Темы урока

Общие команды

В SQL Server есть команды, которые не являются инструкциями Transact-SQL, но распознаются программами, например, SQL Server Management Studio. Эти команды используются для повышения удобочитаемости и упрощения выполнения пакетов и скриптов.

GO

```
GO [count]
```

`GO` информирует программы SQL Server об окончании пакета инструкций Transact-SQL. Пакет, предшествующий команде `GO`, будет выполняться заданное в count раз или единожды, если параметр отсутствует.

Инструкция Transact-SQL не может располагаться на той же строке, что и команда `GO`. Тем не менее строка с командой `GO` может содержать комментарии.

При использовании команды `GO` нужно соблюдать требования, предъявляемые к пакетам. Например, при любом вызове хранимой процедуры после первой инструкции пакета нужно использовать ключевое слово `EXECUTE`. Область видимости локальных (пользовательских) переменных ограничена пакетом, и к ним нельзя обращаться после команды `GO`.

Приложения, основанные на API-интерфейсах ODBC или OLE DB, при попытке выполнить команду `GO` получают уведомление о синтаксической ошибке. **Программы SQL Server никогда не отправляют команду `GO` серверу**!

USE

Объяснить по слайду, показать примеры.

PRINT

Объяснить по слайду, показать примеры. Можно показать вместе с `GO 5`, например,— выведется 5 одинаковых сообщений.

RAISERROR

Объяснить по слайду, показать примеры.

Transact-SQL Live Hero Tour

Что будет происходить дальше?

А дальше вместо переписывания и перечитывания по слайдам документации по синтаксису и возможным конструкциям SQL, я устрою живую демонстрацию работы с несложной базой данных из двух таблиц. От проектирования схемы до выборки данных. По ходу объясняю все использующиеся конструкции.

...и почему?

Ввиду того, что в программе курса это последний урок именно на SQL, думаю, такого рода демонстрация будет наиболее полезна. Таким образом студенты смогут наиболее быстро охватить спектр возможностей и на базе полученных знаний о существующих конструкциях (пользуясь документацией, а не слайдами) проектировать собственные решения.

Создаём и играем с таблицей Category (категории товаров)

Освещаемые темы

- UNIQUEIDENTIFIER
- PRIMARY KEY (ещё раз повторяем)
- UNIQUE (ещё раз повторяем)
- NEWID()
- UPDATE
- WHERE (=/LIKE)
- ORDER BY (ASC/DESC)
- DELETE
- TRUNCATE

Начинаем с простого и идём последовательно, рассказывая по ходу демо нюансы тех или иных запросов.

Код L28_C01_playing_with_categories.sql

```
CREATE DATABASE QueryTests;
GO
USE QueryTests;
GO
-- Создаём таблицу с GUID-полем
-- DROP TABLE dbo.[Category]
CREATE TABLE dbo.[Category] (
       Id UNIQUEIDENTIFIER NOT NULL,
       [Name] VARCHAR(50) NOT NULL,
       CONSTRAINT PK_Category PRIMARY KEY CLUSTERED (Id),
       CONSTRAINT UQ_Category_Name UNIQUE ([Name])
);
GO

-- Вставка значений
INSERT INTO dbo.[Category] (Id, [Name]) VALUES (NEWID(), 'Mobile Phones')
INSERT INTO dbo.[Category] (Id, [Name]) VALUES (NEWID(), 'TV')
-- Смотрим результат
SELECT Id, [Name] FROM dbo.[Category]
GO

-- Изменяем значение поля
UPDATE dbo.[Category]
       SET [Name] = 'Mobile Phone'
WHERE [Name] = 'Mobile Phones'

-- Смотрим результат
SELECT Id, [Name] FROM dbo.[Category]
GO
```

```
-- Попытка повторной вставки значений
INSERT INTO dbo.[Category] (Id, [Name]) VALUES (NEWID(), 'Mobile Phones')
INSERT INTO dbo.[Category] (Id, [Name]) VALUES (NEWID(), 'TV')

-- Смотрим результат подозрительным значениям
SELECT Id, [Name]
FROM dbo.[Category]
WHERE [Name] LIKE 'Mobile%'
ORDER BY [Name] ASC
GO
-- Удаляем лишнюю запись
DELETE FROM dbo.[Category]
WHERE [Name] = 'Mobile Phones'

-- Смотрим результат по всем значениям
SELECT Id, [Name] FROM dbo.[Category]
GO

-- Полностью очищаем таблицу
--TRUNCATE TABLE dbo.[Category]
```

Создаём и играем с таблицей Goods (сами товары)

Начинаем с простого и идём последовательно, рассказывая по ходу демо нюансы тех или иных запросов.

