.1. Создание, модификация и удаление домена

Для этого нужно выбрать необходимую базу данных, в ней выбрать папку, где размещаются ее пользовательские типы данных **(Programmability / Types)**. После этого следует вызвать ее контекстное меню и выбрать пункт **New / User-Defined Data Type.**



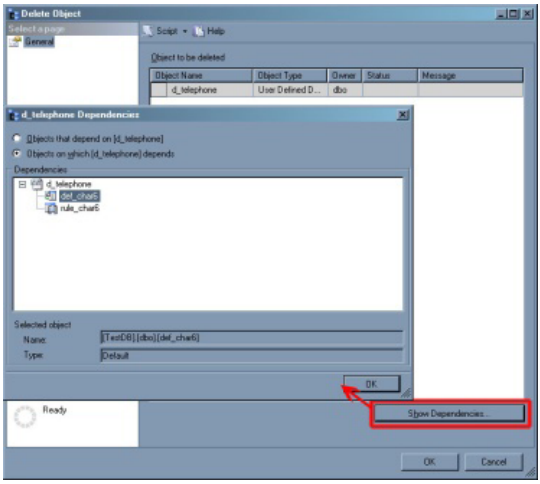
Далее вы увидите диалоговое окно для создания нового домена



Следует отметить, что поле **Precision** активируется в случае, если для типа данных можно указать разрядность, а поле Storage, если существует возможность задать размер. Например, вы можете выбрать тип данных varchar и задать ему длину 10.

Для удаления доменов в Management Studio следует открыть папку User-defined Data Types и в контекстном меню необходимого типа данных выбрать пункт **Delete (Удалить)**.

Перед Вами появится диалоговое окно **Delete Objects (Удаление Объектов)**, которое имеет следующий вид:



В нем вы можете еще раз просмотреть тип, который вы удаляете и, при нажатии кнопки "OK", тип данных будет уничтожен. Но следует помнить, что удаление типа домена будет невозможным в случае его использования в базе данных в данный момент. Для того, чтобы проверить используется пользовательский тип, нажмите на кнопку "Show Dependencies ..." (Показать зависимости), как показано выше. При этом на экране появится список таблиц и полей, в которых присутствует проверяемый пользовательский тип.

3.1.2. Значение по умолчанию и правила

Значение по умолчанию – это значение, которое автоматически присваивается записям поля, если им не было присвоено значение явно. Значение по умолчанию задаются в директиве INSERT с помощью оператора **DEFAULT** или в случае пропуска данных в списке ввода.

SQL Server поддерживает **два вида значений по умолчанию**:

- **ограничение DEFAULT в формате ANSI**, к которым относят: свойство IDENTITY (для генерации уникальных значений подобно типа данных "Счетчик"), ограничение на проверку данных (check), ограничение первичного, внешнего, уникального ключей, информация о которых размещается в системных таблицах sysreferences , syscomments, sysobjects и иногда в sysforeignkeys;
- **отдельные объекты-значения**, для создания которых используется оператор CREATE DEFAULT:

```
CREATE DEFAULT имя_значения_по_умолчанию
AS выражение_константа
```

Выражение-константа соответствует типу данных полей, для которых используется значение по умолчанию. К примеру:

- для типа char (3) – константа "x" или "xx", или "xxx";
- для целочисленного типа – произвольное число;
- для битных типов – константа начинается с 0x;
- для денежных типов – константа начинается с $;
- для типов данных в формате Unicode – константа начинается с N;

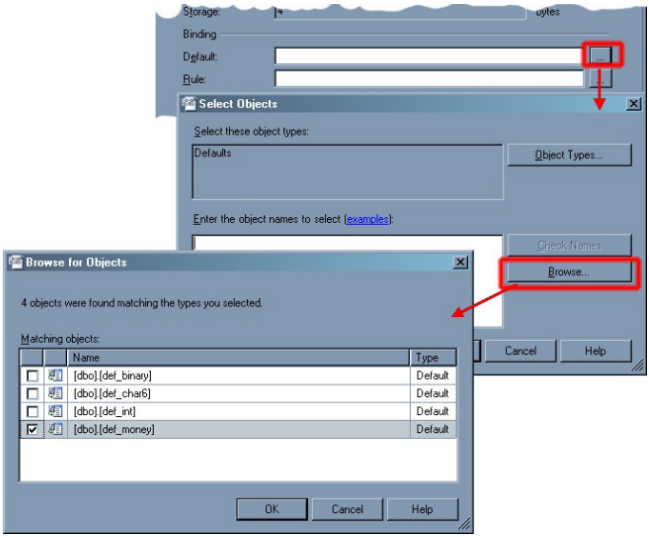
В выражении можно использовать функции SQL, но при условии, если они возвращают данные определенного типа.

В каждой базе данных значения по умолчанию (как и большинство объектов базы данных) хранятся в системной таблице **sysobjects**, текст оператора записывается в таблицу **syscomments**, а физическое размещение находится в папке по пути **Programmability / Defaults**.

Итак, создадим несколько значений по умолчанию:

```
create default def_int as 5 /*для целочисленных типов*/
go
create default def_char6 as 'Ivanko'/* для строчных
типов, размером не менее 6 символов*/
go
create default def_money as $5.0 /*для денежных типов
данных */
go
create default diff binary as 0x00 /*для двоичных типов*/
go
```

Итак, имея в распоряжении несколько значений по умолчанию, выберем одно из них для нашего домена, то есть привяжем созданное значение для домена:



Удалить существующее значение по умолчанию можно с помощью оператора **DROP DEFAULT**.

```
 DROP DEFAULT имя_домена [,...n]
--например
 drop default def_int
```

Со значением по умолчанию разобрались, остались правила. **Правила** обеспечивают дополнительную поддержку доменной целостности с помощью проверки допустимости значений. Заданные с помощью правил **значения проверяются на:**

- соответствие шаблона с помощью оператора LIKE;
- соответствие значению из множества с помощью оператора IN;
- принадлежность диапазона с помощью оператора BETWEEN.

Для создания правил используют оператор CREATE RULE:

```
 EATE RULE имя_правила
 AS условие_выражение
-- условие-выражение должно иметь следующую форму
 @сменная директива_where
```

Имя правила задается произвольно, но в большинстве случаев его задают таким образом, чтобы оно было созвучно с именем поля, на которое накладывается правило.

Условие в данном операторе представляет собой произвольную допустимую директиву WHERE, включая арифметические операторы, операторы BETWEEN, LIKE, IN, AND, OR, NOT и тому подобное. Но правило не может ссылаться на переменные значения или другие поля базы данных. Для этого нужно воспользоваться ограничениями, которые задаются значениями по умолчанию или триггерами.

К примеру:

```
create rule rule_int as @ivar >= 0
go
create rule rule_char6 as @name like 'A%' or @name like
'B%'
go
create rule rule_money as @price between $10.0 and $50.0
go
```

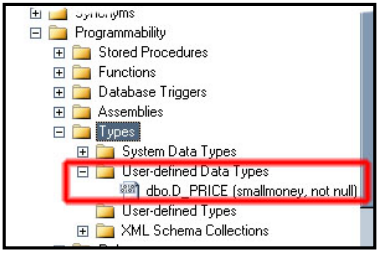
Рабочие правила, как и значение по умолчанию, представляют собой отдельные объекты базы данных и хранятся в тех же системных таблицах: sysobjects и syscomments, а физически располагаются по пути **Programmability / Rules**.

Привязка правила к домену осуществляется аналогично, как и привязка значения по умолчанию.

Удалить уже существующее правило можно с помощью оператора DROP RULE.

```
DROP RULE имя_правила [,...n]
--например
drop rule rule_char6
```

После всех действий, мы имеем достаточно неплохой домен, содержащий значение по умолчанию и правило на проверку введенных значений:



Модифицировать домен, то есть изменить его параметры в SQL Server 2008 невозможно. Допускается только его переименование, которое можно сделать с помощью пункта контекстного меню необходимого домена **"Rename"**.

3.2. С помощью SQL запросов

В Transact-SQL существует инструкция CREATE TYPE, которая позволяет создать пользовательский тип данных (домен).

```
CREATE TYPE [схема.] имя_домена
{
  FROM базовый_тип_данных
             [ ( общее_количество_разрядов [,
разрядность_дробной_части ] )  ]
             [ NULL | NOT NULL ]
| EXTERNAL NAME имя_сборки [.имя_класса ]
| AS TABLE ( список_полей_типа [ конструкции ] [ ,...n ]
)
}
```

При создании домена необходимо указать название схемы, к которой он будет принадлежать (более подробно о схемах читайте в разделе "Схемы"), название домена, базовый тип (какой тип домен расширяет) с разрядностью и точностью, если это необходимо. Поле типа домена можно определить как NOT NULL, что укажет на обязательность внесения данных в поле.

