

Projektowanie aplikacji z bazami danych

ADO.NET

Paweł Rajba

Instytut Informatyki
Uniwersytet Wrocławski

Plan wykładu

- Wprowadzenie
- Nawiązywanie połączenia
- Wykonywanie kwerend
- Przeglądanie wyniku zapytania
- Parametryzacja kwerend
- Obiekty DataSet, DataTable, DataRelation, Scalanie danych
- Obiekt DataAdapter
- Transakcje
- Visual Studio i ADO.NET

Wprowadzenie do ADO.NET

- Pozwala aplikacjom na dostęp do baz danych
- Tworzy przestrzeń nazw System.Data
- ADO.NET składa się z dwóch części:
 - dostawców danych i obiektu DataSet
- Dostawcy danych
 - dla SQL Server, przestrzeń nazw System.Data.SqlClient
 - dla OLEDB, przestrzeń nazw System.Data.OleDb
 - dla ODBC, przestrzeń nazw System.Data.Odbc
 - dla Oracle
 - System.Data.OracleClient – sterownik MS
 - Oracle.DataAccess.Client – sterownik Oracle
 - Dostawcy dedykowani (SQL Server i Oracle są wydajniejsi)

Wprowadzenie do ADO.NET

- Kilka słów o sterownikach do Oracle Database
 - Oracle zaleca używania ich sterownika, ponieważ:
 - Sterownik Oracle daje większe możliwości
 - Sterownik MS jest deprecated (wg MS)
<http://www.oracle.com/technology/tech/dotnet/msoc/index.html>
 - Prawdopodobnie MS w ogóle wycofa się z rozwoju sterownika
<http://adtmag.com/articles/2009/06/16/microsoft-kills-oracle-data-provider-for-adonet.aspx>
 - Więcej o sterowniku Oracle'a:
<http://www.oracle.com/technology/tech/windows/odpnet/faq.html>

Wprowadzenie do ADO.NET

- Cztery główne składowe
 - Connection, Command, DataReader, DataAdapter
- Typowy scenariusz użycia
 - połączenia się bazą danych,
 - wykonywania poleceń
 - i pobierania wyników.
- Dwa tryby działania
 - Tryb połączeniowy: DataReader
 - Tryb bezpołączeniowy: DataAdapter

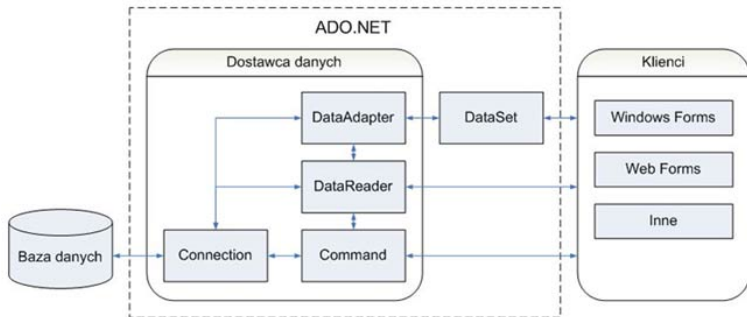
Wprowadzenie do ADO.NET

- DataAdapter ma zdefiniowane komendy
 - SELECT,
 - INSERT,
 - UPDATE,
 - DELETE,

które służą do synchronizacji bazy danych z obiektem DataSet

- DataSet
 - Składa się z DataTables, Relations i Constraints
 - Służy do reprezentacji stanu bazy danych (lub jej fragmentu)

Wprowadzenie do ADO.NET



Źródło: http://www.microsoft.com/poland/developer/net/programowanie/dostep_do_baz.aspx

Wprowadzenie do ADO.NET

- Interfejsy czterech głównych składowych:
 - IDbConnection
 - IDbCommand
 - IDataReader
 - IDbDataAdapter
- Implementacje mają odpowiednie przerostki
 - Sql*, OleDb*, Oracle*, Odbc*
- Interfejsy są dostępne w przestrzeni nazw System.Data