Освещаемые темы

- IDENTITY
- FOREIGN KEY (ещё раз повторяем)
- DECLARE (локальные переменные)
- Присваивание значений переменным
- INSERT (в случае автоинкрементного поля)

Код L28_C02_playing_with_goods.sql

```
USE QueryTests;
GO
-- Создаём таблицу с автоинкрементным полем Id
-- DROP TABLE dbo.Goods
CREATE TABLE dbo.Goods (
	Id INT NOT NULL IDENTITY(1,1),
	CategoryId UNIQUEIDENTIFIER NOT NULL,
	[Name] NVARCHAR(100) NOT NULL,
	CONSTRAINT PK_Goods PRIMARY KEY CLUSTERED (Id),
	-- с точки зрения бизнес-логики ограничение на уникальность
	-- лучше сделать по двум полям, однако это не очень вписывается
	-- в мой пример с откатом транзакции, поэтому я бы предложил
	-- оставить тут только одно поле [Name]
	--CONSTRAINT UQ_Goods_CategoryId_Name UNIQUE (CategoryId, [Name])
	CONSTRAINT UQ_Goods_Name UNIQUE ([Name])
);
GO
```

```
-- Вешаем на неё внешний ключ
ALTER TABLE dbo.[Goods]
       ADD CONSTRAINT FK_Goods_CategoryId FOREIGN KEY (CategoryId)
              REFERENCES dbo.Category(Id);
GO
-- Определяем переменную типа UNIQUEIDENTIFIER
DECLARE @guid AS UNIQUEIDENTIFIER
-- Задаём переменной значение поля CategoryId
-- из таблицы dbo.Category где поле Name равно 'Mobile Phone'
SELECT @guid = Id
FROM dbo.Category AS C
WHERE C.[Name] = 'Mobile Phone'

-- Можем посмотреть значение этой переменной
PRINT @guid

-- GO писать нельзя, так как область видимости переменной @guid на этом завершится
-- Вставляем записи в таблицу с автоинкрементным полем,
-- где CategoryId заполняется из переменной @guid
INSERT INTO dbo.Goods (CategoryId, [Name]) VALUES (@guid, 'iPhone X')
-- системная функция SCOPE_IDENTITY() возвращает последний вставленный
-- в рамках данной сессии идентификаторв в автоинкрементное поле
PRINT 'ID of iPhone X is ' + CONVERT(VARCHAR(15), SCOPE_IDENTITY())

-- Функции CONVERT() и CAST() помогают приводить данные к нужному типу
INSERT INTO dbo.Goods (CategoryId, [Name]) VALUES (@guid, 'Xiaomi Mi 9')
PRINT 'ID of Xiaomi Mi 9 is ' + CAST(SCOPE_IDENTITY() AS VARCHAR(15))

-- Смотрим на содержимое таблицы Goods
SELECT * FROM dbo.Goods
GO
-- Полностью очищаем таблицу
-- (включая текущий счётчик
-- автоинкремента поля Id
--TRUNCATE TABLE dbo.[Goods]
```

Создаём функцию GetCategoryId

Она будет возвращать нам GUID категории по её имени по сложной логике, сначала ищет по точному совпадению, потом выбирает первое попавшееся из поиска по LIKE.

Функция, потому что не сложная и **не меняет данных на сервере**.

Освещаемые темы

- CREATE FUNCTION
- аргументы функции
- RETURNS/RETURN() - тип и значение функции
- IF...ELSE
- BEGIN...END
- TOP 1
- вызов функции через SELECT

Код L28_C03_playing_with_functions.sql

```
USE QueryTests
GO
DROP FUNCTION IF EXISTS [dbo].[GetCategoryId]
GO
CREATE FUNCTION dbo.GetCategoryId (
	@categoryName AS VARCHAR(50) -- like in the field Name of table Category
)
RETURNS UNIQUEIDENTIFIER
AS
BEGIN
	-- объявляем переменную идентификатора
	DECLARE @guid AS UNIQUEIDENTIFIER

	-- пытаемся найти категорию по точному совпадению имени
	SELECT @guid = Id FROM dbo.Category WHERE [Name] = @categoryName

	-- если не нашли (@guid по-прежнему NULL)
	IF (@guid IS NULL)
	BEGIN
		-- ищем по совпадению начальных символов
		SELECT TOP 1 @guid = Id FROM dbo.Category WHERE [Name] LIKE @categoryName + '%'
	END

	-- возвращаем @guid результат
	-- там всё ещё может быть NULL, если ничего не найдено даже
	-- по совпадению начальных символов
	RETURN(@guid)
END
GO

-- проверяем, как работает наша функция на точном соответствии
SELECT dbo.GetCategoryId('TV')
-- проверяем, как работает наша функция на соответствии начальным символам
SELECT dbo.GetCategoryId('T')
-- проверяем, как работает наша функция на заведомо отсутствующем значении
SELECT dbo.GetCategoryId('D')
```