Если планируется использовать в качестве домена тип данных CLR, то необходимо указать сборник SQL Server текущей базы данных (!), в которой он описан. После этого можно указать название класса в середине сборки, который реализует определенный пользователем тип. При этом имя класса может быть указано в квадратных скобках ([]) с указанием пространства имен. Если аргумент "имя_класса" не указан, SQL Server считает, что его значение равно значению аргумента "базовый_тип_данных".

Вы можете создать домен табличного вида, задав при этом список полей такой таблицы с перечнем необходимых конструкций. Более подробно об объявлении полей таблицы смотрите в разделе **"Таблицы. Основы построения таблиц средствами SQL ".**

Приведем несколько примеров на создание доменов с помощью оператора SQL. Сначала приведем простой пример создания доменов на основе базовых типов.

```
create type d_name from varchar(32) not null;
create type dbo.d_PersonName from nvarchar(32) not null;
```

Создадим домен Utf8String, который ссылается на класс utf8string в сборнике utf8string. Отметим, что перед тем как создать такой тип, сборник utf8string нужно

зарегистрировать в локальной базе данных. Зарегистрировать сборку можно с помощью оператора **CREATE ASSEMBLY** (см. Документацию по SQL Server).

```
-- 1. Регистрируем сборку в базе данных
 create assembly utf8string
 from '\\127.0.0.1\library\utf8string.dll'
 go
-- 2. Создаем домен на основе типа utf8string со сборки
 utf8string.dll
 create type Utf8String
 external name utf8string.[Microsoft.Samples.SqlServer.
 utf8string]
 go
```

Следующий пример продемонстрирует создание домена вида таблицы, которая имеет два поля.

```
 create type tabletype
 as table (name varchar(10), digit int)
```

Такой табличный домен будет размещаться в папке **User-Defined Table Types**, которая также размещается в директории Programmability / Types.

Для удаления домена используется оператор **DROP TYPE**:

```
 DROP TYPE [схема.] имя_домена [,...n]
-- например
 drop type d_name, d_PersonName;
```

К сожалению, оператора SQL для модификации домена в MS SQL Server не существует.

3.3. С помощью хранимых процедур

Работать с доменами можно также с помощью системных хранимых процедур. Рассмотрим их по порядку.

1. Для создания доменов существует системная процедура sp_addtype:

```
sp_addtype [ @typename = ] имя_домена,
           [ @phystype = ] базовый_тип_данных
           [, [ @nulltype = ] 'null_type' ]
```

Аргумент nulltype указывает на способ обработки значений NULL доменом. Данный аргумент имеет тип varchar (8), значение по умолчанию NULL, и его следует указывать в одинарных кавычках ('NULL', 'NOT NULL' или 'NONULL ').

К примеру:

```
exec sp_addtype d_salary, 'money'
exec sp_addtype d_telephone, 'varchar(24)', 'NOT NULL'
exec sp_addtype d_birthday,datetime, 'NULL'
```

2. Удаление домена осуществляется с помощью системной хранимой процедуры **sp_drop type:**
   sp_droptype [@typename =] имя_домена [,...n]
   Например:

```
exec sp_droptype d_salary
exec sp_droptype d_birthday
```

3. Модифицировать домен нельзя (то есть изменить его параметры), но допускается его переименование. Переименовать домен можно с помощью системной хранимой процедуры **sp_rename**.

```
sp_rename [ @objname = ] 'имя_домена',
          [ @newname = ] 'новое_имя_домена'
          [, [ @objtype = ] 'тип_объекта' ]
```

Аргумент objtype указывает на тип объекта, который будет переименован и может принимать одно из следующих значений.

COLUMN – поле, которое будет переименовано.

DATABASE – пользовательская база данных. Данный тип необходим при переименовании базы данных.

INDEX – пользовательский индекс.

OBJECT – элемент типа, который отслеживается в списке каталогов sys.objects. Например, значение OBJECT может быть использовано для переименования объектов с ограничениями (CHECK, FOREIGN KEY, PRIMARY / UNIQUE KEY), пользовательских таблиц и правил.

USERDATATYPE – тип данных домена или пользовательские типы CLR

Например:

```
exec sp_rename 'd_salary', 'd_s'
```

4. К уже существующему домену можно привязать или отсоединить существующие правила одним из нижеописанных способов.

```
-- привязать правило к объекту БД
sp_bindrule [ @rulename = ] 'название_правила',
            [ @objname = ] {'домен' | 'таблица.поле' }
            [, [ @futureonly = ] 'futureonly' ]
-- отсоединить правило от объекта БД
sp_unbindrule [ @objname = ] {'домен' | 'таблица.поле' }
              [, [ @futureonly = ] 'futureonly' ]
```

Если флаг futureonly указывается при привязке правила к домену, предполагается наследование нового правила только определенным и существующим полем типа

домена. Если данный аргумент имеет значение NULL, то есть он не установлен, то новое правило привязывается к произвольному полю типа домена, которое на данный момент не имеет правила, или к тому полю, которое использует существующее правило домена.

Аргумент futureonly может указываться и при отсоединении правила. В таком случае предполагается, что существующие поля этого типа данных не будут терять заданного правила.

5. Присоединить или отсоединить существующие значения по умолчанию можно с помощью следующих процедур.

```
-- привязать значение по умолчанию к объекту БД
 sp_bindefault [ @defname = ] 'имя_значения_по_умолчанию',
               [ @objname = ] {'домен' | 'таблица.поле'}
               [,[ @futureonly = ] 'futureonly' ]
-- отсоединить значение по умолчанию от объекта БД
 sp_unbindefault [ @objname = ] {'домен' | 'таблица.поле'
 }
                 [, [ @futureonly = ] 'futureonly' ]
```

Например:

```
--создаем домен и таблицу, которые будут содержать
 знаниние по умолчанию
 exec sp_addtype d_telephone, 'varchar(24)', 'NOT NULL'
 create table Test
 (
   id  int not null primary key,
   age int not null
 );
 go
--создаем значение по умолчанию
 create default def_char6 as 'Ivanko'
```

```
create default def_int as 5
go
-- привязываем к необходимым полям и доменам значение по
умолчанию
--exec sp_bindefault @defname = def_char6, @objname =
'd_telephone'
exec sp_bindefault def_char6, 'd_telephone'
exec sp_bindefault def_int, 'Test.age'
go
--создаем правила
create rule rule_int as @ivar >= 0
create rule rule_char6 as @name like 'A%' or @name like
'B%'
go
-- привязываем к необходимым полям и доменам правила
exec sp_bindrule 'rule_char6', 'd_telephone'
exec sp_bindrule 'rule_int', 'Test.age'
-- правило привязывается к домену d_telephone,
-- но на существующие поля такого типа это не влияет
-- exec sp_bindrule rule_char6, 'd_telephone',
'futureonly'
go
-- отменяем привязку правил ко всем будущим полям типа
домена.
-- для всех поточних полей типа rule_int правила
"остаются в силе"
exec sp_unbindrule 'rule_int', 'futureonly'
```

4. Таблицы. Основы построения таблиц средствами SQL

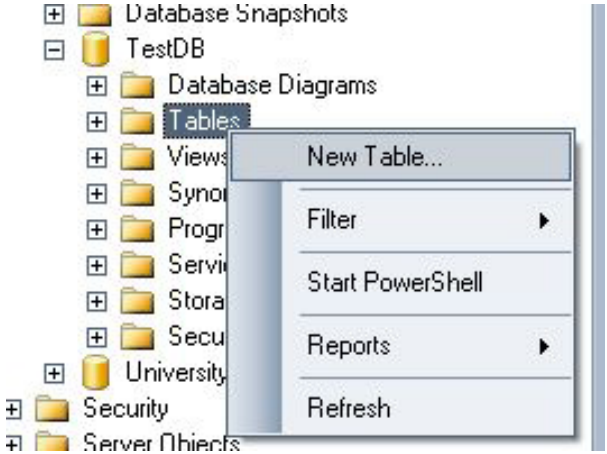
4.1. Средствами MS SQL Server

SQL Server – это реляционная СУБД. Итак, данные в ней представлены в виде таблиц. Все правила создания и нормализации таблиц, которые вы учили на предыдущих курсах баз данных, сохраняются и в SQL Server. Существует также два способа работы с таблицами:

1. Средствами Management Studio;
2. С помощью операторов языка запросов SQL.

Чтобы наш рассказ был полным, рассмотрим оба способа.

