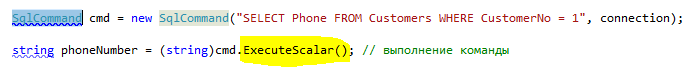
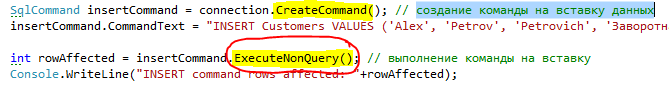
SqlCommand

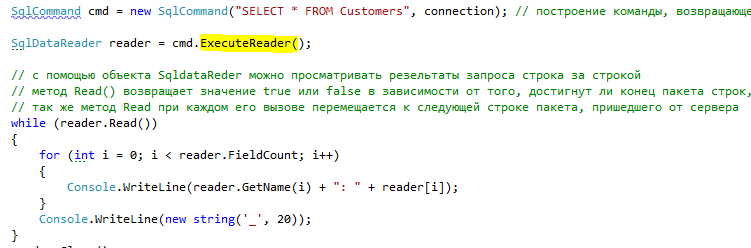
Выполнение команд, возвращающих скалярные значения



создание команды на вставку данных



построение команды, возвращающей данные в табличном представлениии



SqlDataReader

Метод GetFieldValue

reader.GetFieldValue<string>(0) – возвращает 1й столбец приведенный в string

аналогично

reader.GetString(0)

с датой

Console.WriteLine("{0:D}",reader.GetDateTime(8));

Обращение к полю с помощью индексатора (целочисленный или строковой)

public override object this[int i] { get; }

public override object this[string name] { get; }

reader[8]

выполнение пакета операторов SQL с помощъю одного объекта SqlCommand

reader.NextResult(); - переход к следующему запросу

метод IsDbNull позволяет проверить наличие данных

(reader.IsDBNull(5)) - метод IsDbNull позволяет проверить наличие данных в указанном поле источника данных

Или проводить проверку по индексатору

if (reader[5] == DBNull.Value)

async

чтобы не блокировать пользовательский интерфейс

async private void getdataAsyncButtom\_Click(object sender, EventArgs e)

{

using (SqlConnection connection = new SqlConnection(conStr))

{

await connection.OpenAsync(); // асинхронное открытие соединения

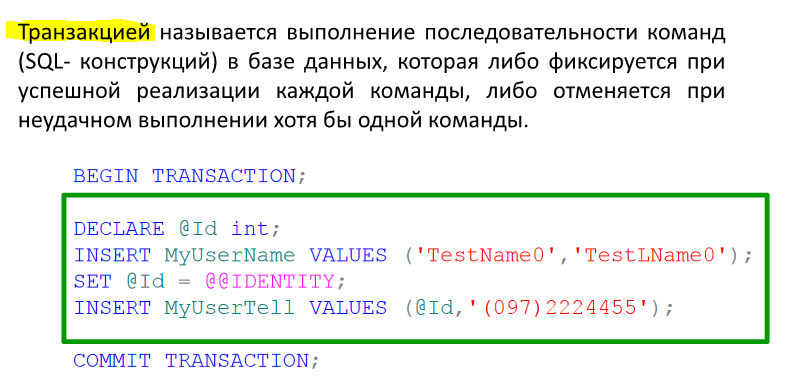
SqlCommand cmd = new SqlCommand("WAITFOR DELAY '00:00:10'", connection); // асинхронное выполнение команды

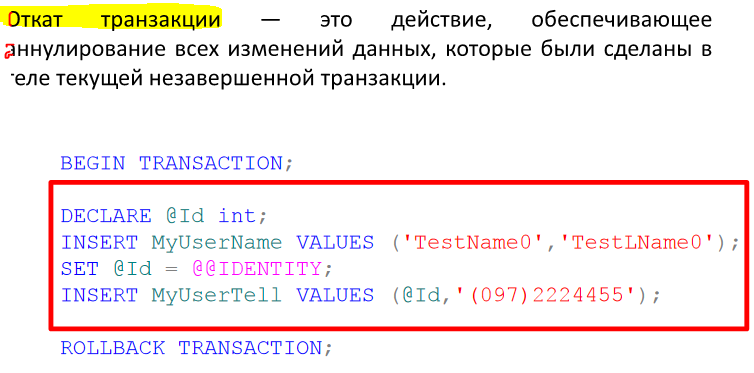
await cmd.ExecuteNonQueryAsync();