Создаём хранимые процедуры CreateCategory, CreateGoodsItem

- `CreateCategory` будет вставлять новую категорию в таблицу и возвращать нам GUID этой категории.
- `CreateGoodsItem` будет вставлять новый товар для указанной по имени категории. Если категория отсутствует, её также нужно будет вставить. Вставка делается транзакционно. Если вставка не удаётся по каким-то причинам, необходимо откатить транзакцию, в противном случае, подтвердить.

Процедура — меняет данные на сервере (производит вставку в таблицу).

Освещаемые темы

- CREATE STORED PROCEDURE
- аргументы ХП
- возвращаемое значение ХП
- SET XACT_ABORT ON
- BEGIN TRY/BEGIN CATCH
- XACT_STATE()
- BEGIN TRANSACTION/COMMIT/ROLLBACK
- EXECUTE

Код L28_C04_playing_with_stored_procedures.sql

```
USE QueryTests
GO
DROP PROCEDURE IF EXISTS [dbo].[CreateCategory]
GO
-- SP будет вставлять новую категорию в таблицу и возвращать нам GUID этой категории.
CREATE PROCEDURE dbo.CreateCategory (
	@categoryName AS VARCHAR(50), -- like in the field Name of table Category
	@guid AS UNIQUEIDENTIFIER OUTPUT
)
AS
BEGIN
	-- отключене сообщение о количестве изменённых записей
	-- (best practice)
	SET NOCOUNT ON

	-- объявляем переменную идентификатора
	DECLARE @tempGuid AS UNIQUEIDENTIFIER
	SET @tempGuid = NEWID()

	INSERT INTO dbo.Category (Id, [Name])
	VALUES (@tempGuid, @categoryName)

	SET @guid = @tempGuid
END
GO
DECLARE @guid AS UNIQUEIDENTIFIER
-- проверяем, как работает наша функция на точном соответствии
EXECUTE dbo.CreateCategory @categoryName  = 'TEST', @guid = @guid OUTPUT
SELECT @guid AS Id
DELETE FROM dbo.Category WHERE [Name] = 'TEST'
GO
```

```
DROP PROCEDURE IF EXISTS [dbo].[CreateGoodsItem]
GO
CREATE PROCEDURE dbo.CreateGoodsItem (
	@categoryName AS VARCHAR(50),
	@goodsItemName AS NVARCHAR(100),
	@goodsItemId AS INT OUTPUT)
AS
BEGIN
	SET NOCOUNT ON
	-- объявляем переменную идентификатора категории
	DECLARE @guid AS UNIQUEIDENTIFIER
	-- пытаемся её найти с помощью нашей функции
	SELECT @guid = dbo.GetCategoryId(@categoryName)
	-- включаем автоматический откат текущей транзакции,
	-- если внутри транзакции происходит ошибка выполнения.
	SET XACT_ABORT ON;
	-- начинаем транзакцию
	BEGIN TRANSACTION
	-- начинаем блок try
	BEGIN TRY
		-- если не получилось найти категорию по имени, пытаемся её создать
		IF (@guid IS NULL)
			EXECUTE dbo.CreateCategory @categoryName, @guid OUTPUT
		-- далее пробем вставить товар с указанной категорией.
		-- тут может возникнуть проблема со вставкой нового товара,
		-- например, нарушено ограничение на уникальность имени товара.
		-- в таком случае создание новой категории (для этого товара),
		-- выполненное выше с помощью dbo.CreateCategory, имело бы смысл отменить.
		INSERT INTO dbo.Goods(CategoryId, [Name])
		VALUES (@guid, @goodsItemName)
		SET @goodsItemId = SCOPE_IDENTITY()
	END TRY
	BEGIN CATCH
		-- проверяем XACT_STATE()
		-- если 1, транзакцию можно закоммитить.
		-- если -1, транзакцию закоммитить нельзя и она должна быть откачена.
		-- XACT_STATE = 0 означает, что мы вообще не в транзакции
		--     и операции commit или rollback приведут к ошибке.
		IF (XACT_STATE()) = -1
			PRINT 'Rolling back the transaction'
		IF (XACT_STATE()) = -1
			ROLLBACK TRANSACTION;
		THROW;
	END CATCH
	-- в этой точке мы не знаем, пришли ли мы сюда
	-- после ошибки и отката транзакции или всё прошло гладко.
	-- Индикатором может быть наличие транзакции, которую можно
	-- завершить. Если такая есть - (XACT_STATE()) = 1 - то
	-- завершаем её.
	IF (XACT_STATE()) = 1
		COMMIT TRANSACTION;
	PRINT 'Transaction has been commited'
END
GO
```

```
-- проверяем, как работает наша процедура на данных для успешной вставки
-- когда не нужно создавать новую категорию
DELETE FROM dbo.Goods WHERE [Name] = 'Epson 200'
GO
EXECUTE dbo.CreateGoodsItem @categoryName = 'Print', @goodsItemName = 'Epson 200'
GO
SELECT * FROM dbo.Category
SELECT * FROM dbo.Goods
GO

-- проверяем, как работает наша процедура на данных для успешной вставки
-- когда нужно создавать новую категорию
DELETE FROM dbo.Goods WHERE [Name] = 'Lenovo Tab'
DELETE FROM dbo.Category WHERE [Name] = 'Tablet'
GO
EXECUTE dbo.CreateGoodsItem @categoryName = 'Tablet', @goodsItemName = 'Lenovo Tab'
GO
SELECT * FROM dbo.Category
SELECT * FROM dbo.Goods
GO

-- проверяем, как работает наша процедура на данных для НЕУСПЕШНОЙ вставки
-- когда нужно создавать новую категорию, но при заведении товара возникает
-- нарушение ограничения на уникальность имени товара
EXECUTE dbo.CreateGoodsItem @categoryName = 'Extra Tablet', @goodsItemName = 'Lenovo Tab'
GO
SELECT * FROM dbo.Category
SELECT * FROM dbo.Goods
GO
```