Для создания таблицы средствами Management Studio, необходимо в контекстном меню папки "Tables" выбрать пункт "New Table".



Перед вами появится окно конструктора таблиц, в котором Вам необходимо заполнить данные про новую таблицу:

- **Column name** – имя поля создаваемой таблицы.
- **Data type** – тип данных для текущего поля. Тип поля по умолчанию nchar (10).
- **Allow NULLs** – допускаются ли пустые (NULL) поля, то есть, являетмя ли оно обязательным. По умолчанию флажок не установлен, то есть поле является обязательным для заполнения.

Кроме того, каждое поле может иметь и другие расширенные характеристики, которые зависят от его типа данных. К основным дополнительным **характеристикам относятся:**

- **Length** – количество символов (если поле текстовое) или количество необходимых байт (если поле числовое или битное)
- **Default Value or Binding** – значение по умолчанию;
- **Description** – описание создаваемого поля;

- **Precision** – общее количество разрядов для действительных типов данных;
- **Scale** – количество разрядов после десятичной точки для действительных типов данных;
- **Identity Specification** – используется для задания счетчика в поля (обычно используется для первичного ключа). Если флаг Is Identity установлен в Yes, то значение в поле можно приращивать. При этом поле Identity Seed определяет начальное значение счетчика, а Identity Increment величину приращения;
- **Not For Replication** – активно при использовании счетчика в поле. Если поле принимает значение Yes, то счетчик в репликации не используется;
- **Collation** – будет осуществляться сортировка поля при удалении данных;
- **Size** – размер поля в байтах;
- **RowGUID** – используется только с типом данных uniqueidentifier.

Например, создадим маленькую таблицу, которая будет иметь следующую структуру:

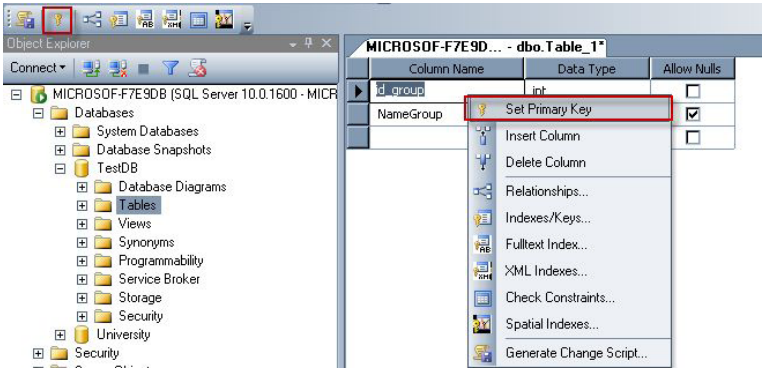
id_group – тип int not null

nameGroup – тип varchar (10) not null

После того, как таблица создана, ее можно сохранить под определенным именем, например, **Groups**.

Как видите, создать оказалось не очень сложно, но это не все. Не следует забывать о том, что согласно правилам нормализации, каждая таблица должна иметь первичный ключ. Для того, чтобы задать первичный ключ определенном полю нашей вновь таблицы, следует выбрать необходимое поле и воспользоваться или кнопкой "Set Primary

Key" на панели инструментов, или пунктом контекстного меню данного поля с аналогичным названием.

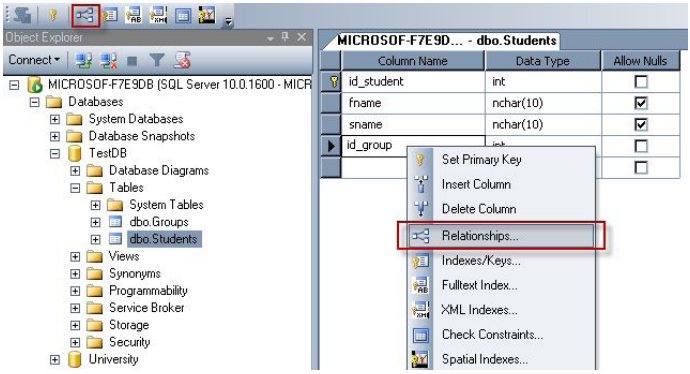


Если после создания таблицы необходимо изменить параметры того или иного поля, нужно в контекстном меню таблицы, параметры которой потребует модификации, выбрать пункт меню "**Design**". Он откроет уже знакомый конструктор таблицы.

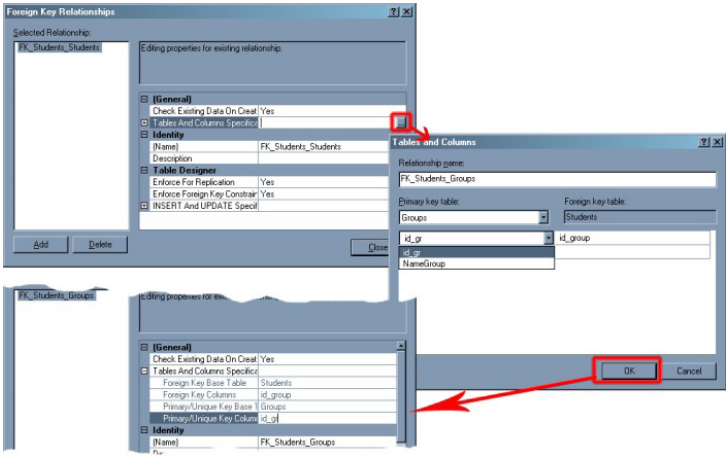
Идем дальше. Как задать первичный ключ таблицы мы уже знаем, но кроме первичных ключей, существуют внешние (вторичные) ключи, которые позволяют связать две таблицы между собой. Чтобы лучше понять, как это можно сделать, создадим еще одну таблицу **Students**, которая будет содержать определенную информацию о студенте. При этом, позволим каждому студенту учиться в разных группах, данные о которых хранятся в уже созданной таблице **Groups**. Итак, таблица **Students** будет следующую структуру:

- id_student – тип int not null;
- fname – тип varchar (10) null;
- sname – тип varchar (10) not null;
- id_group – тип int null.

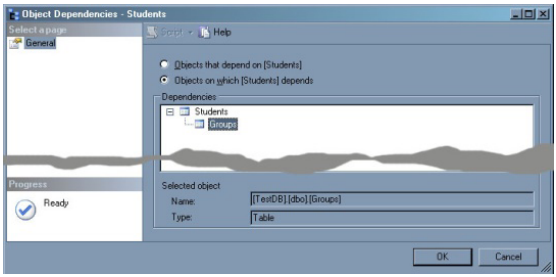
Теперь свяжем таблицы Students и Groups по полю id_group. Для этого делаем это поле активным и вызываем конструктор создания связей или с помощью кнопки "**Relationships**" на панели инструментов, либо с помощью пункта контекстного меню с аналогичным названием



Перед Вами появится диалоговое окно, в котором при нажатии на кнопку "**Add**" будет добавлена новая связь. Далее нужно ее откорректировать, как показано на рисунке снизу, после чего сохранить все настройки, закрыв конструктор.



Чтобы просмотреть в любой момент установленные связи, следует выбрать в контекстном меню таблицы Students пункт **"View Dependencies" (Представление зависимостей)**:



Ну и напоследок. Для удаления уже существующей таблицы нужно в том же контекстном меню таблицы выбрать пункт **"Delete" (Удалить)**.

4.2. С помощью SQL запросов

Создание таблиц в SQL Server осуществляется с помощью инструкции SQL **CREATE TABLE**, обобщенный синтаксис которой следующий:

```
CREATE TABLE [имя_БД.][схема.] название_таблицы
( название_поля тип_данных
                    [ FILESTREAM ]
                    [ COLLATE направление_сортировки ]
                    [ SPARSE ]                          /*
разреженное поле */
                    [ NULL | NOT NULL ]            /*
допускается ли NULL значение */
                    [ CONSTRAINT название_конструкции ]
/* название конструкции */
                    [ DEFAULT виражение_константа ]
/* значение по умолчанию */
                    [
                    [ IDENTITY [ ( нач_значения, шаг_
приращения ) ]
```

```
                    [ NOT FOR REPLICATION ]
                    ]
                    [ ROWGUIDCOL ]
                    [ [CONSTRAINT] конструкции_поля [
...n ] ]
  [ ,...n ]
)
[ON { имя_схемы_секции ( имя_поля_секции ) | группа_
файлов | "default"  } ]
[ TEXTIMAGE_ON { группа_файлов | "default" } ]
[ FILESTREAM_ON { имя_схемы_секции | группа_файлов |
"default"  }]
[ WITH ( опции_таблицы [ ,...n ] ) ]
```

Инструкция **ON** указывает, где физически будет сохраняться таблица. Если файловая группа не задана, SQL Server создаст таблицу, используя файловую группу по умолчанию. В случае установления аргумента "имя_схемы_секции", таблица будет разбита на секции, которые будут сохранены в одной или нескольких указанных файловых группах.