Wprowadzenie do ADO.NET

Dzięki interfejsom możemy pisać uniwersalny kod

```
public IDataReader getData(con As IDbConnection)
{
    IDbCommand cmd = con.CreateCommand();
    cmd.CommandText = "SELECT * FROM Customers";
    IDataReader reader = cmd.ExecuteReader();
    return reader;
}
```

Nawiązywanie połączenia

- Korzystamy z implementacji interfejsu `IDbConnection`
 - dla SQL Server – `SqlConnection`,
 - dla Oracle – `OracleConnection`.
- Do określenia lokalizacji bazy danych podajemy odpowiedni *connection string*, w którym definiujemy parametry:
 - Data source,
 - Initial Catalog,
 - Integrated Security,
 - User Id, Password,
 - Connection Timeout.

Nawiązywanie połączenia

- Obiekty `Connection` udostępniają zdarzenia
 - `StateChange` – odpalane, gdy połączenie zmieni stan z zamkniętego na otwarte i odwrotnie.
 - `InfoMessage` – odpalane, gdy *provider* wysyła ostrzeżenie lub komunikat informacyjny do klienta.
- Jeśli szukamy właściwego *connection string*, warto zajrzeć na
 - <http://www.connectionstrings.com/>
 - http://www.codeproject.com/cs/database/sql_in_csharp.asp

Nawiązywanie połączenia

SQL Server Connection Pooling (ADO.NET)

- Utworzenie połączenia kosztuje
 - otworenie portu
 - *handshake* z serwerem
 - analiza *connection string*
 - uwierzytelnienie
 - ...
- Pula połączeń pozwala na optymalizację
 - jeśli połączenie jest już nawiązane, inni klienci mogą z niego skorzystać
 - pula ma szereg parametrów, np. max/min liczba połączeń
 - jest po stronie sterownika klienta

SQL Server Connection Pooling (ADO.NET)

- Pula jest określana
 - per process
 - per application domain
 - per connection string
 - per Windows identity (w przypadku integrated security)
- W przypadku, gdy pula się wyczerpie, przy próbie nawiązania połączenia pojawi się wyjątek (po timeout'cie)
- Czyszczenie puli
 - metody ClearAllPools, ClearPool

SQL Server Connection Pooling (ADO.NET)

- Problem fragmentacji
 - gdy łączy się wielu użytkowników
 - gdy jest wiele baz danych (sztuczka: łączyć się do jednej, a potem USE db)
- Do poczytania:
 - <http://msdn.microsoft.com/en-us/library/8xx3tyca.aspx>
 - <http://www.codeproject.com/Articles/17768/ADO-NET-Connection-Pooling-at-a-Glance>

Wykonywanie kwerend

- Korzystamy z implementacji interfejsu `IDbCommand`
 - w naszych rozważaniach będzie to *SqlCommand*
- Definiujemy rodzaj zapytania poprzez ustawienie `SqlCommand.CommandType` na
 - `CommandType.Text` – tekst zapytania (domyślne)
 - `CommandType.TableDirect` – nazwa tabeli, z której zostaną zwrócone kolumny (niezaimplementowane)
 - `CommandType.StoredProcedure` – nazwa procedury składowanej

Wykonywanie kwerend

- Tworzymy obiekt klasy `SqlCommand` jednym z 4 konstruktorów
- Ważniejsze właściwości obiektu `SqlCommand`
 - `Connection`
 - `CommandText`
 - `Parameters`
 - `Transaction`
- Do wykonania zapytania mamy metody
 - `ExecuteNonQuery()`
 - `ExecuteReader()`
 - `ExecuteScalar()`
 - `ExecuteXmlReader()` (tylko `SqlCommand`)

Wykonywanie kwerend

Metoda **ExecuteNonQuery**

- Służy do wykonywania zapytań typu insert, update, delete
- Wynik to liczba wierszy, których dotyczyło zapytanie
- Dla zapytań innego niż wymienionego wyżej typu zwracane jest -1

Metoda **ExecuteReader**

- Służy do pobrania pełnego wyniku zapytania
- Wynik to obiekt implementujący interfejs `IDataReader`
 - my będziemy używać obiektów `SqlDataReader`

Metoda **ExecuteReader** (c.d.)

