**Создание внешнего ключа(**

DBCC CHECKIDENT("Customers",RESEED,0); -- Обновление значение IDENTITY = 0

Связь один-к-одному делается с указанием unique и используется для разбивки слишком большой таблицы

**)**

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- Целые типы данных в T-SQL

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-------------------------------------- Bit ----------------------------------

-- Целое без знака - Bit. Диапазон значений (0 или 1).

-- При попытке присвоения значения отличного от 0 и 1,

-- это значение будет неявно преобразовано в 1.

PRINT 'Bit';

DECLARE @bit Bit = 0;

PRINT @bit;

SET @bit = 1;

PRINT @bit;

SET @bit = -4; -- (-4) будет неявно преобразовано в 1.

PRINT @bit; -- bit = 1

------------------------------------- TinyInt -------------------------------

-- Целое без знака TinyInt. Диапазон значений от 0 до 255.

PRINT 'TinyInt';

DECLARE @tinyint TinyInt = 0;

PRINT @tinyint;

SET @tinyint = 25;

PRINT @tinyint;

SET @tinyint = 256; -- Ошибка присвоения: Арифметическое переполнение.

PRINT @tinyint; -- Значение переменной @tinyint останется прежним.

----------------------------------- SmallInt --------------------------------

-- Целое со знаком SmallInt. Диапазон значений от -32 768 до 32 767.

PRINT 'SmallInt';

DECLARE @smallint SmallInt = 32767;

PRINT @smallint;

SET @smallint = -32768;

PRINT @smallint;

-------------------------------------- Int ----------------------------------

-- Целое со знаком Int. Диапазон значений от -2 147 483 648 до 2 147 483 647.

PRINT 'Int'

DECLARE @int Int = 2147483647;

PRINT @int;

SET @int = -2147483648;

PRINT @int;

------------------------------------ BigInt ----------------------------------

-- Целое со знаком BigInt. Диапазон значений от -(2^63) до 2^63-1.

PRINT 'Bigint';

DECLARE @bigint Bigint = 9223372036854775807;

PRINT @bigint;

SET @bigint = -9223372036854775808;

PRINT @bigint;

------------------------------------------------------------------------------

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- Вещественные типы данных в T-SQL

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

------------------------------------- Float ---------------------------------

-- Вещественное со знаком, с плавающей точкой Float.

-- Диапазон значений от -1.79Е+308 до 1.79Е+308.

-- Числовой тип данных с плавающей запятой записывается как Float(n),

-- где n - определяет точность(по умолчанию n = 53)

-- Для типа Float имеется только два вида точности: 7 и 15 знаков после запятой.

-- 7 знаков - при n, в диапазоне от 1 до 24; (выделение 4 байт)

-- 15 знаков - при n, в диапазоне от 25 до 53; (выделение 8 байт)

PRINT 'Float'

DECLARE @float float(24) = 1214782.123;

PRINT @float;

SET @float = 2147482435234412412.4321523598746737654;

PRINT @float;

------------------------------------ Decimal --------------------------------

-- Вещественное со знаком, с фиксированной точкой Decimal (Numeric).

-- Диапазон значений от -10^38+1 до 10^38-1.

-- Числовой тип данных с фиксированной точкой Decimal записывается как Decimal(p, s),

-- где p - определяет точность - максимальное количество знаков из которых состоит полное число

-- (по умолчанию p = 18, максимальное значение p = 38, минимальное значение p = 1),

-- a s - определяет масштаб - максимальное количество знаков после запятой.

PRINT 'Decimal / Numeric'

DECLARE @decimal Decimal(5, 3); -- p = 5, s = 3( не больше p-1)

SET @decimal = 1.42;

PRINT @decimal;

SET @decimal = 2.234654; -- Произойдет округление дробной части.

PRINT @decimal;

SET @decimal = 41.12345; -- Значение округлится, т.к после запятой больше 3-х знаков.

PRINT @decimal;

---------------------------------- SmallMoney -------------------------------

-- Вещественное со знаком, с фиксированной точкой SmallMoney.

-- Диапазон значений от -214 748.3648 до 214 748.3647

PRINT 'SmallMoney';

DECLARE @smoney SmallMoney = 214748.3647;

PRINT @smoney;

SET @smoney = -214748.3648;

PRINT @smoney;

------------------------------------- Money ---------------------------------

-- Вещественное со знаком, с фиксированной точкой Money.

-- Диапазон значений от -2^63 до 2^63-1

PRINT 'Money';

DECLARE @money Money = 1.4234;

PRINT @money;

SET @money = 2.234954;

PRINT @money;

-----------------------------------------------------------------------------

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- Текстовые типы данных в T-SQL

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-------------------------------------- Char ---------------------------------

-- Текстовый тип данных записывается как Char,

-- допустимое количество символов от 1 до 8000 символов.

-- Текстовые типы данных записывается как Char(n),

-- где n - определяет максимальное количество символов(максимально n = 8000).

PRINT 'Char'

DECLARE @char char(5)= 'Hello';

PRINT @char;

SET @char = '1234dfghdf'; -- Часть строки "fghdf" не запишется, т.к. допустимое количество символов (5)

PRINT @char;

-------------------------------------- VarChar ------------------------------

-- Текстовый тип данных записывается как VarChar,

-- диапазон значений от 1 до 2^31 символов.

-- Текстовые типы данных записывается как VarChar(n/max),

-- где n - определяет максимальное количество символов(максимально n = 8000),

-- или же вместо n записывается max то максимальное количество символов становится 2^31.

PRINT 'VarChar'

DECLARE @vchar varchar(max)='Hello'; -- разрешено записывать 2^31 символов.

PRINT @vchar;

SET @vchar = '1234dfghdf';

PRINT @vchar;

-------------------------------------- NChar --------------------------------

-- Текстовый тип данных в Unicode кодировке записывается как NChar,

-- диапазон значений от 1 до 4000 символов.

-- Текстовые типы данных записывается как NChar(n),

-- где n - определяет максимальное количество символов(максимально n = 4000).

-- Если количество вводимых символов меньше n то разница заполниться пробелами

PRINT 'NChar'

DECLARE @Nchar Nchar(7) = 'Привет!';

PRINT @Nchar;

SET @Nchar = '1234 приехали'; -- Часть строки "иехали" не запишется, т.к. допустимое количество символов (7)

PRINT @Nchar;

------------------------------------- NvarChar ------------------------------

-- Текстовый тип данных в Unicode кодировке записывается как NvarChar,

-- диапазон значений от 1 до 2^31 символов.

-- Текстовые типы данных записывается как NvarChar(n/max),

-- где n - определяет максимальное количество символов(максимально n = 4000),

-- или же вместо n записывается max то максимальное количество символов становится 2^31.

PRINT 'NvarChar'

DECLARE @NvarChar NvarChar(max) = 'Привет Мир!';

PRINT @NvarChar;

SET @NvarChar = 'Много много много много много много много много много много много слов';

PRINT @NvarChar;

-----------------------------------------------------------------------------

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- Типы данных даты и времени в T-SQL

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

------------------------------------ DateTime -------------------------------

-- Типы данных даты и времени DateTime.

-- Диапазон значений от 1 января 1753 года до 31 декабря 9999 года.

PRINT 'DateTime'

-- CAST (expression AS data\_type) - это функция преобразования типов,

-- преобразует значение expression к типу data\_type.

DECLARE @datetime DateTime = CAST('2007-05-08 12:35:29.123' AS DateTime);

PRINT @datetime;

SET @datetime = CURRENT\_TIMESTAMP; -- Текущая дата и время

PRINT @datetime;

------------------------------------ DateTime2 ------------------------------

-- Типы данных даты и времени DateTime2.

-- Диапазон значений от 1 января 1 года до 31 декабря 9999 года.

-- Типы данных даты и времени записывается как DateTime2(n),

-- где n - определяет точность до 100нс ( 0<n<7 )

PRINT 'DateTime2';

DECLARE @datetime2 DateTime2(5) = CAST('2007-05-08 12:35:29.1234567' AS DateTime2)

PRINT @datetime2;

SET @datetime2 = CURRENT\_TIMESTAMP; -- Текущая дата и время

PRINT @datetime2;

-------------------------------- SmallDateTime ------------------------------

-- Типы данных даты и времени SmallDateTime.

-- Диапазон значений от 1 января 1900 года до 6 июня 2079 года.

PRINT 'SmallDateTime';

DECLARE @smalldatetime SmallDateTime = CAST('2007-05-08 12:35:29' AS SmallDateTime)

PRINT @smalldatetime;

SET @smalldatetime = CURRENT\_TIMESTAMP; -- Текущая дата и время

PRINT @smalldatetime;

-------------------------------- DateTimeOffSet ------------------------------

-- Типы данных даты и времени DateTimeOffSet.

-- Определяет дату, объединенную со временем дня, с учетом часового пояса в 24-часовом формате.

-- Диапазон значений от 1 января 1 года до 31 декабря 9999 года.

-- Типы данных даты и времени записывается как DateTimeOffSet(n),

-- где n - определяет точность до 100нс ( 0<n<7 )

PRINT 'DateTimeOffSet';

DECLARE @dateTimeOffSet DateTimeOffSet(5) = CAST('2007-05-08 12:35:29.1234567 +12:15' AS DateTimeOffSet)

PRINT @dateTimeOffSet;

SET @dateTimeOffSet = CURRENT\_TIMESTAMP; -- Текущая дата и время

PRINT @dateTimeOffSet;

------------------------------------ Date -----------------------------------

-- Типы данных даты Date.

-- Диапазон значений от 1 января 1 года до 31 декабря 9999 года.

PRINT 'Date';

DECLARE @date Date = CAST('2007-05-08 12:35:29.1234567' AS Date)

PRINT @date;

SET @date = CURRENT\_TIMESTAMP; -- Текущая дата и время

PRINT @date;

------------------------------------ Time -----------------------------------

-- Типы данных времени Time.

-- Типы данных времени записывается как Time(n),

-- где n - определяет точность до 100нс ( 0<n<7 )

PRINT 'Time';

DECLARE @time Time(5) = CAST('2007-05-08 12:35:29.1234567' AS Time);

PRINT @time;

SET @time = CURRENT\_TIMESTAMP;

PRINT @time;

-----------------------------------------------------------------------------

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- Типы двоичных данных в T-SQL

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

------------------------------------ Binary ---------------------------------

-- Типы двоичных данных Binary.

-- Диапазон значений от 1 до 8000 байт.

-- Типы двоичных данных записывается как Binary(n),

-- где n - определяет максимальное количество байт(максимально n = 8000).

PRINT 'Binary'

DECLARE @binary Binary(1) = 16;

PRINT @binary;

----------------------------------- VarBinary -------------------------------

-- Типы двоичных данных VarBinary.

-- Диапазон значений от 1 до 2^31 байт.

-- Типы двоичных данных записывается как VarBinary(n/max),

-- где n - определяет максимальное количество байт(максимально n = 8000),

-- или же вместо n записывается max то максимальное количество байт становится 2^31.

PRINT 'VarBinary'

DECLARE @varBinary VarBinary(max) = 2147483647;

PRINT @varBinary;

-----------------------------------------------------------------------------

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- Создание Базы Данных.

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- Создаём базу данных с именем ShopDB.

CREATE DATABASE ShopDB

ON -- Задаем параметры Базы Данных.

(

NAME = 'ShopDB', -- Указываем логическое имя БД (используется при обращении к БД).

FILENAME = 'D:\ShopDB.mdf', -- Указываем Физическое полное имя файла БД.

SIZE = 10MB, -- Задаем начальный размер файла БД.

MAXSIZE = 100MB, -- Задаем максимальный размер файла БД.

FILEGROWTH = 10MB -- Задаем значение, на которое будет увеличиваться размер файла БД.

)

LOG ON -- Задаем параметры журнала Базы Данных.

(

NAME = 'LogShopDB', -- Указываем логическое имя журнала БД (используется при обращении к журналу БД).

FILENAME = 'D:\ShopDB.ldf', -- Указываем Физическое полное имя файла журнала БД.

SIZE = 5MB, -- Задаем начальный размер файла журнала БД.

MAXSIZE = 50MB, -- Задаем максимальный размер файла журнала БД.

FILEGROWTH = 5MB -- Задаем значение, на которое будет увеличиваться размер файла журнала БД.

)

COLLATE Cyrillic\_General\_CI\_AS -- Задаем кодировку для базы данных по умолчанию

-- Выделить с 6 по 22 строку и нажать F5.

----------------------------------------------------------------------------------

-- Выводим информацию о Базе Данных - ShopDB.

EXECUTE sp\_helpdb ShopDB;

----------------------------------------------------------------------------------

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- Создание Таблицы в Базе Данных ShopDB.

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- ВНИМАНИЕ!

-- Указываем явно имя Базы Данных, которую следует использовать, так как, существует вероятность

-- создания таблицы в Базе Данных, указанной в выпадающем списке на панели инструментов.

-- Например: Часто, ошибочно создают таблицы в БД master и кажется что таблица не создалась.

USE ShopDB

GO

----------------------------------------------------------------------------------

-- Создаем таблицу с именем MyFriends, которая будет содержать пять столбцов.

-- Первый столбец с именем FriendId, типа SmallInt с заданным автоинкрементом.

-- Второй столбец с именем FirstName, типа Varchar, размерностью в 20 символов.

-- Третий столбец с именем LastName, типа Varchar, размерностью в 20 символов.

-- Четвертый столбец с именем BirthDate, типа Date.

-- Пятый столбец с именем PhoneNumber, типа Char, размерностью в 10 символов.

CREATE TABLE MyFriends

(

-- Ключевое слово IDENTITY задает начальное значение и устанавливает авто инкремент.

-- По умолчанию значение первой ячейки равно 1 и с каждой новой записью увеличивается на 1.

FriendId smallint IDENTITY NOT NULL,

FirstName Varchar(20) NOT NULL,

LastName Varchar(20) NULL,

BirthDate date NULL,

PhoneNumber char(10) NOT NULL

)

GO -- Конец пакета инструкций.

----------------------------------------------------------------------------------

Урок 2. T-SQL

DML – DATA MANIPULATION LANGUAGE

Подключить бд -> Object Explorer->DataBases->Attach

USE ShopDB; -- Указываем БД которую будем использовать

GO

-------------------------------------------------------------------------

/\* Создем таблицу с именем MyFriends\*/

-------------------------------------------------------------------------

CREATE TABLE MyFriends

(

FrndId smallint IDENTITY NOT NULL,

FstName Varchar(20) NOT NULL,

LstName Varchar(20) NULL,

BthDate date NULL,

Ntel char(10) NOT NULL

)

GO -- Конец пакета инструкций.

-------------------------------------------------------------------------

-- Процедура sp\_help возвращает сведения об объекте базы данных.

-------------------------------------------------------------------------

EXEC sp\_help MyFriends

-------------------------------------------------------------------------

/\* Оператор вставки INSERT \*/

-------------------------------------------------------------------------

-- Записываем строку в таблицу MyFreand

INSERT INTO MyFriends -- Ключевое слово INTO можно не использовать.

(FstName, BthDate, Ntel) -- Указание порядка записи данных.

VALUES

('Max', 'Petrov', '02/03/1988', '093\*\*\*\*\*\*\*'); -- Вводимые данные.

GO

-------------------------------------------------------------------------

-- Можно явно изменить порядок записываемых даных.

INSERT INTO MyFriends

(LstName, FstName, BthDate, Ntel) -- Указание порядка записи данных.

VALUES

('Martinov', 'Kostya', '01/03/1986', '097\*\*\*\*\*\*\*'); -- Вводимые данные.

GO

-------------------------------------------------------------------------

INSERT INTO MyFriends

-- Указание порядка записи данных отсутствует, значит используем порядок по умолчанию.

VALUES

('Vitia', 'Sokolov', '01/03/1986', '095\*\*\*\*\*\*\*'); -- Вводимые данные.

GO

-------------------------------------------------------------------------

INSERT INTO MyFriends

-- Указание порядка записи данных отсутствует, значит используем порядок по умолчанию.

VALUES

('Sokolov', 'Vitia', '01/03/1986', '095\*\*\*\*\*\*\*'); -- Вводимые данные (Ошибка очередности ввода!).

GO

SELECT \* FROM MyFriends

-------------------------------------------------------------------------

INSERT INTO MyFriends

VALUES

-- Нельзя оставлять пустыми поля помеченные ограничением NOT NULL

(NULL, 'Imelianov', '01/03/1986', '095\*\*\*\*\*\*\*'); -- Вводимые данные.

GO

-------------------------------------------------------------------------

INSERT INTO MyFriends

VALUES

-- Допускается оставлять пустыми поля помеченные ограничением NULL

('Vitia', null, '01/03/1986', '095\*\*\*\*\*\*\*'); -- Вводимые данные.

GO

-------------------------------------------------------------------------

INSERT INTO MyFriends

(FstName,LstName,BthDate,Ntel)

VALUES -- Начиная с 2008 SQL, допускается вводить несколько строк одновременно

('Max', 'Petrov', '02/21/1988', '093\*\*\*\*\*\*\*'),

('Alex', 'Vare', '03/20/1990', '093\*\*\*\*\*\*\*'),

('Sergei', 'Smolnikov', '05/31/1991', '093\*\*\*\*\*\*\*'),

('Petr', 'Vitiushkin', '03/06/1989', '093\*\*\*\*\*\*\*')

GO

-------------------------------------------------------------------------

INSERT INTO MyFriends

-- Указание порядка записи данных для несколиких строк не обязательно

VALUES

('Max', 'Petrov', '02/21/1988', '093\*\*\*\*\*\*\*'),

('Alex', 'Vare', '03/20/1990', '093\*\*\*\*\*\*\*'),

('Sergei', 'Smolnikov', '05/31/1991', '093\*\*\*\*\*\*\*'),

('Petr', 'Vitiushkin', '03/06/1989', '093\*\*\*\*\*\*\*');

GO

-------------------------------------------------------------------------

USE ShopDB;

GO

--------------------------------------------------------------------------

-- Выборка данных. Запрос SELECT

--------------------------------------------------------------------------

-- Производим выборку всех данных из таблицы MyFriends.

SELECT \* FROM MyFriends;

-- Производим выборку данных из столбца FstName, таблицы MyFriends

SELECT FstName FROM MyFriends;

-- Производим выборку данных из столбца LstName, таблицы MyFriends

SELECT LstName FROM MyFriends;

SELECT BthDate FROM MyFriends;

-- Производим выборку данных из столбцов FstName и LstName, таблицы MyFriends

SELECT FstName, LstName FROM MyFriends;

-- Производим выборку данных из столбцов FstName, LstName и BthDate, таблицы MyFriends

SELECT FstName,LstName,BthDate FROM MyFriends;

--------------------------------------------------------------------------

USE AdventureWorks2012;

GO

/\*

Выбор из таблицы CreditCard всех:

типов кредитных карт,

номеров карт,

месяца до которого действительна карта, и

года до которого действительна карта

\*/

SELECT CardType, CardNumber, ExpMonth, ExpYear FROM Sales.CreditCard; -- Sales - схема

--------------------------------------------------------------------------

-- Конструкция WHERE.

--------------------------------------------------------------------------

-- Производим выборку всех данных из таблицы CreditCard, где значения ячеек столбца CreditCardID равны 10.