Инструкция **TEXTIMAGE_ON** указывает на то, что поля типов text, ntext, image, xml, varchar (max), nvarchar (max), varbinary (max), а также пользовательских типов среды CLR будут храниться в отдельной указанной файловой группе. Понятно, что данный параметр недопустим, если в таблице не существует полей с большими значениями.

Параметр **FILESTREAM_ON** задает файловую группу для данных FILESTREAM в случае, если таблица содержит следующие данные и является секционированной. В таком случае указывается схема секционирования файловых групп файлового потока. Более подробную информацию о параметре FILESTREAM смотрите в разделе

"Проектирование и реализация данных FILESTREAM" документации.

Инструкция **WITH** используется для указания дополнительных характеристик таблицы, например, можно указать поддержку компрессию (сжатия) строк.

Параметр **IDENTITY** используется для установки счетчика для поля (обычно используется для первичного ключа). При этом можно определить начальное значение счетчика и величину приращения, которая может быть как положительным, так и отрицательным числом. По умолчанию начальное значение равно 1, а шаг приращения 1.

Параметр **NOT FOR REPLICATION** может указываться для свойств IDENTITY, а также ограничений FOREIGN KEY и CHECK (см. Информацию об ограничении данного раздела). Если эта инструкция указана для свойства IDENTITY, то значения в полях идентификаторов НЕ приращиваются, если вставка выполняется вследствие репликации.

ROWGUIDCOL указывает на то, что в таблице можно назначить только одно поле типа uniqueidentifier. Это свойство можно установить только полю с типом uniqueidentifier и использовать его не допускается, если уровень совместимости базы данных равен 65 или ниже.

SPARSE указывает на разряженное поле. Разряженные поля появились в SQL Server 2008 и представляют собой обычные поля, которые имеют оптимизированное хранилище для NULL значений. Для разряженных полей нельзя указывать параметр NOT NULL. Благодаря этому компоненту значения NULL больше не занимают физическое пространство, делает управление пустыми данными эффективным. Кроме того, разряженные поля позволяют

создавать объектные модели с большим количеством NULL значений, не занимая при этом много места на диске. Но, несмотря на преимущества таких полей, использовать их следует только в случае, если экономится не менее 20-40% места.

Опция **FILESTREAM** допустима только для полей типа varbinary (max) и указывает на хранилище FILESTREAM для данных BLOB типа varbinary (max). При этом таблица должна содержать поле типа uniqueidentifier с атрибутом ROWGUIDCOL. Это поле не должно допускать NULL значений и должно иметь ограничение UNIQUE или PRIMARY KEY. GUID значение для такого поля задается при вставке данных или ограничением DEFAULT, в котором используется функция NEWID ().

Что касается конструкций, которые указываются в конце объявления поля, то они могут начинаться с ключевого слова CONSTRAINT и указывать на ограничение PRIMARY KEY, NOT NULL, UNIQUE, FOREIGN KEY или CHECK для поля.

Например, создадим сначала простенькую таблицу:

```
use TestDB
go
create table Test1
(
 id_test1   int identity not null,
 fname      varchar(20) default 'Unknown',
 Surname    d_name,                 /*поле типа домена*/
 Salary     money not null,  /*ненулевое поле
с возможностью сохранения денежных величин*/
 percent_   as (Salary * 0.2),      /*расчетное поле*/
 age        int check(age > 20),    /*поле
с ограничением*/
```

```
 test       nvarchar(20) sparse    /*разряженное поле*/
);
```

В следующем примере создадим таблицу, которая поддерживает компрессию строк:

```
create table Test2
(
 id_test2   int identity(1,2) not null,
 literal       nvarchar(20)
)
with (DATA_COMPRESSION = ROW);
```

Для модификации уже созданной таблицы используется оператор ALTER TABLE, обобщенный синтаксис которого следующий.

```
ALTER TABLE [имя_БД.][схема.] название_таблицы
{
    /* смена информации про поле */
    ALTER COLUMN название_поля
    {
        [ схема. ] тип_данных
        [ COLLATE направление_сортировки ]
        [ SPARSE | NULL | NOT NULL ]
        | {ADD | DROP } { ROWGUIDCOL | PERSISTED | NOT
FOR REPLICATION | SPARSE }
    }
    | [ WITH { CHECK | NOCHECK } ]
    /* добавление и уничтожение характеристик полев
и самих полей */
    | ADD  { описание_нового_поля | [ CONSTRAINT ]
конструкция } [ ,...n ]
    | DROP {
         [ CONSTRAINT ] имя_конструкции
          [ WITH (параметры_удаления_кластеризованного_
ограничения [ ,...n ] ) ]
          | COLUMN название_поля
```

```
    } [ ,...n ]
   /* включенные или выключенные конструкции. ALL - все
конструкции. Данный параметр может  использоваться только
для конструкций FOREIGN KEY и CHECK */
    | { CHECK | NOCHECK } CONSTRAINT
        { ALL | имя_конструкции [ ,...n ] }
    /* включенные или выключенные триггери, которые
установлены на таблицю. ALL - все триггеры */
    | { ENABLE | DISABLE } TRIGGER
        { ALL | имя_триггера [ ,...n ] }
    /* разрешено или нет отслеживание изменений в данной
таблице. Чтобы разрешить такое отслеживание, в таблице
должен быть первичный ключ */
    | { ENABLE | DISABLE } CHANGE_TRACKING
        [ WITH ( TRACK_COLUMNS_UPDATED = { ON | OFF } ) ]
    | SWITCH [ PARTITION секции_основной_таблицы ]
        TO [ схема. ] целевая_таблица
        [ PARTITION секции_целевой_таблицы ]
    /* указывает местонахождение хранилища данных
FILESTREAM. "NULL" указывает на удаление всех ссылок на
файловые группы  FILESTREAM для таблицы */
    | SET ( FILESTREAM_ON = { имя_схемы_секции | группа_
файлов | "default"  | "NULL" } )
    /* для перестройки таблицы */
    | REBUILD
      [ [PARTITION = ALL]
        [ WITH ( опции_перестройки [ ,...n ] ) ]
      | [ PARTITION = номер_секции
          [ WITH ( <single_partition_rebuild_option> [
,...n ] ) ]
        ]
      ]
    /* опции для блокировки таблицы*/
    | (SET ( LOCK_ESCALATION = { AUTO | TABLE | DISABLE }
))
```

Опция "WITH CHECK | WITH NOCHECK" указывает на то, удовлетворяют ли данные в таблице недавно

добавленному или повторно включенному ограничению FOREIGN KEY или CHECK. Если ничего другого не указано, то для новых ограничений рекомендуется использовать WITH CHECK, а для повторно включенных ограничений – WITH NOCHECK.

Опция SWITCH позволяет переключить блок данных одним из следующих способов:

- переназначает все табличные данные как секцию в уже существующей секционированной таблице;
- переключает секции из одной секционированной таблицы в другую;
- переключает все данные одной секции секционированной таблицы в уже существующую несекционированную таблицу.

Если основная таблица является секционированной, то необходимо указать аргумент PARTITION после опции SWITCH, если же целевая таблица секционированная, то должен указываться аргумент PARTITION после ее имени. Если происходит переназначение данных таблицы, как секции в уже существующую секционированных таблицу, или переключение секции с одной секционированной таблицы на другую, то целевая секция должна существовать и быть пустой.

Аргументы "секции_основной таблицы" и "секции_целевой_таблицы" являются постоянными выражениями, которые могут ссылаться на переменные, включая домены, и функции. При этом они не могут ссылаться на выражения языка Transact-SQL.

Опция SET для блокировки таблицы может принимать один из следующих параметров, которые указывают на метод блокировки:

- AUTO – позволяет SQL Server Database Engine выбрать гранулярность укрупнения блокировки, которая подходит к данной схемы таблицы;
- TABLE (значение по умолчанию) – укрупнение блокировки будет выполняться на уровне гранулярности таблицы, независимо от того, секционированная таблица или нет;
- DISABLE – запрещает укрупнение блокировки, но не полностью. Например, при сканировании таблицы, не имеет кластеризованного индекса на уровне изоляции SERIALIZABLE, компонент Database Engine должен установить блокировку таблицы для защиты целостности данных.

