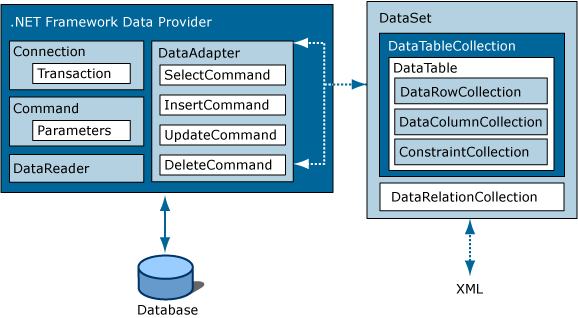
ADO.NET

# Архитектура.

Архитектура ADO.NET предоставляет единую модель для взаимодействия с внешними источниками данных. Это может быть MS SQL серверы, Oracle, соединения посредством OLE или ODBC, а также XML. В основе этой архитектуры лежат 4 основных объекта – Connection, Command, DataReader и DataAdapter. У этих объектов в .NET есть соответствующие базовые классы, но для каждого источника данных должны быть свои реализации. Connection отвечает за соединение с источником данных. Command предлагает единую систему передачи различных команд источнику данных. DataReader считывает из источника данных однопроходный поток данных только для чтения. DataAdapter заполняет DataSet и выполняет обновления в источнике данных.



Кроме этих основных объектов в ADO.NET также представлены следующие объекты:

* Transaction – связывает команды с транзакцией внутри источника данных.
* Parameter – определяет входные, выходные и возвращаемые значения параметров для команд к источникам данных.
* DataSet – единая структура данных, хранящаяся целиком в оперативной памяти.

# Connection.

Базовым классом является DbConnection. Для обеспечения соединения с источником данных используется строка соединения «Connection string». Для каждого конкретного источника данных свой формат «строки соединения».



В данном примере создается объект соединения с базой MS SQL. Используется строка соединения, в которой указан сервер (Serjnew), база данных в этом сервере (serj01), а также указано использовать встроенную аутентификацию Windows.

Поскольку соединение с базой – длительная операция, в ADO.NET используется встроенная система пулов соединений. При первом создании объекта Connection создается пул, связанный с данной строкой соединения. При освобождении соединение возвращается в пул, и если создается новый объект Connection с той же строкой соединения, то само соединение с базой переиспользуется. Если при создании объекта Connection в пуле нет свободных соединений, то создается новое. Размером пула можно управлять через строку соединения, выставляя ей максимальное и минимальное количества соединений в пуле. Также можно принудительно очищать пул соединений.



В данном примере мы задаем, что, если при освобождении соединения, в пуле уже будет одно соединение, то освобождаемое соединение закроется. Если уже создано 2 соединения и создается новый объект соединения, то поскольку достигнут максимум соединений, новое соединение не будет создано, а код, вызвавший создание нового соединения, будет ожидать освобождения используемого соединения.

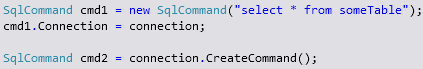


В данном примере из пула убирается указанное соединение, а потом чистятся все пулы.

# Command.

Используется для выполнения команды в источник данных.

# Создание.



В данном примере приведены 2 способа создания команды. В 1-ом способе мы создает объект с текстом команды, потом мы можем указать, через какое именно соединение мы должны выполнить команду.

2-ой способ сразу устанавливает, какому соединению принадлежит команда. Кроме того, таким образом можно работать с абстрактными компонентами, если предполагается использование нескольких баз данных:



Здесь используются переменные абстрактных классов, лишь однажды указывая, что работаем с MS SQL.

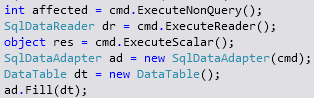
Команды также делятся на типы – Text, Stored Procedure, Table name.



1-ый означает обычный sql-запрос. 2-ой – что используется хранимая процедура, 3-ий – прямое обращение к таблице.

# Выполнение.

Для выполнения команды существует много способов, которые рассчитаны на различное применение команды.



В данном примере используется несколько способов.

В 1-ом случае предполагается, что команда не возвращает никаких данных из базы (или они не нужны), кроме количества строк (Raws), на которые повлияла команда.

Во 2-ом случае в качестве результата получается *DataReader*, т.е. курсор от базы для чтения результатов команды.

В 3-ем случае в качестве результата получается данные из первой строчки, первой колонки результата запроса. Функция возвращает *object*, показывая, что результат может быть любого типа, однако по фактов он является одним из стандартных типов вроде *int, bool, string, DateTime* и тд. Достаточно лишь привести к ожидаемому типу.

В последнем случае для вызова команды используется DataAdapter. В данном случае посредством *SqlDataReader* он помещает данные запроса в *DataTable*.

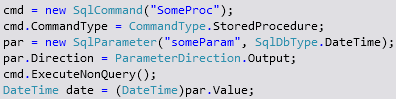
# Параметры.

Команду не всегда можно описать только простым sql-текстом. Например, бинарные данные для blob в текст запроса не поместишь. Более того, вносить меняющиеся части в тексте запроса с использование генерации текста не рекомендуется, поскольку так в текст запроса могут попасть некорректные данные. Во всех таких случаях рекомендуется использовать параметры.



В данном примере создается команда, в которой используется параметр – id, который задается ниже. Здесь «@id» - местозаполнитель для параметра. Как правило в каждой базе данных свой место заполнитель. Например, в оракле используется двоеточие с именем параметра, а в OLE и ODBC просто «?» без указания имени параметра.

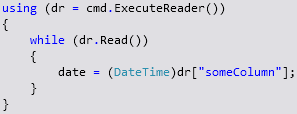
Параметры очень полезны при использовании хранимых процедур, поскольку позволяют получать результаты процедур, которые помещаются в выходные параметры.



Здесь создается параметр, который помечается как выходной, и после выполнения команды мы можем взять значение.

# DataReader.

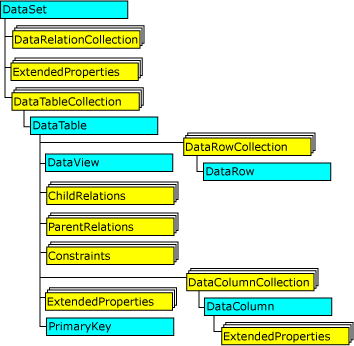
Однопроходный поток данных из базы. Это самый быстрый способ использования результатов запроса. Все остальные используют его, создавая надстройки.



В данном примере показано типичное использование *DataReader*. Объект создается командой. Далее считывается очередная строка результата запроса. И после считывания мы можем обратиться к разным колонкам текущей строки. *DataReader* нужно всегда закрывать после использования, поскольку он блокирует соединение с базой. Более того, пока он открыт использовать соединение нельзя. Нельзя также получить результаты выходных параметров.

# DataAdapter и DataSet.

Для полноценного использования базы данных вполне достаточно 3 объектов – Connection, Command и DataReader. Однако они могут показаться не очень удобными, поэтому поверх данных 3 объектов выстроена система, в которой DataAdapter выступает в качестве управляющего элемента, который строит логику обработки данных с внесением в базу необходимых запросов, а DataSet в качестве хранилища данных.



*DataSet* представляет собой взаимосвязанный набор таблиц, в которых в свою очередь есть первичные ключи, ограничения, и прочее. Таким образом, *DataSet* целиком поддерживает реляционную модель данных, оставляя все данные в памяти.

*DataAdapter* позволяет организовать логику взаимодействия *DataSet* с базой данных, отправляя нужные команды в базу, если что-то изменилось в *DataSet* или наоборот получать данные из базы.