SELECT \* FROM Sales.CreditCard

WHERE CreditCardID = 10; -- В конструкции WHERE применена операция сравнения (=)

-- Производим выборку всех данных из таблицы CreditCard, где значения ячеек столбца CreditCardID меньше 10.

SELECT \* FROM Sales.CreditCard

WHERE CreditCardID < 10; -- В конструкции WHERE применена операция сравнения (<)

-- Производим выборку всех данных из таблицы CreditCard,

-- где значения ячеек столбца CreditCardID лежат в диапазоне от 1 до 3 (включительно).

SELECT \* FROM Sales.CreditCard

WHERE CreditCardID BETWEEN 1 AND 3 -- В конструкции WHERE применена операция проверки диапазона.

-- Производим выборку данных из столбцов CardType и ExpYear, таблицы CreditCard,

-- где значения ячеек столбца ExpYear равно 2005 или 2006.

SELECT \* FROM Sales.CreditCard

WHERE ExpYear = 2005 OR ExpYear = 2007 -- Логическая операция "ИЛИ"

-- Производим выборку всех данных из таблицы CreditCard,

-- где значения ячеек столбца ExpYear лежат в диапазоне от 2005 до 2006 (включительно).

SELECT \* FROM Sales.CreditCard

WHERE ExpYear BETWEEN 2005 AND 2007 -- В конструкции WHERE применена операция проверки диапазона.

-- Производим выборку данных из столбцов CardType и ExpYear, таблицы CreditCard,

-- где значения ячеек столбца ExpYear равно 2005 и значения ячеек столбца CardType равно 'Vista'.

SELECT CardType, ExpYear FROM Sales.CreditCard

WHERE ExpYear = 2005 AND CardType = 'Vista' -- Логическая операция "И"

-- Производим выборку данных из столбцов CardType и ExpYear, таблицы CreditCard,

-- где значения ячеек столбца ExpYear не равно 2006

SELECT CardType, ExpYear FROM Sales.CreditCard

WHERE NOT ExpYear = 2006 -- Логическая операция "НЕ"

-- Производим выборку данных из столбцов CardType и ExpYear, таблицы CreditCard,

-- где значения ячеек столбца CardType соответствуют шаблону 'ColonialVoice'

SELECT CardType, ExpYear FROM Sales.CreditCard

WHERE CardType LIKE 'ColonialVoice' -- LIKE - операция проверки соответствия заданному шаблону - 'ColonialVoice'

SELECT CardType, ExpYear FROM Sales.CreditCard

WHERE CardType = 'ColonialVoice'

-- Производим выборку данных из столбцов CardType и ExpYear, таблицы CreditCard,

-- где значения ячеек столбца CardType соответствуют шаблону 'Dis%'

SELECT CardType, ExpYear FROM Sales.CreditCard

WHERE CardType LIKE 'Dis%' -- Вывод всех карт, имя типа которых начинается на 'Dis',

-- знак % обозначет произвольное количество символов после Dis.

SELECT CardType, ExpYear FROM Sales.CreditCard

WHERE CardType = 'Dis%' -- Выборка пустая потому что такого значения как 'Dis%' нет в таблице

-- Производим выборку данных из столбцов CardType и ExpYear, таблицы CreditCard,

-- где значения ячеек столбца CardType соответствуют шаблону 'Vis\_\_'

SELECT CardType, ExpYear FROM Sales.CreditCard

WHERE CardType LIKE 'Vis\_\_' --\_ - нижнее подчеркивание определяет любой один символ после Vis

-- Производим выборку данных из столбцов CardType и ExpYear, таблицы CreditCard,

-- где значения ячеек столбца CardType соответствуют шаблону 'Vis\_'

SELECT CardType, ExpYear FROM Sales.CreditCard

WHERE CardType LIKE 'Vis\_' -- почему выборка пустая?

-- Производим выборку данных из столбцов CardType и ExpYear, таблицы CreditCard,

-- где значения ячеек столбца CardType соответствуют шаблону 'Vis\_а'

SELECT CardType, ExpYear FROM Sales.CreditCard

WHERE CardType LIKE 'Vis\_a'

-- Производим выборку данных из столбцов CardType и ExpYear, таблицы CreditCard,

-- где значения ячеек столбца CardType соответствуют шаблону '%ist%'

SELECT CardType, ExpYear FROM Sales.CreditCard

WHERE CardType LIKE '%ist%'

-- Производим выборку всех данных из таблицы CreditCard,

-- где значения ячеек столбца ExpMonth равняются одному из значений ряда (1, 5, 11)

SELECT \* FROM Sales.CreditCard WHERE ExpMonth IN (1, 5, 11); -- IN (1, 5, 11) определяет значения ExpMonth равные 1 или 5 или 11

SeLeCt \* from Sales.CreditCard where ExpMonth = 1 or ExpMonth = 5 or ExpMonth = 11;

USE AdventureWorks2012

GO

SELECT BusinessEntityID, FirstName, LastName, MiddleName, ModifiedDate FROM Person.Person

--------------------------------------------------------------------------

-- Конструкция ORDER BY - УПОРЯДОЧИТЬ ПО...

--------------------------------------------------------------------------

-- Производим выборку данных из столбцов BusinessEntityID, FirstName, LastName, MiddleName и ModifiedDate,

-- где данные упорядочены по столбцу FirstName, таблицы Person.Person

SELECT BusinessEntityID, FirstName, LastName, MiddleName, ModifiedDate

FROM Person.Person

ORDER BY FirstName; -- Упорядочить по имени.

-- Производим выборку данных из столбцов BusinessEntityID, FirstName, LastName, MiddleName и ModifiedDate,

-- где данные упорядочены по столбцу ModifiedDate, таблицы Person.Person

SELECT BusinessEntityID, FirstName, LastName, MiddleName, ModifiedDate

FROM Person.Person

ORDER BY ModifiedDate; -- Сортировать по ModifiedDate

-- Производим выборку данных из столбцов BusinessEntityID, FirstName, LastName, MiddleName и ModifiedDate,

-- где данные упорядочены по столбцам FirstName и LastName, таблицы Person.Person

SELECT BusinessEntityID, FirstName, LastName, MiddleName, ModifiedDate

FROM Person.Person

ORDER BY FirstName, LastName; -- Cортировать по Имени и Фамилии

-- Производим выборку данных из столбцов BusinessEntityID, FirstName, LastName, MiddleName и ModifiedDate,

-- где данные упорядочены по столбцу FirstName, таблицы Person.Person

SELECT BusinessEntityID, FirstName, LastName, MiddleName, ModifiedDate

FROM Person.Person

ORDER BY FirstName ASC; -- Сортировка по возрастанию. ASC - по умолчанию.

-- Производим выборку данных из столбцов BusinessEntityID, FirstName, LastName, MiddleName и EmailAddress,

-- где данные упорядочены в порядке убывания, по столбцу FirstName, таблицы Person.Contact

SELECT BusinessEntityID, FirstName, LastName, MiddleName, ModifiedDate

FROM Person.Person

ORDER BY FirstName DESC; -- Сортировка по убыванию.

USE AdventureWorks2012

GO

--------------------------------------------------------------------------

-- Агрегирование данных. Конструкция GROUP BY

--------------------------------------------------------------------------

-- Производим выборку данных из столбцов SalesOrderID и OrderQty,

-- таблицы Sales.SalesOrderDetail, где значения ячеек столбца SalesOrderID

-- равняются одному из значений ряда (43666, 43660, 43664)

SELECT SalesOrderID, OrderQty

FROM Sales.SalesOrderDetail

WHERE SalesOrderID IN (43666, 43660, 43664);

-- Производим выборку данных из столбца SalesOrderID

-- и возвращаемого значения агрегированной функции SUM по столбцу OrderQty,

-- таблицы Sales.SalesOrderDetail, где значения ячеек столбца SalesOrderID

-- равняется одному из значений ряда (43666, 43660, 43664),

-- при этом производится группировка по значениям столбца SalesOrderID.

-- Агрегирование - процесс обьединения элементов

SELECT SalesOrderID, SUM(OrderQty) -- Агрегированная функция SUM()

FROM Sales.SalesOrderDetail

WHERE SalesOrderID IN (43666, 43660, 43664)

GROUP BY SalesOrderID;

SELECT SalesOrderID AS ID, SUM(OrderQty) AS Total -- После ключевого слова AS задаем псевдоним(alias) для результата функции SUM().

FROM Sales.SalesOrderDetail

WHERE SalesOrderID IN (43666, 43660, 43664)

GROUP BY SalesOrderID; -- Агрегирование данных.

-- Производим выборку возвращаемого значения агрегированной функции COUNT таблицы HumanResources.Employee.

SELECT COUNT(\*) AS Emp

FROM HumanResources.Employee; -- Агрегированная функция COUNT() выводит количество строк в таблице

-- Производим выборку данных из столбца ProductID и возвращаемого значения агрегированной функции COUNT,

-- таблицы Sales.SalesOrderDetail, при этом производится группировка по значениям столбца ProductID.

SELECT ProductID AS Product, COUNT(\*) AS [Count]

FROM Sales.SalesOrderDetail

GROUP BY ProductID;

--------------------------------------------------------------------------

-- Производим выборку данных из столбца ProductID и возвращаемого значения агрегированной функции COUNT,

-- таблицы Sales.SalesOrderDetail, при этом производится группировка по значениям столбца ProductID,

-- где возвращаемое значение агрегированной функции COUNT больше 3300-ти.

SELECT ProductID AS Product, COUNT(\*) AS [Count]

FROM Sales.SalesOrderDetail

GROUP BY ProductID

HAVING COUNT(\*) > 3300; -- HAVING - должен использоваться совместно с GROUP BY (HAVING аналогичен WHERE).

-- HAVING - условие применяемое только к группам.

--------------------------------------------------------------------------

-- Ошибка. HAVING - должен использоваться совместно с GROUP BY.

SELECT ProductID AS Product

FROM Sales.SalesOrderDetail

HAVING ProductID > 10; -- Неправильное использование HAVING (без GROUP BY)

-- WHERE работает до группировки, а HAVING работает вместе с группировкой.

SELECT SalesOrderID, COUNT(\*) AS TOTAL

FROM Sales.SalesOrderDetail

WHERE SalesOrderID IN (43666, 43660, 43664)

GROUP BY SalesOrderID

HAVING COUNT(\*) > 5;

USE ShopDB;

GO

INSERT MyFriends

(FstName, Ntel)

VALUES ('Max', '050\*\*\*\*\*\*\*');

GO

SELECT \* FROM MyFriends;

--------------------------------------------------------------------------

-- Оператор UPDATE (изменение(обновление) данных в таблице)

--------------------------------------------------------------------------

UPDATE MyFriends

SET Ntel = '098\*\*\*\*\*\*\*' -- изменить номер телефона того,

WHERE FstName = 'Max' -- чье имя 'Max'

GO

SELECT \* FROM MyFriends;

-- Возможно изменять значение сразу в нескольких столбцах

UPDATE MyFriends

SET FstName = 'TestName',

Ntel = '000\*\*\*\*\*\*\*'

WHERE FstName = 'Max'

SELECT \* FROM MyFriends;

UPDATE MyFriends

SET Ntel = '060\*\*\*\*\*\*\*' -- Если не задана консрукция WHERE, то изменяется весь столбец на указанное значение

GO

SELECT \* FROM MyFriends;

--------------------------------------------------------------------------

-- Оператор DELETE (удаление данных из таблицы)

--------------------------------------------------------------------------

-- Удалить всех пользователей имя которых TestName

DELETE MyFriends

WHERE FstName = 'TestName';

SELECT \* FROM MyFriends;

-- Удаление всех данных из таблицы с помощью DELETE.

DELETE MyFriends;

SELECT \* FROM MyFriends;

-- Для удаления всех данных из таблицы лучше использовать - TRUNCATE TABLE,

-- так как TRUNCATE удаляет информацию из базы быстрее чем стандартный DELETE.

TRUNCATE TABLE MyFriends;

SELECT \* FROM MyFriends;

Основы DLL

* **Data Definition Language** (**DDL**) (язык описания данных) — это семейство компьютерных языков, используемых в компьютерных программах для описания структуры баз данных.
*  В DDL входят такие операторы как:CREATE (создать), ALTER (изменить), DROP (удалить).

**Целостность базы данных** — свойство базы данных обеспечивающее корректность и непротиворечивость хранимых данных в любой момент времени

Целостность БД не гарантирует достоверности содержащейся в ней информации, но обеспечивает по крайней мере правдоподобность этой информации, отвергая заведомо невероятные, невозможные значения.

Целостность базы данных обеспечивается ограничениями на вводимые пользователем данные.

**Типы целостности** базы данных:

* 1)Доменная целостность
* 2)Целостность сущностей
* 3)Ссылочная целостность

**Доменная целостность** гарантирует наличие в некотором столбце только допустимых значений. Ограничения, обеспечивающие доменную целостности:

* 1)Использование ограничений проверки(CHECK).
* 2)Использование ограничений на умолчания(DEFAULT).

**Целостность сущностей** гарантирует уникальность записей в таблицах(сущностях). Ограничения, обеспечивающие целостность сущностей:

* 1)Использование ограничений первичного ключа(Primary Key).
* 2)Использование ограничений на уникальность(UNIQUE).

**Ссылочная целостность** гарантирует невозможность создания записи в одной таблице, которая ссылается на несуществующую запись другой таблицы. Ограничения обеспечивающие

ссылочную целостность:

* 1)Использование ограничений внешнего ключа(Foreign Key).

-- Перед удалением какой либо базы, рекомендуется

-- предварительно переключиться на базу master.

USE master;

GO

-- Оператор DROP DATABASE удаляет БазуДанных

DROP DATABASE ShopDB;

GO

-------------------------------------------------------------------------

/\* Создание Базы Данных\*/

-------------------------------------------------------------------------

CREATE DATABASE ShopDB

ON

(

NAME = 'ShopDB',

FILENAME = 'D:\ShopDB.mdf',

SIZE = 10MB,

MAXSIZE = 100MB,

FILEGROWTH = 10MB

)

LOG ON

(

NAME = 'LogShopDB',

FILENAME = 'D:\LogShopDB.ldf',

SIZE = 5MB,

MAXSIZE = 50MB,

FILEGROWTH = 5MB

)

COLLATE Cyrillic\_General\_CI\_AS -- Задаем кодировку для базы данных по умолчанию

-------------------------------------------------------------------------

/\* Изменение Базы Данных\*/

-------------------------------------------------------------------------

EXEC sp\_helpdb ShopDB

ALTER DATABASE ShopDB -- Оператор ALTER DATABASE изменяет БД

MODIFY FILE -- Изменение размера БД

( NAME = 'ShopDB',

SIZE = 100MB )

GO

ALTER DATABASE ShopDB -- Оператор ALTER DATABASE изменяет БД

MODIFY FILE

( NAME = 'ShopDB',

MAXSIZE = 1000MB, -- Изменение максимального размера БД

FILEGROWTH = 20% ) -- и параметра увеличения размера БД

-- Если FILEGROWTH задается в процентах, то процент считается от текущего размера БД (FILEGROWTH = 0,2\*SIZE).

GO

ALTER DATABASE ShopDB

MODIFY FILE

(

NAME = 'ShopDB',

MAXSIZE = 10MB -- Измененный максимальный размер не может быть меньше текущего размера БД

)

GO

ALTER DATABASE ShopDB

MODIFY FILE

( NAME = 'ShopDB',

SIZE = 100MB ) -- Не разрешается задавать размер БД меньше или равный текущему.

GO

EXEC sp\_helpdb ShopDB

USE ShopDB

GO

-------------------------------------------------------------------------

/\* Создание таблицы \*/

-------------------------------------------------------------------------

DROP TABLE Customers; -- DROP TABLE - оператор удаление таблицы

GO

CREATE TABLE Customers

( -- Ключевое слово IDENTITY создает столбец идентификации.(seed = 1, step = 2)

CustomerNo int IDENTITY(1, 2) NOT NULL,

CustomerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

Address2 varchar(25) DEFAULT 'Unknown', -- Ключевое слово DEFAULT присваивает значение по умолчанию.

City varchar(15) NOT NULL,

[State] char(2) NOT NULL,

Zip varchar(10) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(10) NOT NULL,

FedIDNo varchar(10) NOT NULL,

DateInSystem smalldatetime NOT NULL

);

GO

DROP TABLE Customers;

GO

CREATE TABLE Customers

( -- Ключевое слово IDENTITY создает столбец идентификации.(seed = 1, step = 2)

CustomerNo int IDENTITY(1, 2) NOT NULL,

CustomerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

Address2 varchar(25) DEFAULT 'Unknown', -- Ключевое слово DEFAULT присваивает значение по умолчанию.

City varchar(15) NOT NULL,

[State] char(2) NOT NULL,

Zip varchar(10) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(10) NOT NULL,

FedIDNo varchar(10) NOT NULL,

DateInSystem smalldatetime NOT NULL

)

WITH (DATA\_COMPRESSION = ROW); -- таблица с сжатием по строкам (Не работает в SQL Express)

GO

INSERT INTO Customers

(CustomerName, Address1, City, [State], Zip, Contact, Phone, FedIDNo, DateInSystem)

VALUES

('Jeremy', 'new str', 'NewCity', 'ds', 'newZIP', 'newContact', '0567878932', 'NewIDNO', GETDATE());

GO

SELECT \* FROM Customers;

GO

-------------------------------------------------------------------------

/\* Изменение таблицы \*/

-------------------------------------------------------------------------

-- Изменяем таблицу Customers,

-- добавив столбец NewFild

ALTER TABLE Customers -- Оператор ALTER TABLE изменяет таблицу

ADD -- ADD - добавляет

NewFild int NULL -- столбец.

GO

SELECT \* FROM Customers;

GO

-- Изменяем таблицу Customers,

-- удалив столбец NewFild

ALTER TABLE Customers -- Оператор ALTER TABLE изменяет таблицу

DROP COLUMN -- DROP COLUMN - удаляет

NewFild -- столбец.

GO

SELECT \* FROM Customers;

GO

ALTER TABLE Customers

ADD

NewFild2 varchar(10) NOT NULL

-- Невозможно добавить колонку с ограничением NOT NULL не установив значение по умолчанию.

GO

ALTER TABLE Customers

ADD

NewFild2 varchar(10) NOT NULL

DEFAULT 'Unknown'

GO

SELECT \* FROM Customers

GO

USE ShopDB

GO

DROP TABLE Customers;

GO

-------------------------------------------------------------------------

-- Создание первичного ключа

-------------------------------------------------------------------------

-- PRIMARY KEY (первичный ключ, ограничение первичного ключа)

-- Значения ячеек столбца(-ов) с ограничением PRIMARY KEY

-- однозначно определяет (-ют) каждую строку в таблице.

CREATE TABLE Customers

(

CustomerNO int NOT NULL PRIMARY KEY, -- На столбце CustomerNO задаем первичный ключ

CustomerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12)

)

GO

INSERT INTO Customers

(CustomerNO, CustomerName, Address1, City, Contact, Phone)

VALUES

(1,'Петренко Петр Петрович', 'Луганская 25', 'Конотоп', 'PetrPetrenko@mail.ru', '(093)1231212'),

(2,'Иваненко Иван Иванович', 'Дихтяревская 5', 'Чернигов', 'IvanenkoIvan@gmail.com', '(095)2313244');

GO

-- Попытка нарушить уникальность.