- Wykonując metodę możemy opcjonalnie podać argument typu `CommandBehavior` o następujących stałych
 - `CloseConnection` – zamknięcie `DataReader`-a powoduje zamknięcie skojarzonego z nim połączenia
 - `SequentialAccess` – pozwala odczytywać dane sekwencyjnie na poziomie kolumn za pomocą metod `GetChars` i `GetBytes`
 - Przydatne przy obsłudze danych typu BLOB (np. obrazków)
 - Opis tego, jak z skorzystać jest na stronie pod adresem:
<http://www.developers.ie/ShowArticle.aspx?id=b7a73799-3965-4c0b-b7ad-943aa4dde944>

Metoda **ExecuteReader** (c.d.)

- CommandBehavior c.d.
 - SchemaOnly – pobierana jest tylko informacja o kolumnach bez pobierania danych
 - co można pobrać jest dostępne na stronie pod adresami:
<http://msdn2.microsoft.com/en-us/library/system.data.idatareader.getschematable.aspx>
<http://msdn2.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader.getschematable.aspx>
 - SingleResult – pobierany jest jeden ResultSet
 - SingleRow – pobierany jest pojedynczy wiersz wyniku

Metoda **ExecuteScalar**

- Służy do pobrania 1-go kolumny 1-go wiersza wyniku
- Używane zwykle do pobrania wyników funkcji agregujących
- Działa wydajniej niż `ExecuteReader`

Metoda **ExecuteXmlReader**

- Do zapytania musimy dopisać `FOR XML AUTO, XMLDATA`
- Metoda zwraca obiekt klasy `XmlReader`
- Więcej informacji można dostać na stronie pod adresem
[http://msdn2.microsoft.com/en-us/library/system.data.sqlclient.sqlcommand.executexmlreader\(vs.80\).aspx](http://msdn2.microsoft.com/en-us/library/system.data.sqlclient.sqlcommand.executexmlreader(vs.80).aspx)

Przeglądanie wyniku zapytania

- Wynik jest reprezentowany przez obiekt klasy `SqlDataReader`
 - Zwracany przez metodę `ExecuteReader`
- `SqlDataReader` może zawierać wyniki kilku zapytań
- Metoda `reader.Read()` pobiera kolejne rekordy wyniku
- Metoda `reader.NextResult` pobiera kolejny wynik
- Sposoby pobrania wartości pól danego wiersza
 - Poprzez metody `GetTYPE(int)` (numerowanie od 0)
 - Poprzez odwołanie `reader[nr_kolumny]` (numerowanie od 0)
 - Poprzez odwołanie `reader[nazwa_kolumny]`
- Na końcu należy pamiętać o zamknięciu poprzez metodę `SqlDataReader.Close()`

Przeglądanie wyniku zapytania

Krótką statystyka dotycząca wydajności odwołań

Provider	Metoda	Czas w sek. (dla 10 milionów rekordów)
SqlClient	reader["ID"]	64.192304
	reader[0]	48.3995952
OleDb	reader["ID"]	152.5092976
	reader[0]	131.1285536

Źródło: Bipin Joshi, et al., *Professional ADO.NET Programming*, Wrox

Ogólny schemat przetwarzania wierszy wyników

```
do {  
    while (result.Read()) {  
        //Przetwarzanie wiersza  
    }  
} while (result.NextResult());
```

Parametryzacja kwerend

- Obiekt `SqlCommand` ma kolekcję `Parameters`
- Parametry są reprezentowane przez obiekty `SqlParameter`
- Domyślnie parametry są typu `IN` (mogą być też `OUT` i `IN OUT`)
- Parametry służą do obsługi zapytań, jak i wywołań procedur