Опция REBUILD используется для перестройки таблицы. Параметр REBUILD WITH используется для перестройки всей таблицы, включая все секции в секционированной таблице. Для перестройки одной секции в секционированной таблице используйте параметр REBUILD PARTITION. Данный параметр может принимать одно из двух значений:

- PARTITION = ALL – перестраивает все секции при изменении настроек компрессии секций;
- REBUILD WITH (опции_перестройки) – все указанные опции будут использованы для таблицы с кластеризованного индексом. Если в таблице нет кластеризованного индекса, то на структуру кучи влияют только определенные параметры.

Приведем несколько примеров на модификацию структуры таблицы.

```
-- добавляем новое поле NullCol типа nvarchar(20)
 alter table MyTable alter column NullCol nvarchar(20) not
 null
-- добавить дополнительное разряженное поле test
 alter table MyTable
 add test char(100) sparse null;
-- превращаем разряженное поле test в неразряженное
 alter table MyTable
 alter column test drop sparse;
```

Для удаления таблицы существует оператор DROP TABLE.

```
 DROP TABLE [имя_БД.][схема.] название_таблицы [ ,...n ]
-- например
 drop table test1;
 drop table testdb.dbo.test2;
```

Чтобы таблицы базы данных были нормальными формами и подлежали всем правилам нормализации, они должны содержать первичные и внешние ключи. Первичный ключ для определенного поля таблицы можно установить одним из следующих способов:

```
-- 1. Путем установления конструкции для поля при
 первичном создании таблицы:
 create table Table1(id_test1   int identity not null
 primary key,
                     ..............
                     );
-- 2. Путем установления конструкции для поля при
 первичном создании таблицы после объявления всех полей
 create table Table1(id_test1   int identity not null,
                     .............. ,
                     constraint pkTable1 primary key(id_
 test1)
                     );
```

```
-- 3. Путем модификации уже существующей таблицы, то есть
когда таблица создана, а первичный ключ еще не задан
alter table Test1
add constraint pkTable1 primary key (id_test1);
```

Для удаления уже установленного первичного ключа нужно написать следующий запрос на изменение таблицы:

```
alter table Test1
drop constraint pkTable1 primary key(id_test1);
```

Для вставки внешнего ключа следует воспользоваться одним из нижеописанных сценариев:

```
-- 1. Путем установления конструкции для поля при
первичном создании таблицы:
create table test2 (id_test2 integer not null primary
key,
            id_test1 integer not null constraint
fkTest2_1 references Test1 (id_test1));
-- или
create table test2 (id_test2 integer not null primary
key,
                    id_test1 integer not null references
Test1 (id_test1));
-- 2. Путем установления конструкции для поля при
первичном создании таблицы после объявления всех полей
create table test2 ( id_test2   integer not null primary
key,
                     id_test1   integer not null,
                constraint fkTest2_1 foreign key (id_
test2) references Test1 (id_test1))
-- 3. Путем модификации уже существующей таблицы, то есть
когда таблица создана, а вторичный ключ еще не задан
alter table test2
add constraint fkTest2_1 foreign key (id_test2)
references Test1 (id_test1))
```

SQL Server также позволяет выполнять ряд автоматических действий при редактировании или удалении

внешнего ключа, то есть данных, на которые он ссылается. Это позволяет обеспечить целостность данных в базе данных. Для этого используется следующий набор опций внешнего ключа:

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

При этом возможны следующие варианты:

- **No Action** (по умолчанию) – невозможно изменить или удалить значение в главной таблице (первичный ключ), пока существуют соответствующие значения в подчиненной таблице (внешний ключ);
- **Cascade** – при изменении или удалении значений в связанном поле (первичный ключ) в главной таблице изменяются или уничтожаются (соответственно) значения в связанном поле из подчиненной таблицы (внешний ключ);
- **Set Null** – значение поля, которое является внешним ключом устанавливается в NULL;
- **Set Default** – значение поля, которое является внешним ключом, устанавливается в значение по умолчанию, которое должно быть в списке значений первичного ключа главной таблицы.

Например, обеспечим целостность данных, на которые ссылается внешний ключ таблицы test2.

```
create table test2 ( id_test2   integer not null primary key,
                     id_test1   integer not null,
                constraint fkTest2_1 foreign key (id_test2)
references Test1 (id_test1)
on delete no action
on update cascade )
```

Вы можете рассмотреть еще несколько примеров использования оператора CREATE TABLE для создания таблиц в файле University.sql текущей директории.

5. Схемы

В SQL Server выделяют такое понятие как схемы, которое созвучно с пространствами имен, в пределах которых могут находиться таблицы, представления, хранимые процедуры и триггеры, и другие объекты базы данных. Начиная с версии MS SQL Server 2005, схемы реализованы в соответствии со стандартом ANSI и представляют собой коллекции объектов баз данных, тогда как в ранних версиях объекты находились в пределах их владельца. Имя пользователя теперь не является частью имени объекта, поэтому имена пользователей базы данных можно изменять и даже удалять, не внося при этом изменений в приложение. В каждой схемы есть свой владелец – пользователь или роль, и если их удалить, владение схеме следует передать другому пользователю или роли.

Для создания схем используется оператор CREATE SCHEMA, сокращенный синтаксис которого выглядит следующим образом:

```
CREATE SCHEMA { название_схемы | AUTHORIZATION
собственник
                | название_схемы AUTHORIZATION
собственник }
              [ элемент_схемы [ ...n ] ]
-- элемент схемы описывается:
{ таблица | представление
  объявление_GRANT | объявление_REVOKE | объявление_DANY
}
```

Как видно из инструкции, при создании схемы можно ей указать конкретного владельца или оставить владельцем текущего пользователя, который является ее создателем. Инструкции, содержащие CREATE SCHEMA AUTHORIZATION, то есть не указывают имя, разрешены только для обратной совместимости.

После указанного имени и владельца, можно осуществить перечень элементов схемы, то есть перечень таблиц и других объектов, которые будут созданы внутри схемы. Этот перечень задается инструкциями CREATE TABLE, CREATE VIEW, GRANT, REVOKE или DENY. Последние три инструкции предоставляют, отменяют или запрещают права доступа на защищенные объекты, за исключением вновь схемы.

После создания, информация о схеме заносится в представление каталога sys.schemas.

Приведем несколько примеров:

```
-- создается схема, которая принадлежит текущему
 пользователю
 create schema mgmt;
-- создается схема people, которая принадлежит
 пользователю Pupkin
 create schema people authorization Pupkin;
-- создается схема people, которая принадлежит
 пользователю Pupkin и содержит таблицу test Table.
 Инструкция также дает право на SELECT для пользователя
 Jackson и запрещает SELECT для Voldemar.
-- Примечание! и схема people и таблица Test Table
 создаются в одной инструкции
 create schema people authorization Pupkin
     create table testTable (id int not null, number int)
     grant select to Jeckson
     deny select to Volydemar;
```

Назначать таблицу (или другой объект) определенной схеме не обязательно при ее непосредственном создании. Гораздо удобнее указать область видение объекта при создании самого объекта, то есть таблицы, указав перед ее именем название схемы. К примеру:

```
create table people.testTable (id int not null, number int);
go
select *
from people.testTable;
go
```

Чтобы изменить схему, используется инструкция ALTER SCHEMA:

```
ALTER SCHEMA название_схемы
      TRANSFER [ тип_сущности :: ] имя_объекта_для_
перемещения
```

В инструкции alter schema необходимо обязательно указать название новой схемы и имя объекта для перемещения (как правило, с указанием имени старой схемы). В случае необходимости можно указать тип перемищаемои сущности, который может принимать одно из следующих значений:

- object (по умолчанию);
- type;
- xml_schema_collection.

Например:

```
-- переместить таблицу Address из схемы People в схему
Resources
alter schema Resources transfer People.Address;
-- переместить тип Test Type из схемы People в схему
Resources
alter schema Resources transfer type::People.TestType ;
```

Для удаления существует оператор DROP SCHEMA:

```
DROP SCHEMA название_схемы [, ...n ]
-- например
drop schema people;
```

6. Операторы вставки, модификации и удаления таблиц

С созданием базы данных, таблиц, схем, в которых можно размещать таблицы и другие объекты, мы разобрались. Осталось разобраться с тем, как добавлять, удалять и обновлять данные в базе данных. Средствами Management Studio, начиная с версии SQL Server 2008 это сделать невозможно. Осуществить такие действия можно только средствами языка структурированных запросов SQL, в которой для этого используется три **оператора**:

1. Вставки записей – INSERT.
2. Удаление записей в таблице – DELETE.
3. Изменения значений в существующих записях таблицы – UPDATE.