INSERT INTO Customers

(CustomerNO, CustomerName, Address1, City, Contact, Phone)

VALUES

(1,'Сидоров Семен Семенович', 'Драйзера 12', 'Харьков', 'SemSem@mail.ru', '(063)4533272'); -- Ошибка!

SELECT \* FROM Customers;

DROP TABLE Customers;

GO

-- Если первичный ключ состоит из двух и более столбцов,

-- его называют составным первичный ключом.

-- Попытка вставить второй Первичный ключ в таблицу приводит к ошибке.

-- Первичный ключ в таблице может быть только один (простой или составной)

CREATE TABLE Customers

(

CustomerNo int NOT NULL,

CustomerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12) NOT NULL,

PRIMARY KEY (CustomerNo, CustomerName) -- задаем составной первичный ключ на столбцах CustomerNo, CustomerName

)

GO

INSERT INTO Customers

(CustomerNO, CustomerName, Address1, City, Contact, Phone)

VALUES

(1,'Петренко Петр Петрович', 'Луганская 25', 'Конотоп', 'PetrPetrenko@mail.ru', '(093)1231212'),

(2,'Иваненко Иван Иванович', 'Дихтяревская 5', 'Чернигов', 'IvanenkoIvan@gmail.com', '(095)2313244');

GO

INSERT INTO Customers

(CustomerNO, CustomerName, Address1, City, Contact, Phone)

VALUES

(1,'Сидоров Семен Семенович', 'Драйзера 12', 'Харьков', 'SemSem@mail.ru', '(063)4533272');

INSERT INTO Customers

(CustomerNO, CustomerName, Address1, City, Contact, Phone)

VALUES

(2,'Иваненко Иван Иванович', 'Янгеля 32', 'Киев', 'IvanenkoIvan@mail.ru', '(050)1752124'); -- Ошибка!

SELECT \* FROM Customers;

-- Создание ограничения первичного ключа на существующей таблице

DROP TABLE Customers;

GO

CREATE TABLE Customers

(

CustomerNO int IDENTITY NOT NULL,

CustomerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12)

)

GO

-- Изменяем таблицу Customers задав ограничение первичного ключа на столбце CustomerNo

ALTER TABLE Customers

ADD CONSTRAINT PK\_Customers

PRIMARY KEY (CustomerNo)

INSERT INTO Customers

(CustomerName, Address1, City, Contact, Phone)

VALUES

('Петренко Петр Петрович', 'Луганская 25', 'Конотоп', 'PetrPetrenko@mail.ru', '(093)1231212'),

('Иваненко Иван Иванович', 'Дихтяревская 5', 'Чернигов', 'IvanenkoIvan@gmail.com', '(095)2313244');

GO

SELECT \* FROM Customers;

DROP TABLE Customers;

GO

CREATE TABLE Customers

(

CustomerNO int NOT NULL,

CustomerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12)

)

GO

ALTER TABLE Customers

ADD -- Имя ограничения не является обязательным.

PRIMARY KEY (CustomerNo,Address1)

INSERT INTO Customers

(CustomerNO, CustomerName, Address1, City, Contact, Phone)

VALUES

(1,'Петренко Петр Петрович', 'Луганская 25', 'Конотоп', 'PetrPetrenko@mail.ru', '(093)1231212'),

(2,'Иваненко Иван Иванович', 'Дихтяревская 5', 'Чернигов', 'IvanenkoIvan@gmail.com', '(095)2313244');

GO

INSERT INTO Customers

(CustomerNO, CustomerName, Address1, City, Contact, Phone)

VALUES

(1,'Сидоров Семен Семенович', 'Драйзера 12', 'Харьков', 'SemSem@mail.ru', '(063)4533272');

INSERT INTO Customers

(CustomerNO, CustomerName, Address1, City, Contact, Phone)

VALUES

(2,'Иваненко Иван Иванович', 'Янгеля 32', 'Киев', 'IvanenkoIvan@mail.ru', '(050)1752124');

SELECT \* FROM Customers;

USE ShopDB

GO

DROP TABLE Customers;

GO

CREATE TABLE Customers

(

CustomerNO int IDENTITY NOT NULL

PRIMARY KEY,

CustomerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12)

)

GO

-------------------------------------------------------------------------

-- Создание внешнего ключа

-------------------------------------------------------------------------

-- FOREIGN KEY (внешний ключ, ограничение внешнего ключа)

DROP TABLE Orders;

GO

-- Значения ячеек столбца дочерней таблицы с ограничением FOREIGN KEY

-- должно совпадать с одним из имеющихся значений в родительской таблице.

CREATE TABLE Orders

(

OrderID int IDENTITY NOT NULL

PRIMARY KEY,

CustomerNo int NOT NULL -- Столбец дочерней таблицы для которой задан FOREIGN KEY

FOREIGN KEY REFERENCES Customers(CustomerNo), -- Родительская таблица Customers ее столбец CustomerNo.

OrderDate date NOT NULL,

Goods varchar(30) NOT NULL

)

GO

INSERT INTO Customers

(CustomerName, Address1, City, Contact, Phone)

VALUES

('Petrenko Petr Petrovich', 'Luganskaya 25', 'Konotop', 'PetrPetrenko@mail.ru', '(093)1231212'),

('Ivanenko Ivan Ivanovich', 'Dehtarivska 5', 'Chernigov', 'IvanenkoIvan@gmail.com', '(095)2313244');

INSERT INTO Orders

(CustomerNo, OrderDate, Goods)

VALUES

(1, GETDATE(), 'KeyBoard'),

(2, GETDATE(), 'Mouse'),

(2, GETDATE(), 'WebCam'),

(1, GETDATE(), 'Mouse');

SELECT CustomerNO, CustomerName, Address1, City FROM Customers;

SELECT \* FROM Orders;

-- Не допускается запись в ссылаемый столбец (столбец с FOREIGN KEY) дочерней таблици,

-- значений несуществующих в ссылочном столбце (столбец с PRIMARY KEY) родительской таблицы.

INSERT INTO Orders

(CustomerNo, OrderDate, Goods)

VALUES

(3, GETDATE(), 'Mouse'); -- Ошибка!

DROP TABLE Orders;

GO

-- Создание ограничения внешнего ключа на существующей таблице

CREATE TABLE Orders

(

OrderID int IDENTITY NOT NULL

PRIMARY KEY,

CustomerNo int NOT NULL,

OrderDate date NOT NULL,

Goods varchar(30) NOT NULL

)

ALTER TABLE Orders

ADD CONSTRAINT FK\_CustomersCustomerNo

FOREIGN KEY (CustomerNo) REFERENCES Customers(CustomerNo);

DROP TABLE Customers -- Невозможно удалить таблицу на которую ссылается дочерняя таблица.

-- Первыми удаляются дочерние таблицы после чего удаляются родительские таблицы.

DROP TABLE Orders;

DROP TABLE Customers;

USE ShopDB

GO

CREATE TABLE Customers

(

CustomerNO int IDENTITY NOT NULL

PRIMARY KEY,

CustomerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12)

)

GO

CREATE TABLE Orders

(

OrderID int IDENTITY NOT NULL

PRIMARY KEY,

CustomerNo int NOT NULL,

OrderDate date NOT NULL,

Goods varchar(30) NOT NULL

)

GO

ALTER TABLE Orders

ADD CONSTRAINT FK\_CustomersCustomerNo

FOREIGN KEY (CustomerNo) REFERENCES Customers(CustomerNo);

GO

-- Удаление ограничения внешнего ключа. FK\_CustomersCustomerNo - имя по умолчанию

ALTER TABLE Orders

DROP CONSTRAINT FK\_CustomersCustomerNo;

GO

DROP TABLE Customers; -- Таблица Customers удалена.

GO

USE ShopDB

GO

DROP TABLE Customers;

DROP TABLE Orders;

CREATE TABLE Customers

(

CustomerNO int IDENTITY NOT NULL

PRIMARY KEY,

CustomerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12)

)

GO

CREATE TABLE Orders

(

OrderID int IDENTITY NOT NULL

PRIMARY KEY,

CustomerNo int NULL DEFAULT 2 ,

OrderDate date NOT NULL,

EmployeeId int NULL

)

GO

ALTER TABLE Orders

ADD CONSTRAINT FK\_CustomersCustomerNo

FOREIGN KEY (CustomerNo) REFERENCES Customers(CustomerNo);

INSERT INTO Customers

(CustomerName, Address1, City, Contact, Phone)

VALUES

('Petrenko Petr Petrovich', 'Луганская 25', 'Конотоп', 'PetrPetrenko@mail.ru', '(093)1231212'),

('Ivanenko Ivan Ivanovich', 'Дихтяревская 5', 'Чернигов', 'IvanenkoIvan@gmail.com', '(095)2313244');

INSERT INTO Orders

(CustomerNo, OrderDate, EmployeeId)

VALUES

(1, GETDATE(), 1);

SELECT \* FROM Customers;

SELECT \* FROM Orders;

-- Невозможно удалить запись из родительской таблицы, так как на нее имеется ссылка из дочерней таблицы.

DELETE Customers

WHERE CustomerName = 'Petrenko Petr Petrovich';

-------------------------------------------------------------------------

-- Ограничения ссылочной целостности

-------------------------------------------------------------------------

-- С помощью ограничений ссылочной целостности (ON DELETE, ON UPDATE)

-- можно определять действия, которые SQL Server будет производить

-- над строками дочерней таблицы, когда пользователь попытается

-- удалить или обновить ключевой столбец родительской таблицы.

-- Действия:

-- 1. CASCADE - при UPDATE, DELETE в родительской таблице ключевого значения,

-- в дочерней таблице также выполняется UPDATE на новое значение,

-- или DELETE строк из дочерней таблицы соответственно

ALTER TABLE Orders

DROP CONSTRAINT FK\_CustomersCustomerNo;

GO

ALTER TABLE Orders

ADD CONSTRAINT FK\_CustomersCustomerNo

FOREIGN KEY (CustomerNo) REFERENCES Customers(CustomerNo)

ON DELETE CASCADE

GO

DELETE Customers

WHERE CustomerName = 'Petrenko Petr Petrovich';

SELECT \* FROM Customers;

SELECT \* FROM Orders;

DBCC CHECKIDENT("Customers",RESEED,0); -- Обновление значение IDENTITY = 0

GO

INSERT INTO Customers

(CustomerName, Address1, City, Contact, Phone)

VALUES

('Петренко Петр Петрович', 'Луганская 25', 'Конотоп', 'PetrPetrenko@mail.ru', '(093)1231212')

DBCC CHECKIDENT("Customers",RESEED,2) -- Обновление значение IDENTITY = 2

GO

DBCC CHECKIDENT("Orders",RESEED,0); -- Обновление значение IDENTITY = 0

GO

INSERT INTO Orders

(CustomerNo, OrderDate, EmployeeId)

VALUES (1,GETDATE(),1);

GO

SELECT \* FROM Customers;

SELECT \* FROM Orders;

-- 2. SET NULL - при UPDATE, DELETE в родительской таблице ключевого значения,

-- в дочерней таблице все значения, составляющие эти внешние ключи,

-- будут изменены на NULL

ALTER TABLE Orders

DROP CONSTRAINT FK\_CustomersCustomerNo;

GO

ALTER TABLE Orders

ADD CONSTRAINT FK\_CustomersCustomerNo

FOREIGN KEY (CustomerNo) REFERENCES Customers(CustomerNo)

ON DELETE SET NULL

GO

DELETE Customers

WHERE CustomerName = 'Петренко Петр Петрович';

GO

SELECT \* FROM Customers;

SELECT \* FROM Orders;

DBCC CHECKIDENT("Customers",RESEED,0); -- Обновление значение IDENTITY = 0

GO

INSERT INTO Customers

(CustomerName, Address1, City, Contact, Phone)

VALUES

('Petrenko Petr Petrovich', 'Луганская 25', 'Конотоп', 'PetrPetrenko@mail.ru', '(093)1231212')

DBCC CHECKIDENT("Customers",RESEED,2) -- Обновление значение IDENTITY = 2

GO

DELETE Orders WHERE OrderID = 1;

GO

DBCC CHECKIDENT("Orders",RESEED,0); -- Обновление значение IDENTITY = 0

GO

INSERT INTO Orders

(CustomerNo, OrderDate, EmployeeId)

VALUES (1,GETDATE(),1);

GO

SELECT \* FROM Customers;

SELECT \* FROM Orders;

-- 3. SET DEFAULT - при UPDATE, DELETE в родительской таблице ключевого значения,

-- в дочерней таблице все значения, составляющие эти внешние ключи,

-- будут изменены на значение по умолчанию.

ALTER TABLE Orders

DROP CONSTRAINT FK\_CustomersCustomerNo;

GO

ALTER TABLE Orders

ADD CONSTRAINT FK\_CustomersCustomerNo

FOREIGN KEY (CustomerNo) REFERENCES Customers(CustomerNo)

ON DELETE SET DEFAULT

GO

DELETE Customers

WHERE CustomerName = 'Petrenko Petr Petrovich';

GO

SELECT \* FROM Customers;

SELECT \* FROM Orders;

DBCC CHECKIDENT("Customers",RESEED,0); -- Обновление значение IDENTITY = 0

GO

INSERT INTO Customers

(CustomerName, Address1, City, Contact, Phone)

VALUES

('Petrenko Petr Petrovich', 'Луганская 25', 'Конотоп', 'PetrPetrenko@mail.ru', '(093)1231212')

DBCC CHECKIDENT("Customers",RESEED,2) -- Обновление значение IDENTITY = 2

GO

DELETE Orders WHERE OrderID = 1;

GO

DBCC CHECKIDENT("Orders",RESEED,0); -- Обновление значение IDENTITY = 0

GO

INSERT INTO Orders

(CustomerNo, OrderDate, EmployeeId)

VALUES (1,GETDATE(),1);

GO

SELECT \* FROM Customers;

SELECT \* FROM Orders;

-- 4. NO ACTION (по умолчанию) - запрещает выполнение UPDATE, DELETE!

ALTER TABLE Orders

DROP CONSTRAINT FK\_CustomersCustomerNo;

GO

ALTER TABLE Orders

ADD CONSTRAINT FK\_CustomersCustomerNo

FOREIGN KEY (CustomerNo) REFERENCES Customers(CustomerNo)

ON DELETE NO ACTION

GO

DELETE Customers

WHERE CustomerName = 'Petrenko Petr Petrovich';

SELECT \* FROM Customers;

SELECT \* FROM Orders;

////////////////////////////////////////////////////////////////////

-------------------------------------------------------------------------

/\*Пользовательские ограничения\*/

-------------------------------------------------------------------------

DROP TABLE Orders;

GO

DROP TABLE Customers;

GO

CREATE TABLE Customers

(

CustomerNO int IDENTITY NOT NULL,

CustomerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

Contact varchar(25) NOT NULL,

-- Пользовательское ограничение

Phone char(12) CHECK (Phone LIKE '([0-9][0-9][0-9])[0-9][0-9][0-9][0-9][0-9][0-9][0-9]')

)

GO

INSERT INTO Customers

(CustomerName, Address1, City, Contact, Phone)

VALUES

('Alex', 'NewSTR', 'City', 'dfgjs@mail.ru', '(093)1231212');

INSERT INTO Customers

(CustomerName, Address1, City, Contact, Phone)

VALUES

('Alex', 'NewSTR', 'City', 'dfgjs@mail.ru', '(093)\*\*\*\*\*\*\*'); -- Ошибка.

SELECT \* FROM Customers;

DROP TABLE Customers;

GO

CREATE TABLE Customers

(

CustomerNO int IDENTITY NOT NULL,

CustomerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12) UNIQUE -- UNIQUE (алтернативный ключ), значение в ячейке должно быть уникальное

)

GO

INSERT INTO Customers

(CustomerName, Address1, City, Contact, Phone)

VALUES

('Alex', 'NewSTR', 'City', 'dfgjs@mail.ru', '(093)\*\*\*\*\*\*\*');

INSERT INTO Customers

(CustomerName, Address1, City, Contact, Phone)

VALUES

('Alex', 'NewSTR', 'City', 'dfgjs@mail.ru', '(093)1231213');

SELECT \* FROM Customers;

DROP TABLE Customers;

GO

-- Возможно задать сразу несколько ограничений в произвольном порядке

CREATE TABLE Customers

(

CustomerNO int IDENTITY NOT NULL,

CustomerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12) UNIQUE CHECK (Phone LIKE '([0-9][0-9][0-9])[0-9][0-9][0-9][0-9][0-9][0-9][0-9]')

)

GO

INSERT INTO Customers

(CustomerName, Address1, City, Contact, Phone)

VALUES

('Alex', 'NewSTR', 'City', 'dfgjs@mail.ru', '(093)1231212');

INSERT INTO Customers

(CustomerName, Address1, City, Contact, Phone)

VALUES

('Alex', 'NewSTR', 'City', 'dfgjs@mail.ru', '(093)\*\*\*\*\*\*\*'); -- Ошибка (Не соответствие шаблону).

INSERT INTO Customers

(CustomerName, Address1, City, Contact, Phone)

VALUES

('Alex', 'NewSTR', 'City', 'dfgjs@mail.ru', '(093)1231212'); -- Ошибка (Повторяющееся значение).

SELECT \* FROM Customers;

DROP TABLE Customers;

GO

CREATE TABLE Customers

(

CustomerNO int IDENTITY NOT NULL,

CustomerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12)

)

GO

-- Создание пользовательского ограничения на существующей таблице

ALTER TABLE Customers

ADD CONSTRAINT CN\_CustomersPhoneNo

-- Ограничение CHECK

CHECK (Phone LIKE '([0-9][0-9][0-9])[0-9][0-9][0-9][0-9][0-9][0-9][0-9]')

GO

INSERT INTO Customers

(CustomerName, Address1, City, Contact, Phone)

VALUES

('Alex', 'NewSTR', 'City', 'dfgjs@mail.ru', '(093)1231212');

INSERT INTO Customers

(CustomerName, Address1, City, Contact, Phone)

VALUES

('Alex', 'NewSTR', 'City', 'dfgjs@mail.ru', '(093)\*\*\*\*\*\*\*'); -- Ошибка (Не соответствие шаблону).

-------------------------------------------------------------------------

ALTER TABLE Customers

NOCHECK CONSTRAINT CN\_CustomersPhoneNo; -- Отключение работы ограничения CHECK

INSERT INTO Customers

(CustomerName, Address1, City, Contact, Phone)

VALUES

('Alex', 'NewSTR', 'City', 'dfgjs@mail.ru', '(093)\*\*\*\*\*\*\*');

-------------------------------------------------------------------------

ALTER TABLE Customers

CHECK CONSTRAINT CN\_CustomersPhoneNo; -- Включение работы ограничения CHECK

INSERT INTO Customers

(CustomerName, Address1, City, Contact, Phone)

VALUES

('Alex', 'NewSTR', 'City', 'dfgjs@mail.ru', '(093)\*\*\*\*\*\*\*'); -- Ошибка (Не соответствие шаблону).