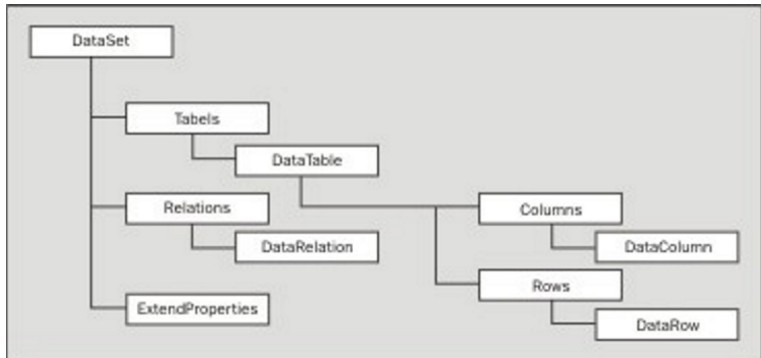
Przykłady

- `ConnectionExample`
- `ConnectionEvents`
- `ExecuteCommands`
- `ExecuteCommands2`

Wprowadzenie

- Jest głównym komponentem w bezpołączeniowym modelu tworzenia aplikacji
- Zawiera tabele, relacje i więzy
- Umożliwia modyfikacje tych danych, a później synchronizację z bazą danych
- Używa XML-a jako formatu przesyłania danych

Obiekt DataSet



Źródło: Bipin Joshi, et al., *Professional ADO.NET Programming*, Wrox

Obiekt DataSet

DataSet składa się z

- Kolekcji tabel
 - Dostępna przez `DataSet.Tables` (`DataTableCollection`)
 - Zawiera zero lub więcej obiektów typu `DataTable`
 - `DataTable` reprezentuje pojedynczą tabelę
- Kolekcji relacji
 - Dostępna przez `DS.Relations` (`DataRelationCollection`)
 - Zawiera zero lub więcej obiektów typu `DataRelation`
 - `DataRelation` reprezentuje relację typu *parent-child*
- Kolekcji `ExtendedProperties`
 - Pary klucz/wartość, które można dowolnie definiować

Obiekt DataTable

DataTable składa się z

- Kolekcji kolumn dostępnej przez `DataTable.Columns`
(instancja klasy `DataColumnCollection`)
- Kolekcji wierszy dostępnej przez `DataTable.Rows`
(instancja klasy `DataRowCollection`)
- Kolekcji ograniczeń dostępnej przez `DataTable.Constraints`
(instancja klasy `ConstraintCollection`)

Obiekt DataTable

DataColumn

- Reprezentuje pojedynczą kolumnę
- Możemy utworzyć jednym z 5-ciu konstruktorów
 - chociaż używać będziemy głównie dwuparametrowego:
`DataTableColumn(string columnName, Type dataType)`
- Nową kolumnę możemy dodać do kolekcji na kilka sposobów
 - `tabela.Columns.Add()`
 - dodawane są kolumny o nazwach *Column1*, *Column2*, ...
 - `tabela.Columns.Add(Column c)`
 - `tabela.Columns.Add(string colName, Type dataType)`

Obiekt DataTable

DataRow

- Reprezentuje pojedynczy wiersz
- Możemy dodać na dwa sposoby:
 - Za pomocą metody `table.Rows.Add(lista_wartosci)`
 - Korzystając z metody `table.NewRow()`
 - Metoda zwraca DataRow o kolumnach odpowiadających schematowi table
 - Uzupełniamy wartości pól zwróconego rekordu
 - Dodajemy za pomocą metody `table.Rows.Add(rekord)`

Zdarzenia

- Możemy obsługiwać 6 następujących zdarzeń:
 - ColumnChanging, ColumnChanged
 - RowChanging, RowChanged
 - RowDeleting, RowDeleted
- Metody do obsługi powyższych zdarzeń mają jako parametry:
 - Obiekt DataColumnChangeEventArgs ze składowymi
 - Column, ProposedValue, Row
 - Obiekt DataRowChangeEventArgs ze składowymi
 - Action, Row

Obiekt DataTable

Więzy integralności

- Mamy dwie klasy w przestrzeni System.Data
 - UniqueConstraint
 - ForeignKeyConstraint
- Więzy są dostępne w `DataTable.Constraints` jako kolekcja `ConstraintsCollection`
- Wymuszenie sprawdzania więzów integralności jest określone przez właściwość `DataSet.EnforceConstraints`
- Przy scalaniu danych, więzy są nakładane dopiero po scaleniu `DataSet-ów`
 - dzięki temu działa to szybciej, ponieważ sprawdzanie nie musi być przy każdym wierszu