Итак, оператор INSERT предназначен для добавления записей в таблицу или представление. Различные варианты оператора INSERT позволяют добавлять в таблицу более одной записи методом считывания данных из другой таблицы или представления, а также при выполнении хранимой процедуры или функции. В каждом из перечисленных случаев необходимо учитывать структуру таблицы:

- количество полей;
- тип данных каждого поля;

- имена полей, в которые заносятся данные;
- ограничения и свойства полей.

```
Упрощенный синтаксис оператора INSERT следующий:
INSERT
    [ TOP (кількість) [ PERCENT ] ] /*количество или
процент случайных записей */
    [ INTO ]
    { таблица | представления }
{
    [ ( список_полей ) ]
    { VALUES ( { DEFAULT | NULL | значение } [ ,...n ] )
[ ,...n ]
      | инструкция_SELECT
      | инструкция_EXECUTE
      | DEFAULT VALUES      /*заполняет строку значениями
по умолчанию */
    }
}
```

Согласно вышеописанному синтаксису в указанную таблицу (представление) вставляется запись со значениями полей, указанными в перечне фразы VALUES (значение), причем 1-е значение соответствует 1-му полю в списке полей (поля, не указанные в списке, заполняются NULL-значениями или соответствующими значениями по умолчанию). Если в списке VALUES указаны все поля модифицируемой таблицы и порядок их перечисления соответствует порядку полей в описании таблицы, то список полей при INTO можно проигнорировать.

Кроме того, таблицу можно заполнить результирующими значениями выборки SELECT или данными, которые возвращаются оператором EXECUTE.

Например, добавим в таблицу Students нового студента:

```
-- данные добавляются в таблицу в соответствии с их
физическим порядком
INSERT INTO Students
VALUES (1, 'Леся', 'Поваренко', 1);
- данные добавляются только в определенные поля,
а недостающие в списке наполняются NULL значениями,
значениями по умолчанию, а для полей с типом IDENTITY или
rowversion (timestamp) новые значения будут сгенерированы
самостоятельно
INSERT INTO Students(fname, sname, id_group)
VALUES ('Вова', 'Писаренко', 2),
       ('Олег', DEFAULT, 2);
-- добавить новую строку заполнен значение по умолчанию
INSERT INTO Students
DEFAULT VALUES;
```

Оператор DELETE предназначен для удаления определенного количества полей из таблицы или представления и имеет следующий сокращенный синтаксис:

```
DELETE
    [ TOP (количество) [ PERCENT ] ] /*количество или
процент первых записей */
    [ FROM ]
    { таблица | представления }
    [ FROM исходная_таблица [ ,...n ] ]
    [ WHERE { условие
          | [ CURRENT OF /*удаление с текущей позиции */
                { [ GLOBAL ] имя_курсора } /*
с позиции, на которую указывает курсор */
                ]
           }
    ]
```

Опция "FROM исходная_таблица" – это расширение языка Transact-SQL для инструкции DELETE, которое позволяет задать данные с дополнительной таблицы и удалять соответствующие строки из таблицы в первом

описании оператора FROM. Такое расширение может использоваться вместо вложенного запроса в параметре WHERE для указания строк, которые удаляются.

Например:

```
-- удалить все записи таблицы Students
 delete
 from Students;
-- удалить из базы данных студентов с именем Вова
 delete
 from Students
 where fname = 'Вова';
```

Следует также сказать, что для удаления данных из таблицы кроме оператора DELETE можно использовать оператор TRUNCATE TABLE, который уничтожает или все данные из таблицы, или же те, которые соответствуют указанному условию, но структуру оставляет неизменной. Но, в отличие от DELETE, этот оператор не возвращает сообщение о количестве удаленных записей и переустанавливает значение поля типа IDENTITY заново.

Например, очистим таблицу Groups, используя оператор TRUNCATE TABLE:

```
 TRUNCATE TABLE [имя_БД.][схема.] название_таблицы
-- например
 truncate table Groups;
```

Третий оператор – это оператор обновления данных UPDATE. Он позволяет изменить значение поля в середине существующих записей.

```
 UPDATE таблица | представление
 SET  название_поля = { значение | NULL | выражение |
(оператор_SELECT) }
```

```
[WHERE условие];
UPDATE
    [ TOP (количество) [ PERCENT ] ] /* количество или
процент первых записей */
    { таблица | представление }
    /* указываем, что необходимо обновить */
    SET
        { название_поля = { выражение | DEFAULT | NULL }
          | { пользовательский_тип.{ { свойство =
выражение | название_поля = выражение }
                                    | нестатический_метод
( аргумент [ ,...n ] )
                                }
            }
          | название_поля { .WRITE ( выражение , @Offset
, @Length ) }
          | @переменная = выражение
          | @ переменная = поле = выражение /* переменной
и полю присваивается одинаковое значение */
          | название_поля { += | -= | *= | /= | %= | &= |
^= | |= } выражение
          | @переменная { += | -= | *= | /= | %= | &= |
^= | |= } выражение
          | @переменная = поле { += | -= | *= | /= | %= |
&= | ^= | |= } выражение
        } [ ,...n ]
    [ FROM исходная_таблица [ ,...n ] ]
    [ WHERE { условие
              | [ CURRENT OF /* удаление с текущей
позиции */
                  { [ GLOBAL ] имя_курсора } /*
с позиции, на которую указывает курсор */
                ]
            }
    ]
```

Опция .WRITE (выражение, @Offset, @Length) указывает на то, что должен быть изменен раздел значения "название_поля". Аргумент @Length – это длина раздела

в поле, начиная с @Offset, который заменяется на указанное выражение. Аргумент @Offset отсчитывается с нуля, имеет тип bigint и не может быть отрицательным числом. Данную опцию можно указывать только для полей типа varchar (max), nvarchar (max) или varbinary (max). При указании поля нельзя указывать имени таблицы.

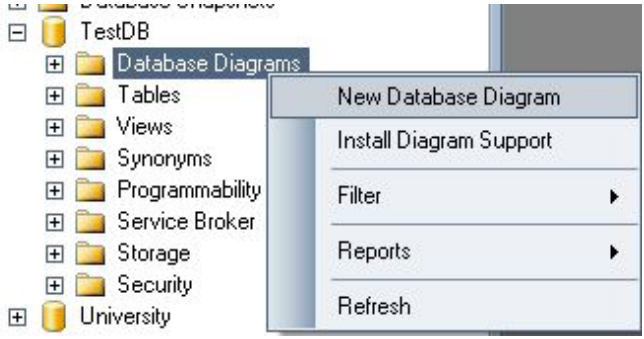
Например:

```
-- заменить имя студента с именем Вова на Вован
 update Students
 set fname = 'Вова'
 where fname = 'Вован';
-- снижаем текущую цену на овсяное печенье на 20%
 и устанавливаем новое значение скидки
 update Product
 set price = price * 0.2,
     discount = 0.2
 where name = 'Овсяное печенье';
```

7. Диаграммы базы данных

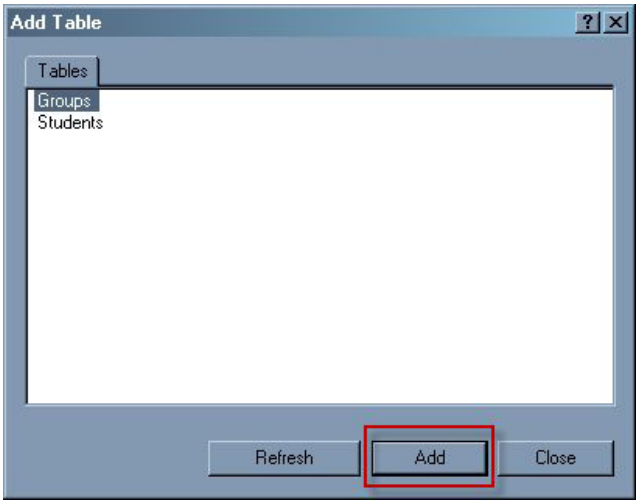
В MS Access у нас была возможность внешнего представления базы данных со всеми ее таблицами и связями. Осуществлялось это с помощью Схемы данных. В SQL Server также есть возможность создания такой схемы данных. Реализуется путем построения диаграмм. Причем диаграмм может быть несколько в одной базе данных и каждая из них может содержать свое внешнее представление тех или иных взаимосвязей в текущей базе данных. Диаграммы является объектом базы данных и размещаются они в папке "Database Diagrams" (диаграммы базы данных).