--------------------------------------------------------------------------

ALTER TABLE Customers

DROP CONSTRAINT CN\_CustomersPhoneNo; -- Удаление ограничения CHECK

INSERT INTO Customers

(CustomerName, Address1, City, Contact, Phone)

VALUES

('Alex', 'NewSTR', 'City', 'dfgjs@mail.ru', '(093)\*\*\*\*\*\*\*');

-- После удаления не возможен возврат ограничения CHECK

ALTER TABLE Customers

**CHECK CONSTRAINT CN\_CustomersPhoneNo;**

**////////////////////////////////////////////////////////////////////////////////////////////////**

-------------------------------------------------------------------------

-- Связь Один к Одному

-------------------------------------------------------------------------

USE ShopDB

GO

DROP TABLE Orders;

GO

DROP TABLE Customers;

GO

CREATE TABLE Customers

(

CustomerNO int IDENTITY NOT NULL

PRIMARY KEY,

CustomerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12)

)

GO

CREATE TABLE Orders

(

OrderID int IDENTITY NOT NULL

PRIMARY KEY,

CustomerNo int NOT NULL UNIQUE

FOREIGN KEY REFERENCES Customers(CustomerNo), -- Связь Один к Одному.

OrderDate date NOT NULL,

Goods varchar(30) NOT NULL

)

GO

INSERT INTO Customers

(CustomerName, Address1, City, Contact, Phone)

VALUES

('Петренко Петр Петрович', 'Луганская 25', 'Конотоп', 'PetrPetrenko@mail.ru', '(093)1231212'),

('Иваненко Иван Иванович', 'Дихтяревская 5', 'Чернигов', 'IvanenkoIvan@gmail.com', '(095)2313244');

INSERT INTO Orders

(CustomerNo, OrderDate, Goods)

VALUES

(1, GETDATE(), 'KeyBoard'),

(2, GETDATE(), 'Mouse');

SELECT CustomerNO, CustomerName, Address1, City FROM Customers;

SELECT \* FROM Orders;

INSERT INTO Orders

(CustomerNo, OrderDate, Goods)

VALUES

(2, GETDATE(), 'WebCam'),

(1, GETDATE(), 'Mouse'); -- Ошибка

-------------------------------------------------------------------------

-- Связь Один ко Многим

-------------------------------------------------------------------------

DROP TABLE Orders;

GO

DROP TABLE Customers;

GO

CREATE TABLE Customers

(

CustomerNO int IDENTITY NOT NULL

PRIMARY KEY,

CustomerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12)

)

GO

CREATE TABLE Orders

(

OrderID int IDENTITY NOT NULL

PRIMARY KEY,

CustomerNo int NOT NULL

FOREIGN KEY REFERENCES Customers(CustomerNo), -- Связь Один ко Многим.

OrderDate date NOT NULL,

Goods varchar(30) NOT NULL

)

GO

INSERT INTO Customers

(CustomerName, Address1, City, Contact, Phone)

VALUES

('Петренко Петр Петрович', 'Луганская 25', 'Конотоп', 'PetrPetrenko@mail.ru', '(093)1231212'),

('Иваненко Иван Иванович', 'Дихтяревская 5', 'Чернигов', 'IvanenkoIvan@gmail.com', '(095)2313244');

INSERT INTO Orders

(CustomerNo, OrderDate, Goods)

VALUES

(1, GETDATE(), 'KeyBoard'),

(2, GETDATE(), 'Mouse'),

(2, GETDATE(), 'WebCam'),

(1, GETDATE(), 'Mouse');

SELECT CustomerNO, CustomerName, Address1, City FROM Customers;

SELECT \* FROM Orders;

-------------------------------------------------------------------------

-- Связь Многие ко Многим

-------------------------------------------------------------------------

DROP TABLE Orders;

GO

DROP TABLE Customers;

GO

CREATE TABLE Customers

(

CustomerNO int IDENTITY NOT NULL

PRIMARY KEY,

CustomerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12)

)

GO

CREATE TABLE Orders

(

OrderID int IDENTITY NOT NULL

PRIMARY KEY,

CustomerNo int NOT NULL

FOREIGN KEY REFERENCES Customers(CustomerNo), -- Связь Один ко Многим.

OrderDate date NOT NULL,

Goods varchar(30) NOT NULL

)

GO

DROP TABLE CreditCards

GO

CREATE TABLE CreditCards

(

CardID bigint UNIQUE NOT NULL,

CVV int NOT NULL,

CustomerNo int NOT NULL

FOREIGN KEY REFERENCES Customers(CustomerNo) -- Связь Один ко Многим.

)

INSERT INTO Customers

(CustomerName, Address1, City, Contact, Phone)

VALUES

('Петренко Петр Петрович', 'Луганская 25', 'Конотоп', 'PetrPetrenko@mail.ru', '(093)1231212'),

('Иваненко Иван Иванович', 'Дихтяревская 5', 'Чернигов', 'IvanenkoIvan@gmail.com', '(095)2313244');

INSERT INTO Orders

(CustomerNo, OrderDate, Goods)

VALUES

(1, GETDATE(), 'KeyBoard'),

(2, GETDATE(), 'Mouse'),

(2, GETDATE(), 'WebCam'),

(1, GETDATE(), 'Mouse');

INSERT INTO CreditCards

VALUES

(1233431241,232,1),

(2312312349,097,2),

(4323456732,876,2),

(5372168424,937,1),

(7643354521,526,1);

SELECT \* FROM Orders;

SELECT CustomerNO, CustomerName, Address1, City FROM Customers;

SELECT \* FROM CreditCards;

**|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||**

* **Нормализация таблиц** – это формальный аппарат ограничений на формирование таблиц, который позволяет устранить дублирование данных, обеспечивает непротиворечивость хранимых в базе данных, уменьшает трудозатраты на ведение базы данных (ввод и корректировку данных).
*  **Первая нормальная форма (1NF)** – отсутствие повторяющихся данных, любое поле любой записи хранит только одно значение.
*  **Вторая нормальная форма (2NF)** – удовлетворяет первой нормальной форме, и каждый столбец должен зависеть от всего ключа.
*  **Третья нормальная форма (3NF)** – удовлетворяет 2NF, и ни в одном не ключевом столбце не может быть зависимости от другого не ключевого столбца. Наличие в таблице производных данных не допускается.
*  **Денормализация** – процесс понижения нормальной формы. Осуществляется если приведенная высшая форма приводит к ухудшению практического использования.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

USE master

DROP DATABASE ShopDB;

CREATE DATABASE ShopDB;

USE ShopDB

CREATE TABLE Orders

(

Customer varchar(200),

OrderDate date,

OrderDetails varchar(100),

Employee varchar(150),

)

INSERT Orders

VALUES

(

'Василий Петрович Лященко; Харьков, Лужная 15; (092)3212211;',

'2009-12-28',

'LV231 Джинсы, 45$, 5, 225$; DG30 Ремень, 30$, 5, 145$; LV12 Обувь, 26$, 5, 125$;',

'Иван Иванович Белецкий'),

('Зигмунд Федорович Унакий; Киев, Дегтяревская 5; (092)7612343;',

'2010-09-01',

'GC11 Шапка, 32$, 10, 320#$; GC111 Футболка, 20$, 15, 300$;',

'Светлана Олеговна Лялечкина'),

('Олег Евстафьевич Выжлецов; Чернигов, Киевская 5; (044)2134212;',

'2010-09-18',

'LV12 Обувь, 26$, 20, 520$; GC11 Шапка, 32$, 18, 576$;',

'Светлана Олеговна Лялечкина'

)

SELECT \* FROM Orders

-------------------------------------------------------------------------

-- Первая НФ

-------------------------------------------------------------------------

-- Первая нормальная форма (1NF) – подразумевает отсутствие повторяющихся

-- строк в таблице, а так же разбиение сложных значений атомарных данных

-- в ячейке на более простые атомарные данные.

DROP TABLE Orders

CREATE TABLE Orders

(

CustFName varchar(15) NOT NULL,

CustMName varchar(15) NOT NULL,

CustLName varchar(15) NOT NULL,

CustomerCity varchar(15),

CustomerAddress varchar(25),

Phone varchar(12) NOT NULL,

OrderDate date NOT NULL,

ProductID char(5) NOT NULL,

ProductDescription varchar(15),

UnitPrice smallmoney NOT NULL,

Qty int NOT NULL,

TotalPrice money,

EmpFName varchar(15) NOT NULL,

EmpMName varchar(15) NOT NULL,

EmpLName varchar(15) NOT NULL,

CHECK (Phone LIKE '([0-9][0-9][0-9])[0-9][0-9][0-9][0-9][0-9][0-9][0-9]')

)

INSERT Orders

VALUES

('Василий', 'Петрович', 'Лященко', 'Харьков', 'Лужная 15', '(092)3212211',

'2009-12-28',

'LV231', 'Джинсы', 45, 5, 225,

'Иван', 'Иванович', 'Белецкий'),

('Василий', 'Петрович', 'Лященко', 'Харьков', 'Лужная 15', '(092)3212211',

'2009-12-28',

'DG30', 'Ремень', 30, 5, 145,

'Иван', 'Иванович', 'Белецкий'),

('Василий', 'Петрович', 'Лященко', 'Харьков', 'Лужная 15', '(092)3212211',

'2009-12-28',

'LV12', 'Обувь', 26, 5, 125,

'Иван', 'Иванович', 'Белецкий'),

('Зигмунд', 'Федорович', 'Унакий', 'Киев', 'Дегтяревская 5', '(092)7612343',

'2010-09-01',

'GC11', 'Шапка', 32, 10, 320,

'Светлана', 'Олеговна', 'Лялечкина'),

('Зигмунд', 'Федорович', 'Унакий', 'Киев', 'Дегтяревская 5', '(092)7612343',

'2010-09-01',

'GC111', 'Футболка', 20, 15, 300,

'Светлана', 'Олеговна', 'Лялечкина'),

('Олег', 'Увстафьевич', 'Выжлецов', 'Чернигов', 'Киевская 5', '(044)2134212',

'2010-09-18',

'LV12', 'Обувь', 26, 20, 520,

'Светлана', 'Олеговна', 'Лялечкина'

),

('Олег', 'Увстафьевич', 'Выжлецов', 'Чернигов', 'Киевская 5', '(044)2134212',

'2010-09-18',

'GC11', 'Шапка', 32, 18, 576,

'Светлана', 'Олеговна', 'Лялечкина'

)

SELECT \* FROM Orders

-- Перед тем как переходить к рассмотрению второй и третьей нормальной формы следует обеспечить сущностную целостность

-- для таблицы Orders(определить первичный ключ)

DROP TABLE Orders

CREATE TABLE Orders

(

OrderID int NOT NULL,

LineItem int NOT NULL,

OrderDate date NOT NULL,

CustFName varchar(15) NOT NULL,

CustMName varchar(15) NOT NULL,

CustLName varchar(15) NOT NULL,

CustomerCity varchar(15),

CustomerAddress varchar(25),

Phone varchar(12) NOT NULL,

ProductID char(5) NOT NULL,

ProductDescription varchar(15),

UnitPrice smallmoney NOT NULL,

Qty int NOT NULL,

TotalPrice money,

EmpFName varchar(15) NOT NULL,

EmpMName varchar(15) NOT NULL,

EmpLName varchar(15) NOT NULL,

CHECK (Phone LIKE '([0-9][0-9][0-9])[0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),

PRIMARY KEY (OrderId, LineItem)

)

INSERT Orders

VALUES

( 1,1,'2009-12-28',

'Василий', 'Петрович', 'Лященко', 'Харьков', 'Лужная 15', '(092)3212211',

'LV231', 'Джинсы', 45, 5, 225,

'Иван', 'Иванович', 'Белецкий'),

( 1,2,'2009-12-28',

'Василий', 'Петрович', 'Лященко', 'Харьков', 'Лужная 15', '(092)3212211',

'DG30', 'Ремень', 30, 5, 145,

'Иван', 'Иванович', 'Белецкий'),

( 1,3, '2009-12-28',

'Василий', 'Петрович', 'Лященко', 'Харьков', 'Лужная 15', '(092)3212211',

'LV12', 'Обувь', 26, 5, 125,

'Иван', 'Иванович', 'Белецкий'),

( 2,1,'2010-09-01',

'Зигмунд', 'Федорович', 'Унакий', 'Киев', 'Дегтяревская 5', '(092)7612343',

'GC11', 'Шапка', 32, 10, 320,

'Светлана', 'Олеговна', 'Лялечкина'),

( 2,2, '2010-09-01',

'Зигмунд', 'Федорович', 'Унакий', 'Киев', 'Дегтяревская 5', '(092)7612343',

'GC111', 'Футболка', 20, 15, 300,

'Светлана', 'Олеговна', 'Лялечкина'),

( 3,1,'2010-09-18',

'Олег', 'Увстафьевич', 'Выжлецов', 'Чернигов', 'Киевская 5', '(044)2134212',

'LV12', 'Обувь', 26, 20, 520,

'Светлана', 'Олеговна', 'Лялечкина'

),

( 3,2, '2010-09-18',

'Олег', 'Увстафьевич', 'Выжлецов', 'Чернигов', 'Киевская 5', '(044)2134212',

'GC11', 'Шапка', 32, 18, 576,

'Светлана', 'Олеговна', 'Лялечкина'

)

SELECT \* FROM Orders

-------------------------------------------------------------------------

-- Вторая НФ

-------------------------------------------------------------------------

-- Вторая нормальная форма (2NF) – удовлетворяет первой нормальной форме,

-- и каждый столбец должен зависеть от всего ключа.

-- В таблице Orders dсе поля кроме ProductID, ProductDescription, UnitPrice, Qty, TotalPrice зависят не от всего ключа.

DROP TABLE Orders

DROP TABLE Employees

DROP TABLE Customers

DROP TABLE OrderDetails

CREATE TABLE Employees

(

EmployeeID int NOT NULL IDENTITY

PRIMARY KEY,

FName nvarchar(15) NOT NULL,

LName nvarchar(15) NOT NULL,

MName nvarchar(15) NOT NULL,

Salary money NOT NULL,

PriorSalary money NOT NULL,

HireDate date NOT NULL,

TerminationDate date NULL,

ManagerEmpID int NULL

)

GO

CREATE TABLE Customers

(

CustomerNo int NOT NULL IDENTITY

PRIMARY KEY,

FName nvarchar(15) NOT NULL,

LName nvarchar(15) NOT NULL,

MName nvarchar(15) NULL,

Address1 nvarchar(50) NOT NULL,

Address2 nvarchar(50) NULL,

City nchar(10) NOT NULL,

Phone char(12) NOT NULL UNIQUE,

DateInSystem date NULL,

CHECK (Phone LIKE '([0-9][0-9][0-9])[0-9][0-9][0-9][0-9][0-9][0-9][0-9]')

)

GO

CREATE TABLE Orders

(

OrderID int NOT NULL IDENTITY PRIMARY KEY,

OrderDate date NOT NULL,

CustomerNo int,

EmployeeID int,

FOREIGN KEY(CustomerNo) REFERENCES Customers(CustomerNo),

FOREIGN KEY(EmployeeID) REFERENCES Employees(EmployeeID),

)

CREATE TABLE OrderDetails

(

OrderID int NOT NULL,

LineItem int NOT NULL,

ProductID char(5) NOT NULL,

ProductDescription varchar(15),

UnitPrice smallmoney NOT NULL,

Qty int NOT NULL,

TotalPrice as Qty \* UnitPrice,

FOREIGN KEY(OrderID) REFERENCES Orders(OrderID),

PRIMARY KEY (OrderId, LineItem)

)

INSERT Employees

(FName, MName, LName, Salary, PriorSalary, HireDate, TerminationDate, ManagerEmpID)

VALUES

('Василий', 'Петрович', 'Лященко', 5000, 800, '11/20/2009', NULL, NULL),

('Иван', 'Иванович', 'Белецкий', 2000, 0, '11/20/2009', NULL, 1),

('Петр', 'Григорьевич', 'Дяченко', 1000, 0, '11/20/2009', NULL, 2),

('Светлана', 'Олеговна', 'Лялечкина', 800, 0, '11/20/2009', NULL, 2);

GO

INSERT Customers

(LName, FName, MName, Address1, Address2, City, Phone,DateInSystem)

VALUES

('Круковский','Анатолий','Петрович','Лужная 15',NULL,'Харьков','(092)3212211','11/20/2009'),

('Дурнев','Виктор','Викторович','Зелинская 10',NULL,'Киев','(067)4242132','08/03/2010'),

('Унакий','Зигмунд','федорович','Дихтяревская 5',NULL,'Киев','(092)7612343','08/17/2010'),

('Левченко','Виталий','Викторович','Глущенка 5','Драйзера 12','Киев','(053)3456788','08/20/2010'),

('Выжлецов','Олег','Евстафьевич','Киевская 3','Одесская 8','Чернигов','(044)2134212','09/18/2010');

GO

INSERT Orders

VALUES

( '2009-12-28', 1, 2),

( '2010-09-01', 3, 4),

( '2010-09-18', 5, 4)

INSERT OrderDetails

VALUES

( 1, 1, 'LV231', 'Джинсы', 45, 5 ),

( 1, 2, 'DG30', 'Ремень', 30, 5 ),

( 1, 3, 'LV12', 'Обувь', 26, 5 ),

( 2, 1, 'GC11', 'Шапка', 32, 10 ),

( 2, 2, 'GC111', 'Футболка', 20, 15 ),

( 3, 1, 'LV12', 'Обувь', 26, 20 ),

( 3, 2, 'GC11', 'Шапка', 32, 18 )

SELECT \* FROM Customers

SELECT \* FROM Employees

SELECT \* FROM Orders

SELECT \* FROM OrderDetails

-------------------------------------------------------------------------

-- Третья НФ

-------------------------------------------------------------------------

-- Третья нормальная форма (3NF) – удовлетворяет 2NF, и ни в одном не

-- ключевом столбце не может быть зависимости от другого не ключевого

-- столбца. Наличие в таблице производных данных не допускается.

DROP TABLE OrderDetails

DROP TABLE Products

CREATE TABLE Products

(

ProductID char(5) NOT NULL PRIMARY KEY,

[Description] varchar(15),

UnitPrice smallmoney NOT NULL,

[Weight] numeric(18, 0) NULL

)

CREATE TABLE OrderDetails

(

OrderID int NOT NULL,

LineItem int NOT NULL,

ProductID char(5) NOT NULL,

Qty int NOT NULL,

FOREIGN KEY(OrderID) REFERENCES Orders(OrderID),

FOREIGN KEY(ProductID) REFERENCES Products(ProductID),

PRIMARY KEY (OrderId, LineItem)

)

INSERT Products

VALUES

( 'LV231', 'Джинсы', 45, 2),

( 'DG30', 'Ремень', 30, 1),

( 'GC111', 'Футболка', 20, 2),

( 'LV12', 'Обувь', 26, 2),

( 'GC11', 'Шапка', 32, 1)

INSERT OrderDetails

VALUES

( 1, 1, 'LV231', 5 ),

( 1, 2, 'DG30', 5 ),

( 1, 3, 'LV12', 5 ),

( 2, 1, 'GC11', 10 ),

( 2, 2, 'GC111', 15 ),

( 3, 1, 'LV12', 20 ),

( 3, 2, 'GC11', 18 )

SELECT \* FROM Customers

SELECT \* FROM Employees

SELECT \* FROM Orders

SELECT \* FROM OrderDetails

SELECT \* FROM Products

-------------------------------------------------------------------------

-- Денормализация

-------------------------------------------------------------------------

-- Денормализация – процесс понижения нормальной формы.