Obiekt DataTable

Więzy integralności

- Więzy są sprawdzane przy modyfikacji danych, w szczególności sprawdzanie wywołują metody:
 - `DataSet.Merge`, `DataTable.LoadDataRow`
 - `DataRowCollection.Add`, `DataRow.EndEdit`
 - `DataRow.ItemArray`
- Mamy dwa wyjątki do obsługi niepomyślnych sytuacji:
 - `ConstraintException` — pojawi się wtedy, gdy wiąz zostanie naruszony (np. przy edycji danych)
 - `InvalidConstraintException` — pojawi się wtedy, gdy chcemy uruchomić wiąz na danych, które tego wiążu nie spełniają (np. ustawiając `EnforceConstraints` na `true`)

Więzy integralności: UniqueConstraint

- Ten rodzaj więzów gwarantuje unikalność wartości kolumn
 - dopuszcza wartość NULL
 - jeśli nie chcemy dopuszczać wartości NULL, możemy użyć PK lub ustawić kolumnom property `AllowDBNull` na `false`
- Tworzenie może się odbyć na dwa sposoby:
 - 1 `tabela.kolumna.Unique = true`
 - tylko w przypadku nakładania wiążu na jedną kolumnę
 - 2 `UniqueConstraint c = new UniqueConstraint(coltab)`
`tabela.Constraints.Add(c)`

Więzy integralności: ForeignKeyConstraint

- Tworzy spójność danych pomiędzy tabelami *parent-a* i *child-a*
- Można tworzyć powiązania jedno- lub wielokolumnowe
- Właściwość `AcceptRejectRule` określa zachowanie przy wykonaniu metod `AcceptChanges` lub `RejectChanges` i ma wartości
 - Cascade, None
- Właściwości `UpdateRule` i `DeleteRule` określają zachowanie w momencie zmiany wiersza i mają wartości
 - Cascade, None, SetDefault, SetNull

Więzy integralności: Tworzenie własnych wiązków

- Nie ma wbudowanych mechanizmów
- Możemy skorzystać ze schematu:
 - Tworzymy klasę, która będzie miała metodę M1 do sprawdzenia, czy wartość spełnia zadane kryteria
 - Dodajemy do klasy metodę M2, która sprawdza zawartość DataTable pod kątem zadanego kryterium
 - Napełniamy DataTable danymi
 - Odpalamy metodę M2
 - jeśli błąd, rzucamy wyjątek `InvalidConstraintException`
 - Tworzymy metodę do obsługi zdarzenia `ColumnChanged`, w której odpalamy metodę M1
 - jeśli błąd, rzucamy wyjątek `ConstraintException`

Klucz główny

- Klucz główny to tablica obiektów DataColumn dostępna w `DataTable.PrimaryKey`
- Jest możliwość zdefiniowania klucza głównego
 - jako tylko do odczytu,
 - generowanego automatycznie,
 - z określonym krokiem kolejnych identyfikatorów.

Przykłady

- DataTableExample
- DataTableExample2
- ConstraintsUnique
- ConstraintsForeignKey

Obiekt `DataRelation`

- `DataSet.Relations` instancją klasy `DataRelationsCollection`
- `DataSet.Relations` zawiera obiekty klasy `DataRelation`
 - tworzą związki *child-parent* pomiędzy tabelami `DataTable`
- Mamy dwa główne sposoby utworzenia relacji
 - Utworzenie obiektu `DataRelation` i dodanie do kolekcji
 - mamy do dyspozycji 6 konstruktorów
 - Wywołanie metody `ds.Relations.Add(...)`
 - mamy do dyspozycji 7 sygnatur

Obiekt DataRelation

- Parametry przy tworzeniu relacji:
 - nazwa relacji (może być tylko jedna relacja o danej nazwie)
 - można jako nazwę podać "" — wtedy tworzona jest nazwa domyślna i może być kilka relacji o takiej nazwie
 - kolumnę/y tabeli *parent-a*
 - kolumnę/y tabeli *child-a*
 - flaga, czy ma być tworzony odpowiedni więz (*constraint*)
- Należy pamiętać, aby kolumny określające *parent-a* i kolumny określające *child-a* miały
 - tą samą liczebność
 - zgodne typy