MessageBox.Show("Command executed async");

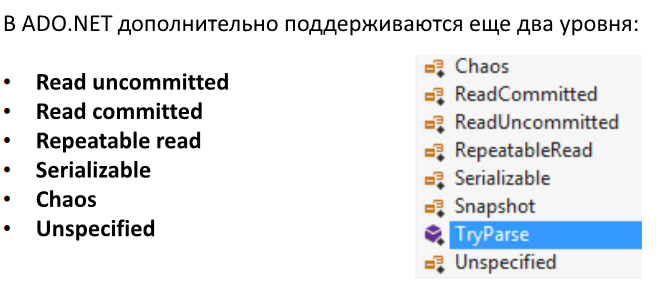
}

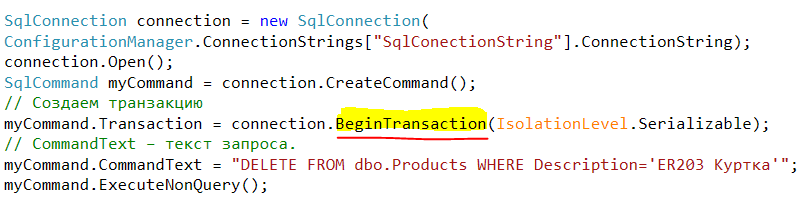
}



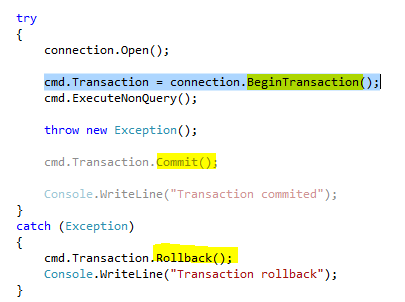


Уровни изолированности транзакции (другой пользователь не сможет выполнить определенную транзакцию если выбранный уровень запрещает делать именно это(может вставка новых данных, или вывод записей на экран)





Получение транзакции



cmd.Transaction = connection.BeginTransaction();

для решения проблем с грязным чтением и другими чтениями нужно при выполнении команды назначить (к примеру IsolationLevel.ReadCommitted) ограничения и если у же кем то запущена транзакция то наша транзакция будет ждать окончания, а после выполниться

cmd.Transaction = connection.BeginTransaction(IsolationLevel.ReadCommitted);

SqlCommand

Parameters.AddWithValue

SqlCommand cmd = new SqlCommand(commandStr, connection); - создание команды

cmd.Parameters.AddWithValue("CustomerNo", customerNo); - добавление параметра в коллекцию параметров команды

не смотря на то что в строке запроса переменная (@CustomerNo;"), в параметрах можно указать без знака @ - CustomerNo

Выходной параметр

SqlCommand cmd = new SqlCommand("SET @Parameter = 2;", connection);

SqlParameter parameter = cmd.Parameters.Add(new SqlParameter("Parameter", System.Data.SqlDbType.Int));

parameter.Direction = System.Data.ParameterDirection.Output; - указание направления параметра

cmd.ExecuteNonQuery();

Console.WriteLine("Parameter value: " + parameter.Value); // вывод на экран значения параметра после выполнения запроса

Хранимая процедура

SqlCommand cmd = new SqlCommand("EXECUTE selectEmp", connection);

Либо так вызывать

SqlCommand cmd = new SqlCommand("selectEmp", connection) { CommandType = System.Data.CommandType.StoredProcedure };

Процедура возвращающая параметры

// код хранимой процедуры CREATE PROCEDURE ProcedureReturnValue

// AS

// BEGIN

// return 1;

// END

SqlCommand cmd = new SqlCommand("ProcedureReturnValue", connection) { CommandType = System.Data.CommandType.StoredProcedure };

SqlParameter parameter = cmd.Parameters.Add(new SqlParameter());

parameter.Direction = System.Data.ParameterDirection.ReturnValue; - после выполнения комманды parameter будет содержать возвращаемое значение хранимой процедуры