-- Осуществляется если приведенная высшая форма приводит

-- к ухудшению практического использования

DROP TABLE OrderDetails

CREATE TABLE OrderDetails

(

OrderID int NOT NULL,

LineItem int NOT NULL,

ProductID char(5) NOT NULL,

Qty int NOT NULL,

TotalPrice money,

FOREIGN KEY(OrderID) REFERENCES Orders(OrderID),

FOREIGN KEY(ProductID) REFERENCES Products(ProductID),

PRIMARY KEY (OrderId, LineItem)

)

INSERT OrderDetails

VALUES

( 1, 12, 'LV231', 5, 5 \* (select UnitPrice FROM Products where ProductID = 'LV231')),

( 1, 2, 'DG30', 5, 5 \* (select UnitPrice FROM Products where ProductID = 'DG30')),

( 1, 3, 'LV12', 5, 5 \* (select UnitPrice FROM Products where ProductID = 'LV12')),

( 2, 1, 'GC11', 10, 10 \* (select UnitPrice FROM Products where ProductID = 'GC11') ),

( 2, 2, 'GC111', 15, 15 \* (select UnitPrice FROM Products where ProductID = 'GC111') ),

( 3, 1, 'LV12', 20, 20 \* (select UnitPrice FROM Products where ProductID = 'LV12') ),

( 3, 2, 'GC11', 18, 18 \* (select UnitPrice FROM Products where ProductID = 'GC11') )

SELECT \* FROM OrderDetails

**Вложеннные запросы JOIN**

USE ShopDB

GO

CREATE TABLE JoinTest1

(id\_jt1 int,

name varchar(50));

GO

CREATE TABLE JoinTest2

(id\_jt2 int,

name varchar(50));

GO

INSERT JoinTest1

VALUES (1,'one'),

(2,'two'),

(3,'three'),

(4,'four'),

(5,'five'),

(9,'nine'),

(10,'ten');

INSERT JoinTest2

VALUES (1,'one'),

(2,'two'),

(3,'three'),

(4,'four'),

(5,'five'),

(6,'six'),

(7,'seven'),

(8,'eight');

select \* from JoinTest1;

select \* from JoinTest2;

-------------------------------------------------------------------------

-- INNER JOIN

-------------------------------------------------------------------------

-- INNER JOIN (внутреннее объединение) - объединение, при котором

-- в запросе все записи из таблицы на левой и правой стороне операции

-- INNER JOIN добавляются в результирующий набор записей, при соответствии

-- условию значений в связанных полях.

-- Производим выборку всех данных из объеденения таблиц JoinTest1 и JoinTest2

-- по связующим полям id\_jt1 и id\_jt2.

SELECT \* FROM

JoinTest2 -- Левая таблица (Таблица JoinTest2)

INNER JOIN -- Оператор объединения.

JoinTest1 -- Правая таблица(Таблица JoinTest1)

ON id\_jt1 = id\_jt2; -- Условие объединения при котором значения в сравниваемых ячейках должны совпадать.

GO

-------------------------------------------------------------------------

-- LEFT OUTER JOIN, RIGHT OUTER JOIN

-------------------------------------------------------------------------

-- LEFT OUTER JOIN (левое внешнее объединение) - внешнее объединение, при котором

-- в запросе все записи из таблицы на левой стороне операции LEFT JOIN в

-- инструкции SQL добавляются в результирующий набор записей, даже если в

-- таблице на правой стороне отсутствуют совпадающие значения в связанных полях.

-- Производим выборку всех данных из результирующего набора данных левого внешнего

-- объеденения таблиц JoinTest1 и JoinTest2 по связующим полям id\_jt1 и id\_jt2.

SELECT \* FROM JoinTest2 -- Левая таблица JoinTest2

LEFT OUTER JOIN JoinTest1 -- LEFT JOIN

ON id\_jt1=id\_jt2;

GO

-- RIGHT OUTER JOIN (правое внешнее объединение) - внешнее объединение, при котором

-- в запросе все записи из таблицы на правой стороне операции RIGHT JOIN в

-- инструкции SQL добавляются в результирующий набор записей, даже если в

-- таблице на левой стороне отсутствуют совпадающие значения в связанных полях.

-- Производим выборку всех данных из результирующего набора данных правого внешнего

-- объеденения таблиц JoinTest1 и JoinTest2 по связующим полям id\_jt1 и id\_jt2.

SELECT \* FROM JoinTest2

RIGHT OUTER JOIN JoinTest1 -- Правая таблица JoinTest2

ON id\_jt1 = id\_jt2;

GO

-------------------------------------------------------------------------

-- FULL OUTER JOIN

-------------------------------------------------------------------------

-- FULL OUTER JOIN (полное обединение) внешнее объединение, при котором

-- в запросе все записи из таблицы на левой и правой стороне операции

-- FULL JOIN добавляются в результирующий набор записей, при соотвецтвии

-- условию значений в связанных полях,а так же:

-- - значения из перавой таблицы, не имеющие соответствий в левой таблице;

-- - значения из левой таблицы, не имеющие соответствий в правой таблице.

SELECT \*

FROM JoinTest2

FULL OUTER JOIN JoinTest1 --FULL JOIN

ON id\_jt1 = id\_jt2;

GO

-------------------------------------------------------------------------

-- CROSS JOIN

-------------------------------------------------------------------------

-- CROSS JOIN (перекрестное обединение) - выполняет декартово произведение таблиц,

-- вовлеченных в объединение. В CROSS JOIN не используется консрукция ON.

-- Производим выборку всех данных из результирующего набора данных перекрестного

-- объеденения таблиц JoinTest1 и JoinTest2.

SELECT \* FROM JoinTest1

CROSS JOIN JoinTest2 -- CROSS JOIN

-- ON - не используется

GO

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

USE ShopDB

GO

-------------------------------------------------------------------------

-- UNION

-------------------------------------------------------------------------

-- UNION объединяет результаты двух запросов SELECT в единую результирующую таблицу.

-- Если результаты обоих запросов содержат строки с совпадающими значениями ячеек,

-- то, операция UNION помещает в результирующую таблицу только одну такую строку.

-- Если в результате одного из запросов имеются строки с уникальными значениями,

-- не совпадающими ни с одной из строк результата другого запроса, то эта строка

-- так-же помещается в результирующую таблицу.

-- Операция UNION требует использования таких запросов, каждый из которых возвращает

-- выборку в табличном представлении, при этом, типы и количество столбцов должны совпадать.

SELECT \* FROM JoinTest1

UNION

SELECT \* FROM JoinTest2

-------------------------------------------------------------------------

-- UNION ALL

-------------------------------------------------------------------------

-- UNION ALL объединяет результаты двух запросов SELECT в единую результирующую таблицу.

-- Если результаты обоих запросов содержат строки с совпадающими значениями ячеек,

-- то, операция UNION ALL помещает в результирующую таблицу все дублирующиеся строки.

-- Если в результате одного из запросов имеются строки с уникальными значениями,

-- не совпадающими ни с одной из строк результата другого запроса, то эта строка

-- так-же помещается в результирующую таблицу.

-- Операция UNION ALL требует использования таких запросов, каждый из которых возвращает

-- выборку в табличном представлении, при этом, типы и количество столбцов должны совпадать.

SELECT \* FROM JoinTest1

UNION ALL

SELECT \* FROM JoinTest2

-------------------------------------------------------------------------

-- EXCEPT

-------------------------------------------------------------------------

-- Операция EXCEPT исключает результаты правого запроса.

-- Если результат левого запроса операции EXCEPT содержит уникальные строки, не совпадающие

-- ни с одной из строк правого запроса, то, только такие строки помещается в результирующую таблицу.

-- Уникальные строки правого запроса операции EXCEPT, никогда не входят в результирующую таблицу.

-- Если результаты обоих запросов содержат совпадающие строки, то, операция EXCEPT игнорирует их.

-- Операция EXCEPT требует использования таких запросов, каждый из которых возвращает

-- выборку в табличном представлении, при этом, типы и количество столбцов должны совпадать.

SELECT \* FROM JoinTest1

EXCEPT

SELECT \* FROM JoinTest2

SELECT \* FROM JoinTest2

EXCEPT

SELECT \* FROM JoinTest1

-------------------------------------------------------------------------

-- INTERSECT

-------------------------------------------------------------------------

-- INTERSECT объединяет результаты двух запросов SELECT в единую результирующую таблицу.

-- Если результаты обоих запросов содержат строки с совпадающими значениями ячеек,

-- то, операция INTERSECT помещает в результирующую таблицу только одну такую строку.

-- Если в результате одного из запросов имеются уникальные строки,

-- не совпадающие ни с одной из строк результата другого запроса,

-- то такие строки игнорируются операцией INTERSECT.

-- Операция INTERSECT требует использования таких запросов, каждый из которых возвращает

-- выборку в табличном представлении, при этом, типы и количество столбцов должны совпадать.

SELECT \* FROM JoinTest1

INTERSECT

SELECT \* FROM JoinTest2

USE ShopDB

GO

-- Показать статистику по количеству проданых единиц товара.

SELECT Products.ProdID, [Description], SUM(Qty) AS Qty, SUM(TotalPrice) AS TotalSold -- Ошибки нет.

FROM Products

INNER JOIN OrderDetails

ON Products.ProdID = OrderDetails.ProdID

GROUP BY Products.ProdID, [Description]

-- Если в используемых таблицах, имеются столбцы с одинаковыми именами,

-- то нужно явно указать к какой таблице относится данный столбец.

SELECT ProdID, [Description], Qty, TotalPrice FROM Products -- Ошибка!

JOIN OrderDetails

ON Products.ProdID = OrderDetails.ProdID

-- Вывести общую суму продаж по сотрудикам.

SELECT FName, LName, MName, SUM(TotalPrice) AS [Total Sold] FROM Employees -- Алиас [Total Sold]

JOIN Orders

ON Employees.EmployeeID = Orders.EmployeeID

JOIN OrderDetails

ON Orders.OrderID = OrderDetails.OrderID

GROUP BY Employees.FName,

Employees.LName,

Employees.MName

-- Вывести общую суму продаж по каждому из сотрудиков

SELECT FName, LName, MName, SUM(TotalPrice) AS [Total Sold] FROM Employees -- Алиас [Total Sold]

LEFT JOIN Orders

ON Employees.EmployeeID = Orders.EmployeeID

LEFT JOIN OrderDetails

ON Orders.OrderID = OrderDetails.OrderID

GROUP BY Employees.FName,

Employees.LName,

Employees.MName

-- Вывести план подченения сотрудников (Кто кому подчиняется)

SELECT Emp1.FName, Emp1.MName, Emp1.LName, Emp2.FName, Emp2.MName, Emp2.LName

FROM Employees Emp1 -- Алиас Emp1

JOIN Employees Emp2 -- Алиас Emp2

ON Emp1.EmployeeID = Emp2.ManagerEmpID

////////////////////////////////////////////////////////////////////////////////////////////

**Индексирование**

USE ShopDB

GO

DROP TABLE BTree;

GO

CREATE TABLE BTree

(

numbers int ,

word varchar(25)

)

GO

INSERT BTree

VALUES

(1,'one'),

(2,'two'),

(3,'three'),

(4,'four'),

(6,'six'),

(7,'seven'),

(8,'eight'),

(9,'nine');

SELECT \* FROM BTree;

INSERT BTree

VALUES (5,'five');

SELECT \* FROM BTree WHERE numbers IN (5,6);

--------------------------------------------------

DROP TABLE BTree;

GO

CREATE TABLE BTree

(

numbers int primary key, -- Первичный ключ, добавляет кластеризованый индекс

word varchar(25)

)

--SET STATISTICS IO ON

INSERT BTree

VALUES

(1,'one'),

(2,'two'),

(3,'three'),

(4,'four'),

(6,'six'),

(7,'seven'),

(8,'eight'),

(9,'nine');

SELECT \* FROM BTree WHERE word = 'one';

SELECT \* FROM sys.indexes

WHERE object\_id = (SELECT object\_id FROM sys.tables

WHERE name = 'BTree')

INSERT BTree

VALUES (5,'five');

SELECT \* FROM BTree;

--------------------------------------------------

DROP TABLE BTree;

GO

CREATE TABLE BTree

(

numbers int ,

word varchar(25)primary key -- кластеризированный индекс

)

INSERT BTree

VALUES

(1,'one'),

(2,'two'),

(3,'three'),

(4,'four'),

(6,'six'),

(7,'seven'),

(8,'eight'),

(9,'nine');

SELECT \* FROM BTree;

INSERT BTree

VALUES (5,'five');

SELECT \* FROM BTree;

--------------------------------------------------

DROP TABLE BTree;

GO

CREATE TABLE BTree

(

numbers int unique, -- некластеризированный индекс на не упорядоченой таблице (данные хранятся не на листевом уровне дерева)

word varchar(25)

)

INSERT BTree

VALUES

(1,'one'),

(2,'two'),

(3,'three'),

(4,'four'),

(6,'six'),

(7,'seven'),

(8,'eight'),

(9,'nine');

SELECT \* FROM BTree;

INSERT BTree

VALUES (5,'five');

SELECT \* FROM BTree where word = 'one';

--------------------------------------------------

DROP TABLE BTree;

GO

CREATE TABLE BTree

(

numbers int primary key, -- кластеризированый индекс

word varchar(25)unique -- некластеризированный индекс на клатеризованной таблице

)

INSERT BTree

VALUES

(1,'one'),

(2,'two'),

(3,'three'),

(4,'four'),

(6,'six'),

(7,'seven'),

(8,'eight'),

(9,'nine');

SELECT \* FROM BTree;

INSERT BTree

VALUES (5,'five');

SELECT \* FROM BTree where word in ('five', 'one'); -- поиск по некластеризованному индексу

SELECT \* FROM BTree where numbers in (3,5); -- поиск по кластеризованному индексу

--------------------------------------------------

DROP TABLE OrderDetails;

GO

DROP TABLE Orders;

GO

DROP TABLE Customers

GO

CREATE TABLE Customers

(

CustumerNo int NOT NULL,

CustumerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

Address2 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

State char(2) NOT NULL,

Zip varchar(10) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12),

FedIDNo varchar(10) NOT NULL,

DateInSystem smalldatetime NOT NULL

)

GO

INSERT INTO Customers

( CustumerNo, CustumerName, Address1, Address2, City, State, Zip, Contact, Phone, FedIDNo, DateInSystem)

VALUES

(1,'Alex', 'NewSTR', 'NewSTR2', 'City', 'NS', 'NewZip', 'dfgjs@mail.ru', '(093)1231212', 'qweq', GETDATE()),

(2,'Alex2', 'NewSTR2', 'NewSTR22', 'City2', 'SN', 'NewZip2', 'dfg2@mail.ru', '(093)2221212', 'qwq2', GETDATE());

GO

SELECT \* FROM Customers;

-----------------------------------------------------------------------------------------------------------------

DROP TABLE Customers;

GO

CREATE TABLE Customers

(

CustumerNo int NOT NULL PRIMARY KEY, -- кластеризованный индекс

CustumerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

Address2 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

State char(2) NOT NULL,

Zip varchar(10) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12),

FedIDNo varchar(10) NOT NULL,

DateInSystem smalldatetime NOT NULL

)

GO

INSERT INTO Customers

( CustumerNo, CustumerName, Address1, Address2, City, State, Zip, Contact, Phone, FedIDNo, DateInSystem)

VALUES

(1,'Alex', 'NewSTR', 'NewSTR2', 'City', 'NS', 'NewZip', 'dfgjs@mail.ru', '(093)1231212', 'qweq', GETDATE()),

(2,'Alex2', 'NewSTR2', 'NewSTR22', 'City2', 'SN', 'NewZip2', 'dfg2@mail.ru', '(093)2221212', 'qwq2', GETDATE());

SELECT \* FROM Customers; -- для поиска используем кластеризованный индекс

-----------------------------------------------------------------------------------------------------------------

DROP TABLE Customers;

GO

CREATE TABLE Customers

(

CustumerNo int NOT NULL,

CustumerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

Address2 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

State char(2) NOT NULL,

Zip varchar(10) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12) UNIQUE, -- некластеризованный индекс

FedIDNo varchar(10) NOT NULL,

DateInSystem smalldatetime NOT NULL

)

GO

INSERT INTO Customers

( CustumerNo, CustumerName, Address1, Address2, City, State, Zip, Contact, Phone, FedIDNo, DateInSystem)

VALUES

(1,'Alex', 'NewSTR', 'NewSTR2', 'City', 'NS', 'NewZip', 'dfgjs@mail.ru', '(093)1231212', 'qweq', GETDATE()),

(2,'Alex2', 'NewSTR2', 'NewSTR22', 'City2', 'SN', 'NewZip2', 'dfg2@mail.ru', '(093)2221212', 'qwq2', GETDATE());

GO

SELECT \* FROM Customers;

SELECT Phone FROM Customers

WHERE Phone = '(093)2221212'; -- Для поиска используем некластеризованный

-----------------------------------------------------------------------------------------------------------------

DROP TABLE Customers;

GO

CREATE TABLE Customers

(

CustumerNo int IDENTITY NOT NULL

PRIMARY KEY, -- кластеризованный индекс

CustumerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

Address2 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

State char(2) NOT NULL,

Zip varchar(10) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12) UNIQUE, -- некластеризованный индекс

FedIDNo varchar(10) NOT NULL,

DateInSystem smalldatetime NOT NULL

)

GO

INSERT INTO Customers

( CustumerName, Address1, Address2, City, State, Zip, Contact, Phone, FedIDNo, DateInSystem)

VALUES

('Alex', 'NewSTR', 'NewSTR2', 'City', 'NS', 'NewZip', 'dfgjs@mail.ru', '(093)1231212', 'qweq', GETDATE()),

('Alex2', 'NewSTR2', 'NewSTR22', 'City2', 'SN', 'NewZip2', 'dfg2@mail.ru', '(093)2221212', 'qwq2', GETDATE());

SELECT \* FROM Customers; -- для поиска используем только кластеризованный индекс

SELECT \* FROM Customers -- Для поиска используем кластеризованный

WHERE Phone = '(093)2221212'; -- и не кластеризированный индексы !!!