Играем с различными типами объединений

Код L28_C05_playing_with_joins.sql

```
USE QueryTests
GO
-- Чтобы было интересно сначала добавим одну пустую категорию
EXECUTE dbo.CreateCategory 'Other', NULL
SELECT * FROM dbo.Category
SELECT * FROM dbo.Goods

-- INNER JOIN
SELECT *
FROM dbo.Category
INNER JOIN dbo.Goods
       ON dbo.Goods.CategoryId = dbo.Category.Id
--
SELECT dbo.Goods.[Name], dbo.Category.[Name]
FROM dbo.Category
INNER JOIN dbo.Goods
       ON dbo.Goods.CategoryId = dbo.Category.Id
ORDER BY 2, 1
```

```sql
SELECT
       C.[Name],
       G.[Name],
       G.Id
FROM dbo.Category AS C WITH(NOLOCK)
INNER JOIN dbo.Goods AS G WITH(NOLOCK)
       ON G.CategoryId = C.Id
ORDER BY 1, 2

-- LEFT JOIN

-- выбрать все(!) категории и соответствующие им товары (если такие есть)
SELECT *
FROM dbo.Category AS C
LEFT JOIN dbo.Goods AS G
       ON G.CategoryId = C.Id

-- выбрать такие категории у которых нет ни одного товары
SELECT *
FROM dbo.Category AS C
LEFT JOIN dbo.Goods AS G
       ON G.CategoryId = C.Id
WHERE G.CategoryId IS NULL

-- RIGHT JOIN
-- поскольку у таблицы Goods стоит внешний ключ на таблицу Category наш RIGHT JOIN
-- по факту вернёт то же самое, что и INNER JOIN
SELECT *
FROM dbo.Category AS C
RIGHT JOIN dbo.Goods AS G
       ON G.CategoryId = C.Id

-- FULL JOIN
-- поскольку у таблицы Goods стоит внешний ключ на таблицу Category наш FULL JOIN
-- по факту вернёт то же самое, что и LEFT JOIN
SELECT *
FROM dbo.Category AS C
FULL JOIN dbo.Goods AS G
       ON G.CategoryId = C.Id

-- Статистические функции
-- Подсчёт количества
SELECT COUNT(Id)
FROM dbo.Goods
```

Настройки кодогенерации MS SQL Server Management Studio

Options > SQL Server Object Explorer > Scripting

Например, мне нравится когда скрипт генерируется с включённой опцией **Check for object existence**.

Домашнее задание

Спроектировать и написать SQL-запросы для создания схемы отношений хранилища нашего чат-бота.