- `DataRelationExample`

Scalanie danych

- Służy do tego metoda `DataSet.Merge()`
- Dołącza do danego zbioru danych nowe elementy.
- Można dołączyć
 - Inny `DataSet`
 - Tabelę
 - Zbiór wierszy
- Parametry określają
 - W przypadku konfliktu, który `DataSet` ma priorytet
 - Co zrobić w przypadku niezgodności schematów
- `DataSet.Merge` ma 7 sygnatur

- DataSetMerging

Obiekt **DataAdapter**

- Mechanizm do pobierania treści z bazy danych
 - tryb bezpołączeniowy w przeciwieństwie do *DataReader-a*
- Jeśli ściśle powiązany z obiektem DataSet
- Hierarchia klas implementujących jest następująca:

DataAdapter

 DbDataAdapter (implementuje IDbDataAdapter)

 SqlDataAdapter

 OleDbDataAdapter

 OdbcDataAdapter

W zakresie naszych zainteresowań będzie SqlDataAdapter

Obiekt DataAdapter

Jak to działa?

- Tworzymy obiekt DataAdapter
- DataAdapter wypełnia obiekt DataSet (DataTable)
 - DataAdapter pobiera dane z zapytania i składa je w DataSet jako tabelę o pewnej nazwie
 - Z powyższego wynika, że to my decydujemy jak będzie wyglądało odwzorowanie bazy danych na DataSet
- Z każdą z operacji select, insert, update i delete jest powiązana odpowiednia kwerenda

Obiekt **DataAdapter**

Ważniejsze składowe **DataAdapter**

- Metoda `Fill`
- Metoda `FillSchema`
- Właściwość `MissingSchemaAction`

Ważniejsze parametry metod `Fill` i `FillSchema`

- `DataSet` lub `DataTable`
- Nazwa tabeli w `DataSet`
- Parametry określające wiersze do pobrania (`Fill`)
- `SchemaType.Mapped` lub `SchemaType.Source` (`FillSchema`)

Mapowanie struktury

- Podejście „klasyczne” — operator AS w kwerendzie SQL
- Z wykorzystaniem mechanizmów ADO.NET
 - Najpierw tworzymy obiekt DataTableMapping
 - Obiekt ten ma kolekcję ColumnMappings
 - Dodajemy do tej kolekcji obiekty DataColumnMapping, czyli odpowiednie mapowania
 - Na końcu obiekt DataTableMapping dodajemy do kolekcji DataAdapter.TableMappings

Obiekt DataAdapter

Mapowanie struktury

- Enumeracja `MissingMappingAction` określa zachowanie przy braku zdefiniowanego mapowania i ma wartości
 - `Passthrough` — nazwa kolumny z tabeli ze źródła danych
 - `Error` — rzucenie wyjątku `SystemException`
 - `Ignore` — ignorowanie niezdefiniowanej kolumny
- Enumeracja `MissingSchemaAction` określa zachowanie przy braku schematu dla danej kolumny i ma wartości
 - `Add` — dodanie kolumny do schematu (bez PK i UNIQUE)
 - `AddWithKey` — j.w., ale z PK i UNIQUE
 - `Ignore` — zignorowanie i dalsze przetwarzanie
 - `Error` — rzucenie wyjątku `SystemException`

Przykłady

- MappingsExample
- DataAdapterExample
- DataAdapterExample2
- DataAdapterExample3

Obiekt `DataAdapter`

Aktualizacja bazy danych

- Enumeracja `DataRowState` określa stan wiersza
 - `Added` — wiersz dodany do `DataRowCollection`
 - `Deleted` — wiersz usunięty za pomocą metody `Delete`
 - `Detached` — wiersz jest utworzony, ale nie jest częścią `DataRowCollection`
 - Stan pojawia się zwykle po utworzeniu wiersza zanim zostanie dodany do kolekcji lub tuż po jego usunięciu
 - `Modified` — wiersz został zmodyfikowany
 - `Unchanged` — wiersz jest bez zmian
- Stan wiersza jest dostępny w `DataRow.RowState`