------------------------- Кластеризованный индекс -------------------------------------------------------

DROP TABLE Customers;

GO

CREATE TABLE Customers

(

CustumerNo int NOT NULL,

CustumerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

Address2 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

State char(2) NOT NULL,

Zip varchar(10) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12),

FedIDNo varchar(10) NOT NULL,

DateInSystem smalldatetime NOT NULL

)

GO

CREATE CLUSTERED INDEX PK\_IND1 -- создание собственного кластеризованного индекса

ON Customers(CustumerNo);

GO

INSERT INTO Customers

( CustumerNo, CustumerName, Address1, Address2, City, State, Zip, Contact, Phone, FedIDNo, DateInSystem)

VALUES

(1,'Alex', 'NewSTR', 'NewSTR2', 'City', 'NS', 'NewZip', 'dfgjs@mail.ru', '(093)1231212', 'qweq', GETDATE()),

(1,'Alex2', 'NewSTR2', 'NewSTR22', 'City2', 'SN', 'NewZip2', 'dfg2@mail.ru', '(093)2221212', 'qwq2', GETDATE()); -- CustumerNo = 1 как и в первой строке

GO

SELECT \* FROM Customers;-- для поиска используем кластеризованный индекс

-----------------------Кластеризованный индекс -----------------------------------------------------------

DROP TABLE Customers;

GO

CREATE TABLE Customers

(

CustumerNo int NOT NULL,

CustumerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

Address2 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

State char(2) NOT NULL,

Zip varchar(10) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12) Primary key,

FedIDNo varchar(10) NOT NULL,

DateInSystem smalldatetime NOT NULL

)

GO

CREATE UNIQUE CLUSTERED INDEX PK\_IND1 -- создание собственного кластеризованного индекса

ON Customers(CustumerNo);

GO

INSERT INTO Customers

( CustumerNo, CustumerName, Address1, Address2, City, State, Zip, Contact, Phone, FedIDNo, DateInSystem)

VALUES

(1,'Alex', 'NewSTR', 'NewSTR2', 'City', 'NS', 'NewZip', 'dfgjs@mail.ru', '(093)1231212', 'qweq', GETDATE()),

(1,'Alex2', 'NewSTR2', 'NewSTR22', 'City2', 'SN', 'NewZip2', 'dfg2@mail.ru', '(093)2221212', 'qwq2', GETDATE()); -- CustumerNo = 1 будет ошибка

GO

INSERT INTO Customers

( CustumerNo, CustumerName, Address1, Address2, City, State, Zip, Contact, Phone, FedIDNo, DateInSystem)

VALUES

(1,'Alex', 'NewSTR', 'NewSTR2', 'City', 'NS', 'NewZip', 'dfgjs@mail.ru', '(093)1231212', 'qweq', GETDATE()),

(2,'Alex2', 'NewSTR2', 'NewSTR22', 'City2', 'SN', 'NewZip2', 'dfg2@mail.ru', '(093)2221212', 'qwq2', GETDATE()); --CustumerNo должен быть уникальным для каждой строки если CREATE UNIQUE CLUSTERED INDEX

GO

SELECT \* FROM Customers; --для поиска используем кластеризованный индекс

------------------ Создание некластеризованного индекса на неупорядоченой таблице -------------------------------------------------

DROP TABLE Customers;

GO

CREATE TABLE Customers

(

CustumerNo int NOT NULL,

CustumerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

Address2 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

State char(2) NOT NULL,

Zip varchar(10) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12),

FedIDNo varchar(10) NOT NULL,

DateInSystem smalldatetime NOT NULL

)

GO

CREATE UNIQUE NONCLUSTERED INDEX NC\_IND1 -- создание собственного некластеризованного индекса на неупорядоченной таблице

ON Customers(Phone);

GO

INSERT INTO Customers

( CustumerNo, CustumerName, Address1, Address2, City, State, Zip, Contact, Phone, FedIDNo, DateInSystem)

VALUES

(1,'Alex', 'NewSTR', 'NewSTR2', 'City', 'NS', 'NewZip', 'dfgjs@mail.ru', '(093)1111111', 'qweq', GETDATE()),

(2,'Alex2', 'NewSTR2', 'NewSTR22', 'City2', 'SN', 'NewZip2', 'dfg2@mail.ru', '(093)2222222', 'qwq2', GETDATE()); --CustumerNo должен быть уникальным для каждой строки если CREATE UNIQUE CLUSTERED INDEX

GO

SELECT \* FROM Customers;

SELECT \* FROM Customers

WHERE Phone = '(093)2222222';

----------------Создание некластеризованного индекса на кластеризованной таблице--------------------------------------

DROP TABLE Customers;

GO

CREATE TABLE Customers

(

CustumerNo int NOT NULL,

CustumerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

Address2 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

State char(2) NOT NULL,

Zip varchar(10) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12),

FedIDNo varchar(10) NOT NULL,

DateInSystem smalldatetime NOT NULL

)

GO

CREATE UNIQUE CLUSTERED INDEX PK\_IND1 -- создание собственного кластеризованного индекса

ON Customers(CustumerNo);

GO

CREATE UNIQUE NONCLUSTERED INDEX NC\_IND1 -- создание собственного некластеризованного индекса на кластеризованной таблице

ON Customers(Phone);

GO

INSERT INTO Customers

( CustumerNo, CustumerName, Address1, Address2, City, State, Zip, Contact, Phone, FedIDNo, DateInSystem)

VALUES

(1,'Alex', 'NewSTR', 'NewSTR2', 'City', 'NS', 'NewZip', 'dfgjs@mail.ru', '(093)1231212', 'qweq', GETDATE()),

(2,'Alex2', 'NewSTR2', 'NewSTR22', 'City2', 'SN', 'NewZip2', 'dfg2@mail.ru', '(093)2221212', 'qwq2', GETDATE()); --CustumerNo должен быть уникальным для каждой строки если CREATE UNIQUE CLUSTERED INDEX

GO

SELECT \* FROM Customers; -- Для поиска используем кластеризованный индекс

SELECT \* FROM Customers -- Для поиска используем кластеризованный

WHERE Phone = '(093)2221212'; -- и не кластеризированный индексы !!!

-----------------------------------------------------------

/\* Изменения индекса ALTER INDEX \*/

-----------------------------------------------------------

DROP TABLE Customers;

GO

CREATE TABLE Customers

(

CustumerNo int NOT NULL,

CustumerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

Address2 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

State char(2) NOT NULL,

Zip varchar(10) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12),

FedIDNo varchar(10) NOT NULL,

DateInSystem smalldatetime NOT NULL

)

GO

CREATE UNIQUE INDEX PK\_IND1 --если включена опция IGNORE\_DUP\_KEY и нету ограничения UNIQUE будет предупреждение (возможна вставка одинаковых индексов Alex и alex)

ON Customers(CustumerName)

WITH

(

IGNORE\_DUP\_KEY = ON -- включаем игнорирование различия между строчными и заглавными

);

GO

INSERT INTO Customers

( CustumerNo, CustumerName, Address1, Address2, City, State, Zip, Contact, Phone, FedIDNo, DateInSystem)

VALUES

(1,'Alex', 'NewSTR', 'NewSTR2', 'City', 'NS', 'NewZip', 'dfgjs@mail.ru', '(093)1231212', 'qweq', GETDATE()),

(2,'alex', 'NewSTR2', 'NewSTR22', 'City2', 'SN', 'NewZip2', 'dfg2@mail.ru', '(093)2221212', 'qwq2', GETDATE()); -- Не будет ошибки

GO

SELECT \* FROM Customers

ALTER INDEX PK\_IND1 -- изменение индекса PK\_IND1

ON Customers -- заданого на таблице Customers

REBUILD WITH

(

IGNORE\_DUP\_KEY = OFF

);

INSERT INTO Customers

( CustumerNo, CustumerName, Address1, Address2, City, State, Zip, Contact, Phone, FedIDNo, DateInSystem)

VALUES

(1,'Alex', 'NewSTR', 'NewSTR2', 'City', 'NS', 'NewZip', 'dfgjs@mail.ru', '(093)1231212', 'qweq', GETDATE()),

(2,'alex', 'NewSTR2', 'NewSTR22', 'City2', 'SN', 'NewZip2', 'dfg2@mail.ru', '(093)2221212', 'qwq2', GETDATE()); -- Не будет ошибки

GO

SELECT \* FROM Customers;

----------------------------УДАЛЕНИЕ ИНДЕКСОВ---------------------------------------------------------------------

DROP TABLE Customers;

GO

CREATE TABLE Customers

(

CustumerNo int NOT NULL,

CustumerName varchar(25) NOT NULL,

Address1 varchar(25) NOT NULL,

Address2 varchar(25) NOT NULL,

City varchar(15) NOT NULL,

State char(2) NOT NULL,

Zip varchar(10) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char(12),

FedIDNo varchar(10) NOT NULL,

DateInSystem smalldatetime NOT NULL

)

GO

CREATE UNIQUE INDEX PK\_IND1 -- если включена опция IGNORE\_DUP\_KEY и нет ограничения UNIQUE будет предупреждения (возможна вставка одинаковых индексов Alex и alex)

ON Customers(CustumerName)

WITH

(

IGNORE\_DUP\_KEY = ON -- включаем игнорирование различия между строчными и заглавными символами

);

GO

INSERT INTO Customers

( CustumerNo, CustumerName, Address1, Address2, City, State, Zip, Contact, Phone, FedIDNo, DateInSystem)

VALUES

(1,'Alex', 'NewSTR', 'NewSTR2', 'City', 'NS', 'NewZip', 'dfgjs@mail.ru', '(093)1231212', 'qweq', GETDATE()),

(2,'alex', 'NewSTR2', 'NewSTR22', 'City2', 'SN', 'NewZip2', 'dfg2@mail.ru', '(093)2221212', 'qwq2', GETDATE()); -- если включена опция IGNORE\_DUP\_KEY и есть ограничения UNIQUE данная строка не запишется

GO

SELECT \* FROM Customers WHERE CustumerName = 'alex'; -- для поиска используем не кластеризованный индекс

DROP INDEX Customers.PK\_IND1;

SELECT \* FROM Customers WHERE CustumerName = 'alex'; -- для поиска не используем индекс

**VIEW**

USE ShopDB

GO

-- Представления это способ вывода ограниченного набора столбцов из реальной таблицы в виде виртуальной таблицы.

DROP TABLE InfoPerson

GO

CREATE TABLE InfoPerson

(

FirstName nvarchar(50) null,

LastName nvarchar(50)null,

BirthDate date null,

AddressLine1 nvarchar(50) null,

AddressLine2 nvarchar(50) null,

EmailAddress nvarchar(50) null,

)

GO

INSERT InfoPerson -- пререносим данные из одной таблицы в другую

SELECT pc.FirstName, pc.LastName, CAST(he.BirthDate as DATE), pa.AddressLine1, pa.AddressLine2, pea.EmailAddress

FROM AdventureWorks2012.Person.Person as pc -- базаДанных.Схема.Таблица AS алиас

JOIN AdventureWorks2012.HumanResources.Employee as he

ON pc.BusinessEntityID = he.BusinessEntityID

JOIN AdventureWorks2012.Person.BusinessEntityAddress as hea

ON he.BusinessEntityID = hea.BusinessEntityID

JOIN AdventureWorks2012.Person.Address as pa

ON hea.AddressID = pa.AddressID

JOIN AdventureWorks2012.Person.EmailAddress as pea

ON pea.BusinessEntityID = pc.BusinessEntityID

GO

SELECT \* FROM InfoPerson;

GO

-----------------------------------------------------------

/\* Создание представления - CREATE VIEW \*/

-----------------------------------------------------------

CREATE VIEW BthPerson -- Создание представления

AS SELECT FirstName, LastName, BirthDate, EmailAddress

FROM InfoPerson;

GO

SELECT \* FROM BthPerson; -- выборка из представления

SELECT \* FROM InfoPerson; -- выборка из таблицы

GO

------------------------------------------------------------

CREATE VIEW BthPerson2

WITH SCHEMABINDING -- Запрещает удаление таблицы для которой создано представление.

AS SELECT FirstName, LastName, BirthDate, EmailAddress

FROM dbo.InfoPerson; -- Двухкомпонентная структура имени таблицы с которой связано представление.

GO

SELECT \* FROM BthPerson2;

DROP TABLE InfoPerson; -- Будет ошибка.

GO

------------------------------------------------------------

CREATE VIEW BthPrsn2Addres -- Создаем представление для выборки данных по условию.

AS SELECT FirstName, LastName, BirthDate, AddressLine1, AddressLine2

FROM InfoPerson

WHERE AddressLine2 is not null; -- Представление будет выводить информацию о тех людях которые имеют 2 адреса

GO

SELECT \* FROM BthPrsn2Addres; -- выборка из представления

SELECT FirstName, LastName, BirthDate, AddressLine1 FROM BthPrsn2Addres; -- как и при выборке из таблицы можно задавать определенный набор столбоцов для вывода

GO

-- Вместо таблицы InfoPerson, в которую записана информация из других таблиц

-- можно создать представление, при этом будет сэкономленна физическая память базы данных.

CREATE VIEW InfoPerson2 AS

SELECT pc.FirstName, pc.LastName, he.BirthDate, pa.AddressLine1, pa.AddressLine2, pea.EmailAddress

FROM AdventureWorks2012.Person.Person as pc -- базаДанных.Схема.Таблица AS алиас

JOIN AdventureWorks2012.HumanResources.Employee as he

ON pc.BusinessEntityID = he.BusinessEntityID

JOIN AdventureWorks2012.Person.BusinessEntityAddress as hea

ON he.BusinessEntityID = hea.BusinessEntityID

JOIN AdventureWorks2012.Person.Address as pa

ON hea.AddressID = pa.AddressID

JOIN AdventureWorks2012.Person.EmailAddress as pea

ON pea.BusinessEntityID = pc.BusinessEntityID

SELECT \* FROM InfoPerson;

SELECT \* FROM InfoPerson2;

-------------- Вставка данных с помощю представления.

SELECT \* FROM BthPerson;

INSERT INTO BthPerson

(BirthDate, EmailAddress, FirstName, LastName)

VALUES

('02/03/1990', 'adgkj@mail.ru', 'Andrew', 'Test1');

-- Выборка из представления.

SELECT \* FROM BthPerson WHERE LastName = 'Test1';

-- Выборка из таблицы.

-- На самом деле данные записались в таблицу InfoPerson.

SELECT \* FROM InfoPerson WHERE LastName = 'Test1';

------------------------------------------------------------

-- Обновление данных с помощью представления.

UPDATE BthPerson

SET EmailAddress = 'testmail1@mail.ru'

WHERE LastName = 'Test1';

SELECT \* FROM BthPerson

WHERE LastName = 'Test1';

SELECT \* FROM InfoPerson

WHERE LastName = 'Test1';

-- Так же данные возможно обновлять и в таблице

UPDATE InfoPerson

SET EmailAddress = 'testmail2@mail.ru'

WHERE LastName='Test1';

SELECT \* FROM BthPerson

WHERE LastName='Test1';

SELECT \* FROM InfoPerson

WHERE LastName='Test1';

-----------------------ИЗМЕНЕНИЕ ПРЕДСТАВЛЕНИЙ--------------

SELECT \* FROM BthPerson;

ALTER VIEW BthPerson -- Изменение представления.

AS SELECT FirstName, LastName, BirthDate, EmailAddress , AddressLine1

FROM InfoPerson;

GO

SELECT \* FROM BthPerson;

----------------------- УДАЛЕНИЕ ПРЕДСТАВЛЕНИЙ ---------------

DROP VIEW BthPerson;

-- Представления больше нет.

SELECT \* FROM BthPerson

WHERE LastName = 'Test1';

-- Данные которые добавлялись через представление остаются в таблице.

SELECT \* FROM InfoPerson

WHERE LastName = 'Test1';

**Хранимые процедуры**

Хранимые процедуры

Хранимыепроцедурываналогичныпроцедурамвдругихязыках программирования:

* •ониобрабатываютвходныепараметрыивозвращаютвызывающей процедуреилипакетузначенияввидевыходныхпараметров;
* •онисодержатпрограммныеинструкции,которыевыполняют операциивбазеданных,втомчислевызывающиедругиепроцедуры;
* •онивозвращаютзначениесостояниявызывающейпроцедуреили пакету,такимобразомпередаваясведенияобуспешномили неуспешномзавершении(ипричиныпоследнего).te Procedure

Создание хранимой процедуры

CREATEPROCspEmployee

--Создание хранимой процедуры.

AS

SELECT\* FROMEmployee;

GO

EXECspEmployee; --Вызов хранимой процедуры.

Пользовательские функции

Определяемаяпользователемфункцияпредставляетсобой подпрограмму,котораяпринимаетпараметры,выполняет действия,такиекаксложныевычисления,азатемвозвращает результатэтихдействийввидезначения.

Возвращаемоезначениеможетбытьскалярнымзначениемили таблицей.

Создание пользовательской функции

CREATEFUNCTIONHello() --создать функцию

RETURNSvarchar(30)

--объявляем тип возвращаемого значения

AS

BEGIN--начало тела функции

DECLARE@MyVarvarchar(20) ='Hello World!';

RETURN@MyVar; --возвращаемое значение функции

END; --конец тела функции

GO

PRINTdbo.Hello();

/\*Операторы управления ходом выполнения\*/

USE AdventureWorks2012;

--------------------------------------------------------------------------

-- Условная конструкция IF..ELSE

--------------------------------------------------------------------------

DECLARE @myVar varchar(10);

--SET @myVar = 'Hello World!!!'

-- В условной конструкции IF, указываем выражение: @myVar is NULL

IF @myVar is NULL

PRINT ('значение myVar не задано'); -- Если условие удовлетворяет истинности.

ELSE

PRINT @myVar; -- ИНАЧЕ Если условие не удовлетворяет истинности.

--------------------------------------------------------------------------

-- Операция EXISTS

--------------------------------------------------------------------------

-- Операция EXISTS возвращает значение TRUE или FALSE, в зависимости от того

-- имеется ли соответствующие запросу SELECT данные

IF EXISTS (SELECT \* FROM Person.Address WHERE City = 'Kyiv') -- В имени таблицы указываем имя схемы (аналог пространства имен)

PRINT 'We have staff from the city Seattle'

ELSE

PRINT 'We have no employees of the city Seattle'

----------------------------------------------------------------------------

USE ShopDB;

-- Системные таблицы sys.schemas, sys.tables

-- таблица sys.schemas содержит информацию о схемах применямых в даной БД

SELECT \* FROM sys.schemas

-- таблица sys.tables содержит информацию о таблицах применямых в даной БД

SELECT \* FROM sys.tables

----------------------------------------------------------------------------

--пример создания таблицы TestTable с предпроверкой не создана ли она ранее.