Для создания диаграммы нужно в контекстном меню папки "Database Diagrams" выбрать пункт "New Database Diagram":

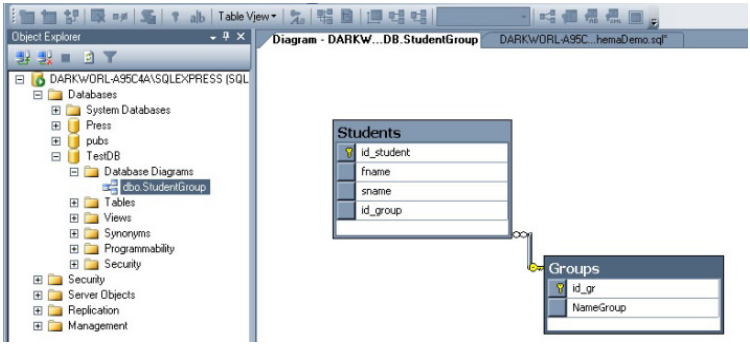


После этого перед Вами появится диалоговое окно, в котором спросят, действительно ли Вы хотите создать

диаграмму. После подтверждения своих действий, если Вы, конечно, уверены, появится окно со списком таблиц на представлений, которые необходимо включить в текущую диаграмму зависимостей:



После нажатия кнопки "Add" будет построена диаграмма, которую можно сохранить для будущих поколений. В нашем случае она будет иметь следующий вид:



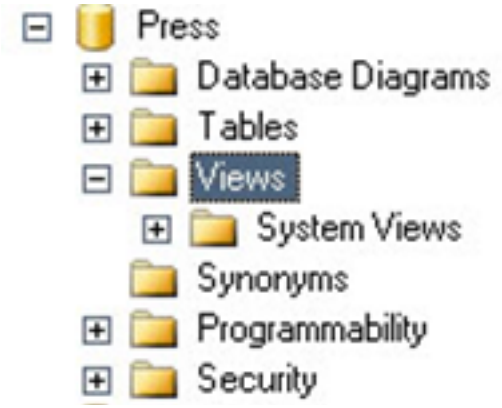
8. Представления в MS SQL Server

Представление (view) – это объект БД, который имеет внешний вид таблицы, но в отличие от нее не имеет своих собственных данных. Представление лишь предоставляет доступ к данным одной или нескольких таблиц, на которых оно базируется. С помощью представления можно осуществлять контроль данных, вводимых пользователем, а также упростить работу разработчиков: запросы, которые чаще всего выполняются, можно поместить в представление, чтобы не писать один и тот же код постоянно.

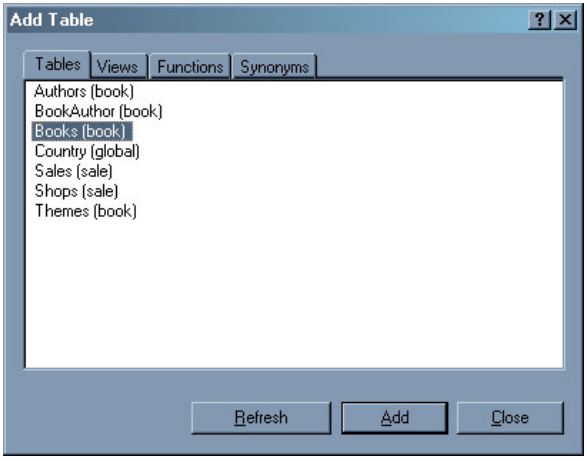
Поскольку представления являются объектом базы данных, то информация о них хранятся в системной таблице **sysobjects**, текст оператора записывается в таблицу **syscomments**, а физическое размещение находится в папке View. Кроме того, автоматически заполняются следующие представления: **sys.views, sys.columns, sys.sql_expression_dependencies** и **sys.sql_modules** (содержится текст инструкции create view).

Представление можно создавать как с помощью Management Studio, так и с помощью SQL. Рассмотрим оба способа.

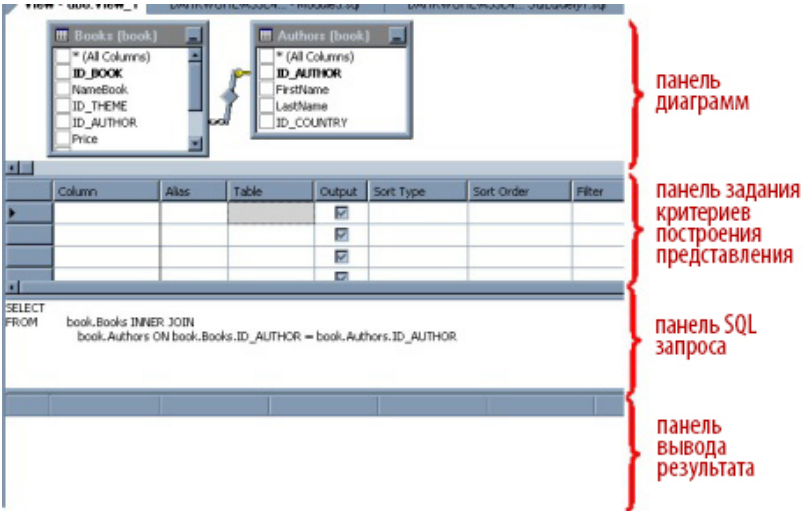
Итак, создадим представление, что позволяет получить информацию о книгах, тираж которых более 3000, и их авторах.



Для того, чтобы создать такой запрос средствами Management Studio, а именно с помощью Create View Wizard, нужно в контекстном меню папки View выбрать пункт **New view**. После этого перед Вами появится окно с предложением выбрать те таблицы, из которых будет браться информация для представления:

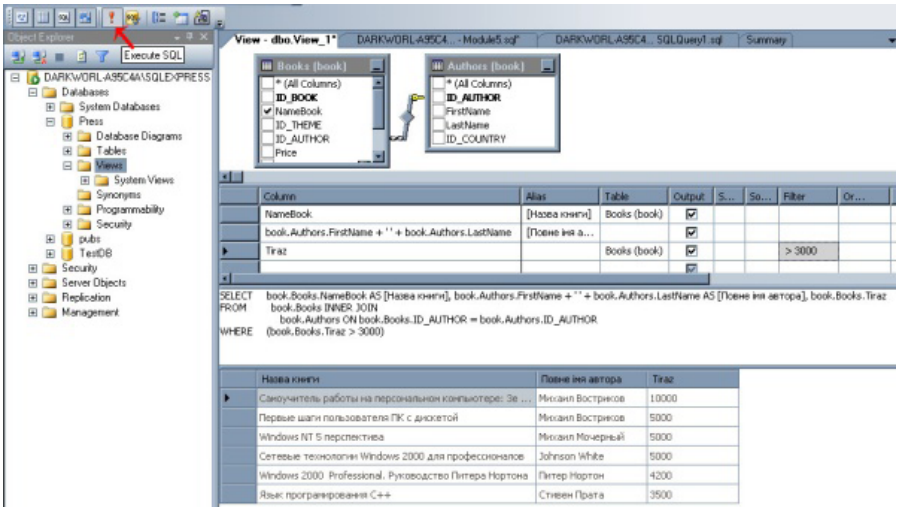


Выбрав таблицы, которые Вам нужны (в нашем случае это Books и Authors) нажимаем Add и Close. После этого появится окно конструктора представлений **View Designer**:



Итак, в поле Table выбираем нужные таблицы, в поле Column – необходимые поля этой таблицы для вывода. В поле Alias можно задать псевдонимы на поля и тому подобное.

После проделанной работы и выполнения запроса мы получим следующий результат:



После этого представления можно сохранить.

Для создания представлений средствами SQL используют инструкцию CREATE VIEW, которая имеет следующий синтаксис:

```
CREATE VIEW [схема.] название_представления [ (поле [,...n] ) ]
[ WITH ENCRYPTION | SCHEMABINDING | VIEW_METADATA ]
AS
<оператор SELECT>
[ WITH CHECK OPTION];
```

Расшифруем коротко каждый из параметров данного оператора:

- **WITH ENCRYPTION** – указывает на то, что представление будет сохраняться в системной таблице syscomments в зашифрованном виде. Расшифровать такое представление невозможно, поэтому при выборе данного параметра следует убедиться, что у вас есть его копия;

- **WITH SCHEMABINDING**. Эта опция устанавливает запрет удаления таблиц, представлений или функций, на которые ссылается представление, раньше, чем оно будет удалено;
- **WITH VIEW_METADATA** – указывает на то, что представление в режиме просмотра будет возвращать его метаданные, то есть информацию о его структуре, а не записи таблиц;
- **WITH CHECK OPTION** – создает модифицированное представление, в котором существует возможность запретить выполнение операций INSERT или DELETE, если при этом нарушается условие, заданное в конструкции WHERE.