IF NOT EXISTS (

SELECT s.name,t.name

FROM sys.schemas AS s -- schemas - системная таблица хранящая информацию о схемах

JOIN sys.tables AS t -- tables - системная таблица хранящая информацию о таблицах

ON s.schema\_id = t.schema\_id -- выборка таблицы находящейся в схеме

WHERE s.name = 'dbo' -- где схема dbo

AND t.name = 'TestTable' -- и таблица TestTable

)

CREATE TABLE TestTable --создать таблицу если она не существует

(

Col1 int,

Col2 varchar(20)

)

ELSE

PRINT 'Таблица TestTable уже существует!' -- вывести сообщение если существует

----------------------------------------------------------------------------

----- групировка операторов в блоки -----

DROP TABLE TestTable;

-- Если необходимо выполнить несколько операторов, их обеденяют в блоки BEGIN ... END

IF NOT EXISTS (

SELECT s.name,t.name

FROM sys.schemas s

JOIN sys.tables t

ON s.schema\_id=t.schema\_id

WHERE s.name = 'dbo'

AND t.name = 'TestTable'

)

BEGIN -- Начало блока

PRINT 'Таблица TestTable не найдена';

PRINT 'Создаю таблицу TestTable';

CREATE TABLE TestTable -- Создать таблицу если она не существует

(

Col1 int,

Col2 varchar(20)

)

END -- Конец блока

ELSE

BEGIN

PRINT 'Таблица TestTable существует!';

PRINT 'Удаляю таблицу TestTable';

DROP TABLE TestTable;

PRINT 'Таблица TestTable удалена';

END;

GO

----------------------------------------------------------------------------

-- Оператор CASE

----------------------------------------------------------------------------

-- Простой оператор CASE

DECLARE @myTinyVar TinyInt =3 ;

PRINT CASE @myTinyVar -- входное выражение для CASE

WHEN 0 THEN 'zero' -- если @myIntVar = 0 то выводим 'zero'

WHEN 1 THEN 'One' -- если @myIntVar = 1 то выводим 'One'

WHEN 2 THEN 'Two' -- если @myIntVar = 2 то выводим 'Two'

WHEN 3 THEN 'Three' -- если @myIntVar = 3 то выводим 'Three'

ELSE 'More than three' -- если не сработало ни одно из условий то выводим 'More than three'

END --конец оператора CASE

GO

----------------------------------------------------------------------------

-- Поисковый оператор CASE

DECLARE @MyIntVar int;

SET @MyIntVar = 0;

PRINT CASE -- отсутствует входное выражение

WHEN @MyIntVar IS NULL THEN 'переменная @MyIntVar пустая' -- выражение в конструкции WHEN должно принимать булево значение

WHEN @MyIntVar < 0 THEN 'переменная @MyIntVar меньше нуля'

WHEN @MyIntVar > 0 THEN 'переменная @MyIntVar больше нуля'

WHEN @MyIntVar > 3 THEN 'переменная @MyIntVar больше трех' -- данная строка никогда не будет выполнятся, так как на предыдущей строке была проверка на условиее больше нуля, а любое значение больше трех тоже больше нуля

ELSE 'Непредвиденная ситуация'

END

GO

----------------------------------------------------------------------------

-- Оператор WHILE. Организация циклов

DECLARE @myVar int;

SET @myVar = 0;

WHILE (@myVar < 21) -- условие выполнения цикла, пока условие истинно выполняется цикл.

BEGIN

PRINT 'Текущее Значение ' + CAST (@myVar as varchar);

SET @myVar = @myVar + 1;

END

GO

----------------------------------------------------------------------------

DECLARE @myVar int;

SET @myVar =0;

WHILE @myVar < 21

BEGIN

PRINT 'Текущее Значение ' + CAST (@myVar as varchar);

IF @myVar =5

BEGIN

SET @myVar = @myVar + 2;

CONTINUE; -- Прерывает дальнейшее выполнение текущей итерации и возвращается в начало цикла WHILE

END;

SET @myVar = @myVar + 1;

END

GO

----------------------------------------------------------------------------

DECLARE @myVar int;

SET @myVar = 0;

WHILE @myVar < 21

BEGIN

PRINT 'Текущее Значение ' + CAST (@myVar as varchar);

IF @myVar = 7

BEGIN

PRINT '@myVar = 7! Прерывание цикла!'

BREAK; -- Оператор прерывания цикла (не рекомендуется использовать)

END

SET @myVar = @myVar + 1;

END

GO

----------------------------------------------------------------------------

--------------------------- оператор WAITFOR -------------------------------

WAITFOR DELAY '00:00:10'; -- Задержка на промежуток времени. Возможные значения: в часах:минутах:секундах

PRINT 'Прошло 10 секунд';

WAITFOR TIME '15:20:20'; -- Задержка до указанного значения времени. Возможные значения: в часах:минутах:секундах

PRINT ' пора продолжать';

----------------------------------------------------------------------------

--------------------------- Блоки TRY и CATCH ------------------------------

-- предназначены для обработки ошибок

BEGIN TRY -- Попытка выполнить код, если возникает ошибка то бросаем ее в блок CATCH, иначе продолжаем работу без блока CATCH.

CREATE TABLE TestTable

(

col1 int,

col2 varchar(10)

);

END TRY

BEGIN CATCH -- блок CATCH - блок-обработчик ошибок

DECLARE @ErrorNo int,

@Message nvarchar(4000);

SELECT

@ErrorNo = ERROR\_NUMBER(), --системная функция, возвращает код текущей ошибки

@Message = ERROR\_MESSAGE(); --системная функция, возвращает сообщение текущей ошибки

IF @ErrorNo = 2714

PRINT 'Данная таблица уже существует!'

ELSE

PRINT CAST(@ErrorNo as varchar)+' '+@Message;

END CATCH

GO

----------------------------------------------------------------------------

BEGIN TRY -- пытаемся выполнить код, если возникает ошибка то бросаем ее в блок CATCH, иначе продолжаем работу без блока CATCH.

DROP TABLE TestTable;

END TRY

BEGIN CATCH -- блок CATCH - блок-обработчик ошибок

DECLARE @ErrorNo int,

@Message nvarchar(4000);

SELECT

@ErrorNo = ERROR\_NUMBER(), --системная функция, возвращает код текущей ошибки

@Message = ERROR\_MESSAGE(); --системная функция, возвращает сообщение текущей ошибки

IF @ErrorNo = 3701

PRINT 'Данная таблица уже удалена!'

ELSE

PRINT CAST(@ErrorNo as varchar)+' '+@Message;

END CATCH

GO

----------------------------------------------------------------------------

-- Активация сообщения об ошибке вручную

IF NOT EXISTS (

SELECT s.name,t.name

FROM sys.schemas as s

JOIN sys.tables t

ON s.schema\_id = t.schema\_id

WHERE s.name = 'dbo'

AND t.name = 'TestTable'

)

BEGIN -- начало блока

PRINT 'Таблица TestTable не найдена';

PRINT 'Создаю таблицу TestTable';

CREATE TABLE TestTable --создать таблицу если она не существует

(

Col1 int,

Col2 varchar(10)

)

END -- конец блока

ELSE

BEGIN

RAISERROR('Данная таблица существует', 11, 238) -- Ошибка вызваная пользователем в ручную

-- RAISERROR('Данная таблица существует', 11, 238)-- Снять комментарий

-- (второй аргумент RAISERROR - степень серезности ошибки и обозначение состояния)

-- RAISERROR (строка сообщения, степень серезности ошибки, обозначение состояния)

/\*степень серезности ошибки - определяет какие действия следует принимать при возникновении ошибки

обозначение состояния - произвольное значение, применяется в роли маркера места возникшей ошибки,

если у нас в нескольких местах происходит ошибка, то видя значение

обозревателя состояния можно определить где возникла в коде ошибка\*/

END;

GO

**Хранимые процедуры**

USE AdventureWorks2012;

GO

-------------------------------------------------------------------

CREATE PROC spEmployee -- Создание хранимой процедуры.

AS

SELECT \* FROM HumanResources.Employee;

GO

EXEC spEmployee; --Вызов хранимой процедуры.

GO

--------------------------------------------------------------------

ALTER PROC spEmployee -- Изменение хранимой процедуры

AS

SELECT he.BirthDate,he.BusinessEntityID FROM HumanResources.Employee he;

GO

EXEC spEmployee; -- Вызов измененной хранимой процедуры.

DROP PROC spEmployee; -- Удаление хранимой процедуры.

EXEC spEmployee; -- ОШИБКА, так как данная процедура удалена.

GO

----------------------------------------------------------------------

/\*Хранимая процедура с параметрами\*/

DROP PROC spEmployeeByName

CREATE PROC spEmployeeByName

@LastName nvarchar(25) -- При инициализации параметра значение по умолчанию не задано.

AS

SELECT pc.BusinessEntityID, pc.FirstName, pc.LastName, pc.ModifiedDate

FROM Person.Person pc

WHERE pc.LastName = @LastName;

GO

EXEC spEmployeeByName 'Abel' -- передаем процедуре обязательное значение

EXEC spEmployeeByName -- ОШИБКА если не передать обязательное значение по умолчанию

------------------------------------------------------------------------

DROP PROC spEmployeeByName;

GO

CREATE PROC spEmployeeByName

@LastName nvarchar(25) = NULL -- чтобы указать что параметр является не обязательным, при инициализации необходимо ввести для него значение по умолчанию.

AS

IF @LastName IS NOT NULL -- условная конструкция IF где @LastName IS NOT NULL является проверочным выражением возвращающим TRUE или FALSE

SELECT pc.BusinessEntityID, pc.FirstName, pc.LastName, pc.ModifiedDate

FROM Person.Person pc

WHERE pc.LastName LIKE @LastName + '%'

ELSE

SELECT pc.BusinessEntityID, pc.FirstName, pc.LastName, pc.ModifiedDate

FROM Person.Person pc;

GO

EXEC spEmployeeByName -- вызов без параметра.

EXEC spEmployeeByName 'Ca' --вызов с параметром

EXEC spEmployeeByName 'Cao'

GO

------------------------------------------------------------------------------

/\*Выходные параметры в хранимых процедурах\*/

DROP PROC spEmployeeCount;

GO

CREATE PROC spEmployeeCount

@Info int = null OUTPUT -- Для обьявления выходного парметра используется ключевое слово OUTPUT

AS

BEGIN

SET @Info =(SELECT Count(\*) From Person.Person);

END

GO

DECLARE @MyInfo int;

EXEC spEmployeeCount @MyInfo OUTPUT; -- при вызове хранимой прроцедуры ключевое слово OUTPUT должно использоваться так же как и при обьявлении, обратите внимание как присваивается значение внешней переменной

PRINT CAST (@MyInfo as varchar);

GO

---------------------------------------------

--- Оператор возврата значения RETURN ---

DROP PROC TestProc

CREATE PROC TestProc

AS

BEGIN

DECLARE @MyVar int = 10;

RETURN @MyVar; -- оператор RETURN в процедурах возвращает ТОЛЬКО целочисленное значение!

END;

GO

DECLARE @MyRTN int;

EXEC @MyRTN = TestProc;

PRINT CAST(@MyRTN as varchar);

GO

-------------------------------------------

DROP PROC TestProc;

GO

CREATE PROC TestProc

AS

BEGIN

DECLARE @MyVar int = 10;

RETURN 'Done' -- оператор RETURN в процедурах возвращает ТОЛЬКО ЦЕЛОЧИСЛЕННОЕ ЗНАЧЕНИЕ!!!

END;

GO

DECLARE @MyRTN2 varchar(5);

EXEC @MyRTN2 = TestProc;

PRINT @MyRTN2;

GO

-------------------------------------------

DROP PROC spTestProc

GO

CREATE PROC spTestProc

AS

BEGIN

PRINT 'Сейчас выполнится первый RETURN';

RETURN; -- по умлочанию оператор RETURN возвращает 0

PRINT 'Сейчас не выполнится второй RETURN';

RETURN 5; -- после выполнения первого оператора RETURN, процедура прекращает свое выполнение

END;

GO

DECLARE @MyVar3 int;

EXEC @MyVar3 = spTestProc;

PRINT @MyVar3;

GO

------------------------------------------

--- Рекурсия ----

-- МАКСИМАЛЬНАЯ ГЛУБИНА РЕКУРСИИ 32 УРОВНЕЙ

DROP PROC spFactorial

CREATE PROC spFactorial

@ValueIn int,

@ValueOut int OUTPUT

AS

BEGIN

DECLARE @InWorking int;

DECLARE @OutWorking int;

IF (@ValueIn != 1)

BEGIN

SET @InWorking = @ValueIn -1;

EXEC spFactorial @InWorking, @OutWorking OUTPUT; -- вызов процедуры из самой себя (рекурсия)

SET @ValueOut = @ValueIn \* @OutWorking;

END

ELSE

SET @ValueOut = 1;

END;

GO

-------------------------------------------

DECLARE @MyVarOut int,

@MyVARIn int;

SET @MyVARIn = 7;

EXEC spFactorial @MyVarIn, @MyVarOut OUTPUT; -- 7!= 1\*2\*3\*4\*5\*6\*7

PRINT CAST(@MyVARIn as varchar) + ' факториал: ' + CAST(@MyVarOut as varchar);

GO

----------------------

-- Ошибка. Не хватает диапазона значений типа использованного для переменных!

DECLARE @MyVarOut int,

@MyVARIn int;

SET @MyVARIn = 13;

EXEC spFactorial @MyVarIn, @MyVarOut OUTPUT; -- 13!= 1\*2\*3\*4\*5\*6\*7\*8\*9\*10\*11\*12\*13

PRINT CAST(@MyVARIn as varchar) + ' факториал: ' + CAST(@MyVarOut as varchar);

----Процедура регистрации ошибок в таблице -----

DROP PROC uspLogError;

GO

DROP TABLE ErrorLog2

CREATE TABLE ErrorLog2

(

ErrorId int IDENTITY,

ErrorLine int,

ErrorMessage varchar(200)

)

GO

-- Создаем процедуру регистрации ошибок

CREATE PROC uspLogError

@ErrorLogId int = 0 OUTPUT

AS

BEGIN

INSERT dbo.ErrorLog2 -- Запись данных об ошибке.

(

ErrorLine,

ErrorMessage

)

VALUES

(

ERROR\_LINE(),

ERROR\_MESSAGE()

)

SET @ErrorLogId = @@IDENTITY; -- @@IDENTITY возвращает номер последнего идентификатора записанного в таблицу.

END;

------------------------------------------------

BEGIN TRY

CREATE TABLE OurTest

(

col int

)

END TRY

BEGIN CATCH

DECLARE @OutIdError int;

IF ERROR\_NUMBER() = 2714

BEGIN

PRINT 'Попытка создать существующую таблицу';

EXEC uspLogError @OutIdError OUTPUT;

PRINT 'Ошибка записана в журнал Номер записи: ' + CAST(@OutIdError as varchar);

END

ELSE

PRINT 'Не известная ошибка ';

END CATCH

SELECT \* FROM ErrorLog2

GO

**Функции**

/\*Пользовательские функции возвращаемые скалярное значение (не табличное)\*/

USE ShopDB

GO

CREATE FUNCTION Hello() -- создать функцию

RETURNS varchar(30) -- обьявляем тип возвращаемого значения

AS

BEGIN --начало тела ф-ции

DECLARE @MyVar varchar(20) ='Hello World!';

RETURN @MyVar; --возвращаемое значение ф-ции

END; -- конец тела ф-ции

GO

PRINT dbo.Hello();

DROP TABLE TestTable;

GO

CREATE TABLE TestTable

(

id int identity not null,

name varchar(25) not null,

CDate smalldatetime not null

)

GO

-------------------------------------------------------

DECLARE @MyVar int =1;

DECLARE @MyVcVar varchar(10);

WHILE @MyVar < 20

BEGIN

SET @MyVcVar = 'Test ' + CAST(@MyVar as varchar);

-- заносим данные в таблицу

INSERT TestTable

( name, CDate )

VALUES (@MyVcVar, DATEADD(MI, @MyVar, GETDATE()));

SET @MyVar = @MyVar + 1;

END

GO

/\* данная выборка будет пустая, так как функция GETDATE()

возвращает значение типа datetime (возвращает не только

дату но и текущее время до 3-х сотых секунды)\*/

SELECT \* FROM TestTable

WHERE CDate = GETDATE();

GO

---------------------------------------------------------

CREATE FUNCTION DateOnly (@Date datetime) -- создаем функцию вводя ее имя и аргумент

RETURNS date --обявляем возвращаемый тип

AS

BEGIN --тело функции

RETURN CAST(@Date as date); -- приводим к типу date возвращаемое значение

END

GO

SELECT \* FROM TestTable -- данная выборка вернет 19 записей(все за сегодняшний день

WHERE dbo.DateOnly(CDate)= dbo.DateOnly(GETDATE()); --нужно явно указывать схему при вызове пользовательской функции

----------------------------------------

USE AdventureWorks2012

GO

/\* Делаем выбоку для просмотра цены каждого горного велосипеда,

в данной выборке используются запросы внутри запросов их назы-

вают вложеными запросами\*/

SELECT Name,

ListPrice,

(SELECT AVG(ListPrice) FROM Production.Product WHERE ProductSubcategoryID = 1) as Average, -- используем вложеный запрос, мы помним что функция AVG - возвращаемая среднее значение

ListPrice - (SELECT AVG(ListPrice) FROM Production.Product WHERE ProductSubcategoryID = 1) as DifferencePrice

FROM Production.Product

WHERE ProductSubcategoryID = 1; -- делаем выборку для Mountain Bikers

GO

DROP FUNCTION AvgPrice

DROP FUNCTION DfrncPrice

GO

-- вложенный запрос (SELECT AVG(ListPrice)FROM Production.Product) можно оформить в виде функции,

-- так как он возвращает выборку из одного значения

CREATE FUNCTION AvgPrice()

RETURNS money

WITH SCHEMABINDING -- Запрещает удаление таблицы.

AS

BEGIN

RETURN (SELECT AVG(ListPrice)FROM Production.Product WHERE ProductSubcategoryID = 1);

END

GO

--- использование вложенных функций

CREATE FUNCTION DfrncPrice(@Price money)

RETURNS money

AS

BEGIN

RETURN @Price - dbo.AvgPrice(); -- возможно создавать вложенные функции

END;

GO

-- выполним тот же запрос для просмотра цены каждого горного велосипеда

SELECT Name,

ListPrice,

dbo.AvgPrice() as AvgPrice,

dbo.DfrncPrice(ListPrice) as DifferencePrice

FROM Production.Product

WHERE ProductSubcategoryID = 1;

GO

/\*Пользвательские функции возвращемые таблицу\*/

USE AdventureWorks2012;

GO

DROP FUNCTION fnContactList

CREATE FUNCTION fnContactList() -- создаем функцию

RETURNS TABLE -- возвращаемый тип TABLE указывает на то что возвращаться будет таблица

AS

RETURN (SELECT LastName, FirstName, ModifiedDate

FROM Person.Person);

GO

SELECT \* FROM dbo.fnContactList(); -- выводим всю информацию из таблицы сформированой при помощи функции fnContactList();

GO

--ЗАДАНИЕ: повторите предыдущий пример с помощью представления

DROP FUNCTION fnContactSearch

CREATE FUNCTION fnContactSearch(@LastName varchar(30)) --передаем один аргумент типа varchar (Фамилия или часть фамилии)

RETURNS TABLE

AS

RETURN (SELECT FirstName , LastName, ModifiedDate

FROM Person.Person

WHERE LastName LIKE @LastName + '%'); -- выборка по фамилии

GO

SELECT \* FROM dbo.fnContactSearch('Berry'); -- выводим всю информацию о сторудниках часть фамилии или фамилия которых 'Berry' из таблицы сформированой при помощи fnContactSearch();

SELECT \* FROM dbo.fnContactSearch('Ber'); -- выводим всю информацию о сторудниках часть фамилии или фамилия которых 'Ber' из таблицы сформированой при помощи fnContactSearch();

**Функции поиска**

USE ShopDB

GO

---------- Создаем таблицу сотрудников. ----------

CREATE TABLE MyEmployee

(

EmployeeID int NOT NULL,

ManagerID int NULL REFERENCES MyEmployee(EmployeeID), -- Показвает кому подченяется данный сотрудник.

JobTitle nvarchar(50) NOT NULL,

LastName nvarchar(50) NOT NULL,

FirstName nvarchar(50) NOT NULL,

CONSTRAINT PK\_Employee2\_EmployeeID PRIMARY KEY CLUSTERED (EmployeeID ASC)

);

GO

-- Иерархические данные (так как таблица содержит информацию о сторудниках подчиняющихся сотрудникам информация о которых так же хранится в этой таблице)

INSERT INTO MyEmployee(EmployeeID,ManagerID,JobTitle,LastName,FirstName)

VALUES

(1, NULL, 'Генеральный директор','Сидоров', 'Виктор'),

(2, 1, 'Финансовый директор', 'Левченко', 'Людмила'),

(3, 1, 'Кадровая поддержка','Стипанов','Григорий'),

(4, 1, 'Исполнительный директор', 'Самойленко', 'Виктор'),

(5, 4, 'Инженер', 'Тимченко', 'Виталий'),

(8, 5, 'Инженер', 'Хабиб', 'Эльдар'),

(9, 5, 'Программист', 'Дулев', 'Павел'),

(10 ,5, 'Data Architect', 'Черчев', 'Робнрт'),

(11 ,5, 'Програмист', 'Залозный', 'Михаил'),

(6, 4, 'Персональный секретарь', 'Радченко', 'Вика'),

(7, 4, 'Начальник охраны','Стельмах','Игорь');

SELECT \* FROM MyEmployee;

-- Узнаем кто подчиняется исполнительному директору.

SELECT sub.EmployeeID,

sub.FirstName,

sub.LastName

FROM

MyEmployee as boss

JOIN

MyEmployee as sub

ON boss.EmployeeID = sub.ManagerID

WHERE boss.JobTitle LIKE 'Исполнительный директор';

GO

-- Рекурсивная функция вывода всех подчиненных.

CREATE FUNCTION fnGetSub (@EmployeeId int) -- Создаем функцию с одним аргументом.

RETURNS @Sub TABLE -- Обявляем возвращаемую таблицу.

(

EmployeeId int,

SubId int,

Name varchar (90)

)

AS

BEGIN

DECLARE @EmpId int;

INSERT @Sub -- Добаляем запись о руководителе в выходную таблицу.

SELECT EmployeeID, ManagerID ,FirstName+' '+LastName

FROM MyEmployee

WHERE EmployeeID = @EmployeeId;

SET @EmpId = (SELECT MIN(EmployeeID) -- Определяем первого подчеенного.

FROM MyEmployee

WHERE ManagerID = @EmployeeId);

-- Если такой есть то заходим в циклическую конструкцию WHILE

WHILE @EmpId IS NOT NULL -- Пока есть подчиненные выполняем цикл.

BEGIN

INSERT @Sub -- Добавляем запись о подчененных относительно выше выбраного руководителя.

SELECT \* FROM dbo.fnGetSub(@EmpId) -- Рекурсия.

SET @EmpId =(SELECT MIN(EmployeeID) -- Определяем следующего подчиненного.

FROM MyEmployee

WHERE EmployeeID > @EmpId

AND

ManagerID = @EmployeeId);

END;

RETURN; -- Если мы обявили возвращаемую таблицу то в RETURN не подставляем никакого значения, иначе возникнет ошибка.

END;

GO

SELECT \* FROM dbo.fnGetSub(4); -- Воспользуемся для поиска функцией - fnGetSub.

SELECT \* FROM MyEmployee;

drop function fnGetSub

**Транзакции**

Транзакциейназываетсявыполнениепоследовательностикоманд (SQL-конструкций)вбазеданных,котораялибофиксируетсяпри успешнойреализациикаждойкоманды,либоотменяетсяпри неудачномвыполнениихотябыоднойкоманды.

Откаттранзакции—этодействие,обеспечивающее аннулированиевсехизмененийданных,которыебылисделаныв телетекущейнезавершеннойтранзакции.

Привыполнениитранзакцийнесколькимипользователямиодной базыданныхмогутвозникатьследующиепроблемы:

* •**Dirtyreads**
* •**Non-repeatablereads**
* •**Phantomreads**

**Dirtyreads**–«грязное» чтение. Первый пользователь начинает транзакцию, изменяющую данные. В это время другой пользователь (или создаваемая им транзакция) извлекает частично измененные данные, которые не являются корректными.

**Non-repeatablereads**–неповторяемое чтение. Первый пользователь начинает транзакцию, изменяющую данные. В это время другой пользователь начинает и завершает другую транзакцию. Первый пользователь при повторном чтении данных (например, если в его транзакцию входит несколько инструкций SELECT) получает другой набор записей.

**Phantomreads**–чтение фантомов. Первый пользователь начинает транзакцию, выбирающую данные из таблицы. В это время другой пользователь начинает и завершает транзакцию, вставляющую или удаляющую записи. Первый пользователь получит другой набор данных, содержащий фантомы –удаленные или измененные строки.

**Readuncommitted**.Транзакцияможетсчитыватьданные,скоторыми работаютдругиетранзакции.Применениеэтогоуровняизоляцииможет привестиковсемперечисленнымпроблемам.

**Readcommitted**.Транзакциянеможетсчитыватьданные,скоторыми работаютдругиетранзакции.Применениеэтогоуровняизоляции исключаетпроблему«грязного»чтения.

**Repeatableread**.Транзакциянеможетсчитыватьданные,скоторыми работаютдругиетранзакции.Другиетранзакциитакженемогутсчитывать данные,скоторымиработаетэтатранзакция.Применениеэтогоуровня изоляцииисключаетвсепроблемы,кромечтенияфантомов.

**Serializable**.Транзакцияполностьюизолированаотдругихтранзакций. Применениеэтогоуровняизоляцииполностьюисключаетвсепроблемы.

USE ShopDB;

GO

--IF OBJECT\_ID('MyUserName') IS NOT NULL

-- DROP TABLE MyUserName;

--GO

-- Удаляем, если существуют такие таблицы...

DROP TABLE MyUserName;

GO

DROP TABLE MyUserTell;

GO

DROP TABLE MyUserInfo;

GO

-- ... ошибка, какая и как исправить?

--- создать таблицу ---

CREATE TABLE MyUserName

(

IdTest int IDENTITY PRIMARY KEY,

FName varchar(25),

LName varchar(25)

)

GO

CREATE TABLE MyUserTell

(

IdTest int IDENTITY PRIMARY KEY,

IdUser int FOREIGN KEY REFERENCES MyUserName(IdTest) UNIQUE,

TellN char(12)

CHECK (TellN LIKE ('([0-9][0-9][0-9])[0-9][0-9][0-9][0-9][0-9][0-9][0-9]')) -- Что это значит?

)

GO

CREATE TABLE MyUserInfo

(

IdTest int IDENTITY PRIMARY KEY,

IdUser int FOREIGN KEY REFERENCES MyUserName(IdTest),

Bdate date

)

GO

--------------------------------------------------------------------

-------------------------- Транзакции ------------------------------

--------------------------------------------------------------------

---------------------------------------------------------

-- Мы уже использовали транзакции (неявно)

BEGIN TRANSACTION; -- начало транзакции

INSERT MyUserName

VALUES('TestName0','TestLName0')

COMMIT TRANSACTION; -- успешное завершение транзакции

SELECT \* FROM MyUserName;

---------------------------------------------------------

-- откат транзакции

BEGIN TRANSACTION -- начало транзакции

INSERT MyUserName

VALUES

('RollTest1','RollTestL1')

ROLLBACK TRANSACTION; -- происходит откат транзакции, наша вставка не выполнится, тоесть:

-- INSERT MyUserName VALUES ('TestName1','RollTestL1') ,- не произойдет

SELECT \* FROM MyUserName;

---------------------------------------------------------

-- в одной транзакции можно выполнять несколько действий

BEGIN TRANSACTION; -- начало транзакции

DECLARE @Id int;

INSERT MyUserName

VALUES

('TestName0','TestLName0')

SET @Id = @@IDENTITY;

INSERT MyUserTell

VALUES

(@Id,'(097)2224455');

INSERT MyUserInfo

VALUES (@Id,'01/02/1990');

COMMIT TRANSACTION; -- успешное завершение транзакции

SELECT \* FROM MyUserName;

SELECT \* FROM MyUserInfo;

SELECT \* FROM MyUserTell;

GO

---------------------------------------------------------

-- В одной транзакции можно "откатить" несколько действий.

BEGIN TRANSACTION; -- Начало транзакции.

DECLARE @Id int;

INSERT MyUserName

VALUES

('RollBackTest','TestLName0')

SET @Id = @@IDENTITY;

INSERT MyUserTell

VALUES

(@Id,'(097)2224455');

INSERT MyUserInfo

VALUES (@Id,'01/02/1990');

ROLLBACK TRANSACTION; -- Откат последовательности дейсвий.

SELECT \* FROM MyUserName;

SELECT \* FROM MyUserInfo;

SELECT \* FROM MyUserTell;

---------------------------------------------------------

-- Точка сохранения транзакции (промежуточное сохранение транзакции)

BEGIN TRAN

INSERT MyUserName

VALUES

('SavePoinTest1','TestLName0')

SAVE TRAN SavePointTran; -- Точка сохранения транзакции (промежуточное сохранение транзакции)

PRINT 'SAVE POINT';

INSERT MyUserName

VALUES

('SavePointTest2','TestLName0')

ROLLBACK TRAN SavePointTran; -- Откат к точке востановления.

COMMIT TRAN -- Снять комментарий после выполнения транзакции ниже по коду.

SELECT \* FROM MyUserName;

GO

---------------------------------------------------------

-- Точка сохранения транзакции (промежуточное сохранение транзакции)

BEGIN TRAN

INSERT MyUserName

VALUES

('SavePoinTest3','TestLName0')

SAVE TRAN SavePointTran; -- Точка сохранения транзакции (промежуточное сохранение транзакции)

PRINT 'SAVE POINT';

INSERT MyUserName

VALUES

('SavePointTest4','TestLName0')

ROLLBACK TRAN -- если не указана точка сохранения к которой нужно произвести откат,

-- то откат происходит к началу транзакции

SELECT \* FROM MyUserName;

GO

---------------------------------------------------------

DROP PROC MyTransactProc;

GO

-- Процедура записи данных о пользователе (имя, фамилия, номер телефона, дата рождения) , есть мальнькая ошибка, исправте пожалуйста...

CREATE PROC MyTransactProc

@FName varchar(25),

@LName varchar(25),

@TellN char(12),

@BDate date

AS

BEGIN

DECLARE @Id int;

BEGIN TRAN -- Начало транзакции.

INSERT MyUserName

VALUES (@FName,@LName);

SET @Id = @@IDENTITY;

INSERT MyUserTell

VALUES (@Id,@TellN);

INSERT MyUserInfo

VALUES (@Id,@BDate);

IF EXISTS (SELECT \* FROM MyUserName WHERE @FName = FName AND @LName = LName AND IdTest != @Id) -- Проверка на наличие данных об этом человеке.

BEGIN

ROLLBACK TRAN; -- Откат транзакции если человек существует.

RETURN 1;

END;

COMMIT TRAN ; -- Успешное завершение транзкции если данный пользователь еще не записан в БД.

END;

-- Проверка результатов работы процедуры. --

EXEC MyTransactProc -- Запись данных.

@Fname = 'Name1',

@LName = 'LName1',

@TellN = '(097)7775566',

@BDate = '01/02/2010';

GO

EXEC MyTransactProc -- Повторная попытка ввести эти же данные.

@Fname = 'Name1',

@LName = 'LName1',

@TellN = '(097)7775566' ,

@BDate = '01/02/2010';

GO

EXEC MyTransactProc

@Fname = 'TestName2',

@LName = 'TestLName2',

@TellN = '(097)7775566' ,

@BDate = '01/02/2010';

GO

SELECT \* FROM MyUserName;

SELECT \* FROM MyUserInfo;

SELECT \* FROM MyUserTell;

**Триггеры**

USE AdventureWorks2012;

GO

-- Блок настроек для успешной работы триггеров.

SET ANSI\_NULLS ON -- Если SET ANSI\_NULLS установлено в ON, любая инструкция SELECT, использующая предложение WHERE column\_name = NULL, не вернет ни одной записи, даже если в столбце column\_name есть значения NULL.

GO

-- Если параметры SET ARITHABORT и SET ANSI WARNINGS установлены в значение ON,

-- эти условия ошибок приведут к завершению запроса. Если параметр SET ARITHABORT установлен в значение ON,

-- а параметр SET ANSI WARNINGS установлен в значение OFF, то эти условия ошибок приведут к завершению пакета.

SET ARITHABORT ON -- Если при исполнении транзакции произошли ошибки, то для этой транзакции выполняется откат.

GO

SET CONCAT\_NULL\_YIELDS\_NULL ON -- Когда параметр SET CONCAT\_NULL\_YIELDS\_NULL установлен в ON, объединение значения NULL со строкой дает в результате NULL.

GO

SET QUOTED\_IDENTIFIER ON -- Если параметру SET QUOTED\_IDENTIFIER присвоено значение ON, то идентификаторы можно заключать в двойные кавычки и литералы должны быть разделены одинарными кавычками.

GO

SET ANSI\_PADDING ON -- Столбцы, определенные типами данных char, varchar, binary и varbinary, имеют определенный размер.

GO

SET ANSI\_WARNINGS ON -- Если значения NULL появляются в агрегатных функциях, таких как SUM, AVG, MAX, MIN, STDEV, STDEVP, VAR, VARP или COUNT, то при установке значения ON формируется предупреждающее сообщение.

GO

-- Если параметр SET NUMERIC\_ROUNDABORT имеет значение ON, после потери точности в выражении формируется ошибка.

-- Если задано значение OFF, потери точности не приводят к формированию сообщений об ошибках, а результат округляется с точностью столбца или переменной, в которых будет сохранен.

SET NUMERIC\_ROUNDABORT OFF

GO

-- http://msdn.microsoft.com/ru-ru/library/ms188048.aspx

--------------------------------------------------------------------------

DROP TRIGGER Sales.SaleaOrderDetailNotDiscontinued; -- Удаление тригеров.

GO

--------------------------------------------------------------------------

-- Триггер - это обработчик который можно выполнить во время выполнения операций - INSERT, UPDATE, DELETE.

-- Вместе с созданием триггера постоянно создаются две служебные таблицы: Inserted и deleted

CREATE TRIGGER Sales.SaleaOrderDetailNotDiscontinued

ON Sales.SalesOrderDetail

FOR INSERT, UPDATE

AS

IF EXISTS

(

SELECT \*

FROM Inserted as i -- Данные вставляемые пользователем в таблицу Sales.SalesOrderDetail(inserted - внутренняя таблица тригера где хранятся строки которые буду проверяться и при хорошем результате проверки потом вставляться в таблицу Sales.SalesOrderDetail)

JOIN Production.Product p

ON i.ProductID = p.ProductID

WHERE p.DiscontinuedDate IS NOT NULL -- проверяем есть ли дата снятия с продажи

)

BEGIN

RAISERROR('Товар отсутсвует на складе!', 1, 16); -- выводим ошибку если товар уже снят с продажи

ROLLBACK TRAN

END

GO

SELECT \* FROM Production.Product WHERE ProductID = 706;

UPDATE Production.Product -- создаем проверочную строку где указана дата снятия с продажи так как в таблице Production.Product таковых данных нету

SET DiscontinuedDate = '01-03-2011'

WHERE ProductID = 940;

SELECT \* FROM Sales.SalesOrderDetail;

INSERT INTO Sales.SalesOrderDetail -- попытаемся добавить этот заказ c товаром ID которого 940

VALUES

(43659,'4911-403C-98',1,940,1,6.45,0.00,NEWID(),GETDATE());

INSERT INTO Sales.SalesOrderDetail -- попытаемся добавить этот заказ c товаром ID которого 706

VALUES

(43659,'4911-403C-98',1,706,1,6.45,0.00,NEWID(),GETDATE());

------------------------------------------------------------------------------------------------------

--- Приминение тригеров для проверки дельты (изменения) обновления ---

--Дельта обновления - велечина изменения обновленного значения относительно предыдущего.

SELECT \* FROM Production.ProductInventory

GO

DROP TRIGGER Production.ProductIsRationed

GO

CREATE TRIGGER Production.ProductIsRationed

ON Production.ProductInventory

FOR UPDATE

AS

IF EXISTS

(

SELECT \*

FROM inserted i

JOIN deleted d

ON d.ProductID = i.ProductID

AND d.LocationID=i.LocationID

WHERE (d.Quantity - i.Quantity) > d.Quantity/2 --вычисляем дельта обновление (разница между текущим количеством товара и сколько останется после продажи)

AND d.Quantity - i.Quantity > 0

)

BEGIN

RAISERROR('Не возможно покупать больше 50 процентов от оставшегося количества товаров ',1,2)

ROLLBACK TRAN

END

GO

SELECT \* FROM Production.ProductInventory WHERE ProductID = 4 AND LocationID = 6

UPDATE Production.ProductInventory

SET Quantity = 200 -- Пытаемся продать 422-200 = 222-больше 50% процентов

WHERE ProductID = 4

AND LocationID = 6

UPDATE Production.ProductInventory

SET Quantity = 222 -- Пытаемся продать 422-222 = 200- меньше 50% процентов

WHERE ProductID = 4

AND LocationID = 6

UPDATE Production.ProductInventory

SET Quantity = 422

WHERE ProductID = 4

AND LocationID = 6

GO

---------------------------------------------------------------------

---отмена действия тригеров на таблице без его удаления---

ALTER TABLE Production.ProductInventory

DISABLE TRIGGER ALL ; -- отключаем тригеры

GO

UPDATE Production.ProductInventory

SET Quantity = 200 --пытаемся продать 422-200 = 222-больше 50% процентов

WHERE ProductID = 4

AND LocationID = 6

GO

UPDATE Production.ProductInventory

SET Quantity = 422

WHERE ProductID = 4

AND LocationID = 6

GO

ALTER TABLE Production.ProductInventory

ENABLE TRIGGER ALL ; -- включаем тригер

GO

UPDATE Production.ProductInventory

SET Quantity = 200 --пытаемся продать 422-200 = 222-больше 50% процентов

WHERE ProductID = 4

AND LocationID = 6

GO

---Production.ProductIsRationed - реализован правильно но не оптимизирован! --

------------------------------------------------------------------------------

---- Правильно так: -----

ALTER TRIGGER Production.ProductIsRationed -- ALTER TRIGGER позволяет изменить существующий тригер не удаляя его

ON Production.ProductInventory

FOR UPDATE

AS

IF UPDATE(Quantity) -- Функция UPDATE() - возвращает TRUE или FALSE в зависимости от того осуществляется ли обновление конкретного столбца.

BEGIN

IF EXISTS

(

SELECT \*

FROM inserted i

JOIN deleted d

ON d.ProductID = i.ProductID

AND d.LocationID=i.LocationID

WHERE (d.Quantity - i.Quantity) > d.Quantity/2 --вычисляем дельта обновление (разница между текущим количеством товара и сколько останется после продажи)

AND d.Quantity - i.Quantity > 0

)

BEGIN

RAISERROR('Не возможно покупать больше 50 процентов от оставшегося количества товаров ',1,16)

ROLLBACK TRAN

END

END

GO