После ключевого слова AS должен размещаться запрос с использованием оператора SELECT, предназначенный для выбора данных, которые будут включены в результирующее представление. Стоит отметить, что для того, чтобы изменить произвольное представление, необходимо его пересоздать, то есть удалить и снова создать. А при удалении нужно удалить все зависящие от него объекты – триггеры, хранимой процедуры и т.п. Это является одним из основных неудобств при работе с представлениями.

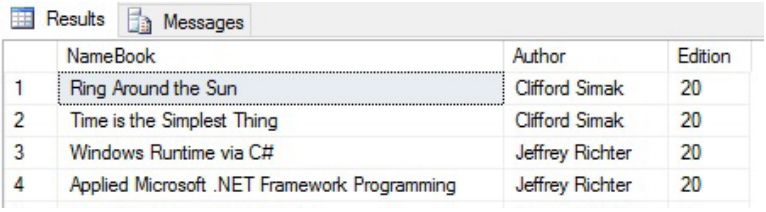
Итак, попробуем написать представление, которое мы создавали с помощью View Designer, используя оператор SQL CREATE VIEW:

```
create view BooksEdition_View2 (NameBook, Author, Edition)
as
select b.NameBook, a.FirstName + ' ' + a.LastName as
'FullName', b. Quantity
from Books b, Authors a
where b.id_author = a.id and b.quantity > 10;
```

Созданное представление можно использовать как любую таблицу базы данных. И чтобы получить из него данные, следует написать простой запрос на выборку с помощью оператора SELECT:

```
select *
from book.BooksEdition_View2;
```

Результат:



	NameBook	Author	Edition
1	Ring Around the Sun	Clifford Simak	20
2	Time is the Simplest Thing	Clifford Simak	20
3	Windows Runtime via C#	Jeffrey Richter	20
4	Applied Microsoft .NET Framework Programming	Jeffrey Richter	20

Но само по себе представление НЕ СОДЕРЖИТ никаких данных. Оно просто ссылается на уже существующие данные, то есть является обычным SELECT запросом, который просто имеет имя. Фактически, когда с помощью запроса вы обращаетесь к представлению, оптимизатор запросов заменяет эту ссылку на описание представления (тело), а затем создает план выполнения.

Представления могут быть основаны на данных из нескольких таблиц и даже на основе других представлений. Также представление могут содержать данные, полученные на основе различных выражений, в том числе и на основе функций агрегирования.

Различают следующие **типы представлений**:

1. **Обычные представления на основе объединений** – это представление на выборку данных.

2. **Модифицированные (обновляемые) преставления** – это представления, которые поддерживают модификацию данных.
3. **Индексированные или материализованные представления** – это представления с использованием кластеризованного индекса. Такие представления позволяют существенно повысить производительность даже очень сложных запросов. Индексированные представления подробнее будут рассмотрены при изучении индексов.
4. **Секционированные представления** – это представления, которые при создании используют оператор UNION ALL для объединения нескольких запросов, которые построены на основе таблиц с одинаковой структурой и хранятся либо в одном экзампляре SQL Server, или в группе автономных экземпляров SQL Server, которые называются федеративными серверами баз данных. Подробнее информацию можно прочитать в разделе "Создание секционированных представлений" MSDN.

Для успешной работы стоит только понять, что в основе представления лежат обычные запросы на выборку. Итак, все, что можно делать с запросами относится и к представлениям. И, если необходимо вывести данные в отсортированном порядке нам не удастся, поскольку сортировать можно только результирующие данные представления:

```
select *
from book.BooksEdition_View2
order by 1;
```

Хотя и здесь существует нюанс: если в операторе SELECT используется опция TOP, тогда инструкция ORDER BY может использоваться при создании представления. То есть следующий запрос будет верным:

```
create view TestView
as
select top (3) b.NameBook, t.NameTheme, b.DateOfPublish
from book.Books b, book.Themes t
where b.ID_THEME = t.ID_THEME order by 1;
```

Несмотря на удобство использования представлений, кроме ограничения на использование выражения ORDER BY существует еще ряд существенных ограничений:

- нельзя использовать в качестве источника данных набор, полученный в результате выполнения хранимых процедур;
- при создании представления оператор SELECT не должен содержать операторы COMPUTE или COMPUTE BY;
- представление не может ссылатся на временные таблицы, поэтому в операторе создания ЗАПРЕЩЕНО использовать оператор SELECT INTO;
- данные, которые используются представлением, не сохраняются отдельно, поэтому при изменении данных представления меняются данные базовых таблиц;
- представление не может ссылаться больше, чем на 1024 поля;
- UNION и UNION ALL недопустимы при формировании представлений.

Для модификации представления используется оператор **ALTER VIEW**:

```
ALTER VIEW [схема.] название_представления [ (поле [,...
n] ) ]
[ WITH ENCRYPTION | SCHEMABINDING | VIEW_METADATA ]
AS
<оператор SELECT>
[ WITH CHECK OPTION];
```

Для того, чтобы удалить представления используется оператор **DROP VIEW**:

```
DROP VIEW название_представления;
```

В начале данного раздела мы говорили, что представления могут быть **модифицированными (обновляемыми, updateable view)**. Что же это за представления? Как видно из самого названия, такие представления позволяют не только читать данные, но и изменять их. Модифицированными (обновляемыми) представления становятся, если при их создании указать инструкцию **WITH CHECK OPTIONS**.

К таким представлениям можно применять команды INSERT, UPDATE и DELETE. Причем они позволяют запретить выполнение операций INSERT или DELETE, если при этом нарушается условие, заданное в конструкции WHERE. Это можно объяснить так: если новая запись, который вставляется пользователем или полученный в результате обновления существующей записи, не удовлетворяет условие запроса, то вставка этой записи будет отменена и возникнет ошибка.

Рассмотрим **условия создания модифицированных представлений**:

- все модификации должны касаться только одной таблицы, то есть модифицированные представления строятся только на однотабличних запросах;
- все изменения должны касаться только полей таблицы, а не производных полей. То есть, нельзя модифицировать поля, полученые:
  - с помощью агрегатной функции: AVG, COUNT, SUM, MIN, MAX, GROUPING, STDEV, STDEVP, VAR и VARP;
  - на основе расчетов с участием других полей или операций над полем, например substring. Поля, сформированные с помощью операторов UNION, UNION ALL, CROSSJOIN, EXCEPT и INTERSECT также считаются расчетными и также не могут обновляться.
- при определении представления нельзя использовать инструкции GROUP BY, HAVING и DISTINCT;
- нельзя использовать опцию TOP вместе с инструкцией WITH CHECK OPTION, даже в подзапросах.

Если указанные ограничения не позволяют модифицировать данные, тогда можно воспользоваться триггерами INSTEAD OF или секционированными представлениями.

Например, необходимо иметь представление, которое будет отображать информацию о книгах, которые имели тираж более 10 экземпляров:

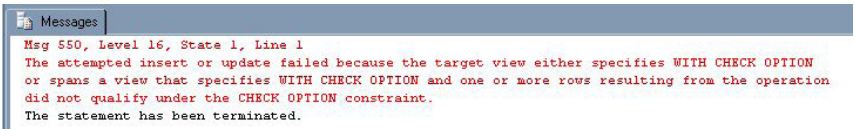
```
create view Books10 (NameBook, Price, Quantity)as
select NameBook, Price, Quantity
from Books
where quantity > 10
with check option;
```

Само по себе модифицированное представление ничем не отличается от обычного, но попробуем проверить данное модифицированное представление на операцию вставки данных.

```
insert into Books5000 (NameTovar, Price, DrawingOfBook)
values ('Microsoft SQL Server 2000 за 21 день', 350.50,
3000);
```

В случае обычного представления, пользователь может добавить такую книгу и ошибка при этом не возникнет. Но данный запрос на вставку является некорректным и приведет к путанице: представление при следующем обращении введенных данных показывать не будет, поскольку они не соответствуют критерию отбора данных.

При введении таких данных в модифицированное представление, сервер сразу выдаст сообщение об ошибке, поскольку данные ввода не соответствуют основному условию отбора (тираж более 10 экземпляров).



Следующий запрос на вставку является корректным и выполняется, поскольку тираж выпуска указывается более 10.

```
Insert into Books10 (NameBook, Price, Quantity)
values
('Modern Web Development', 35, 50);
```

Подытоживая, следует сказать, что хотя модифицированные представления и используются для изменения данных, но для этой цели они практически никогда не используются. Лучше всего для таких действий подходят хранимые процедуры. Кроме того, они гораздо более гибкие.

Урок №1

Работа с таблицами и представлениями в MS SQL Server