Obiekt DataAdapter

Aktualizacja bazy danych

- Metoda `Fill` napełniając `DataSet` ustawia każdemu wierszowi stan `Unchanged`
- Do aktualizacji danych w bazie danych służy metoda `Update`
 - metoda przegląda wszystkie wiersze, sprawdza ich stan i w zależności od potrzeby wykonuje jedno ze zdefiniowanych wcześniej zapytań `INSERT`, `UPDATE` lub `DELETE`
- Kwerendy `INSERT`, `UPDATE` i `DELETE` można zdefiniować
 - Korzystając z obiektu `CommandBuilder`
 - my skorzystamy z implementacji `SqlCommandBuilder`
 - Samodzielnie poprzez utworzenie odpowiednich kwerend SQL

Aktualizacja bazy danych

- Przydatne są metody
 - `DataSet.HasChanges()` — informuje, czy coś zostało zmienione
 - `DataSet.GetChanges()` — zwraca `DataSet` zawierający tylko zmodyfikowane rekordy (tabele)
- Użycie powyższych metod redukuje zwykle ruch w sieci przy aktualizacji

Zastosowanie SqlCommandBuilder

- Użycie SqlCommandBuilder to najprostsze rozwiązanie
- SqlCommandBuilder analizuje zapytanie SELECT i automatycznie generuje pozostałe trzy zapytania
- Warunki zastosowania SqlCommandBuilder
 - 1 Zapytanie SELECT jest zdefiniowane i pobiera dane z tylko jednej tabeli
 - 2 Na liście kolumn zapytania SELECT musi zostać określony klucz główny lub kolumna unikalna (UNIQUE constraint)

Samodzielne tworzenie kwerend poprzez SqlCommand

- W przypadku, gdy nie możemy lub nie chcemy wykorzystywać SqlCommandBuilder, możemy utworzyć kwerendy samodzielnie
- Robimy to przez zdefiniowanie właściwości InsertCommand, UpdateCommand i DeleteCommand
 - wszystkie są typu SqlCommand lub OleDbCommand
 - dla SqlCommand parametry to np. @param
 - dla pozostałych (np. OleDbCommand) parametry to ?

Samodzielne tworzenie kwerend poprzez SqlCommand

- Definiowanie kwerend InsertCommand, UpdateCommand i DeleteCommand
 - Przy definiowaniu kwerend musimy także zdefiniować parametry, czyli obiekty Parameter (np. SqlParameter)
 - Tworząc obiekt np. SqlParameter możemy podać szereg argumentów: parameterName, dbType, size, direction, isNullable, precision, scale, sourceColumn, sourceVersion, value

Obiekt DataAdapter

Wersje DataRow

- Mamy dostępne cztery stałe:
 - Default — wartości domyślne
 - Current — bieżące wartości wiersza
 - Original — wartości wiersza z bazy danych
 - Proposed — nowe wartości (proponowane)
 - dostępne po rozpoczęciu edycji (`row.BeginEdit()`),
 - ale przed wywołaniem `row.EndEdit()`, `row.CancelEdit()`,
`ds.AcceptChanges()` lub `ds.RejectChanges()`
- Ogólna sekwencja czasowa: Original → Current → Proposed
- Więcej na temat (+rozważania):

<http://msdn.microsoft.com/en-us/library/system.data.datarowversion.aspx>

Obiekt DataAdapter

Zdarzenia

- Mamy dwa istotne zdarzenia: `RowUpdating` i `RowUpdated`
- Metoda obsługująca `RowUpdating` bierze parametr `SqlRowUpdatingEventArgs`, którego istotniejsze właściwości
 - `Command` (w tym kolekcja `Parameters`)
 - `Row`
 - `StatementType` (np. `INSERT`)
- Metoda obsługująca `RowUpdated` bierze parametr `SqlRowUpdatedEventArgs`, który ma powyższe właściwości oraz dodatkowo:
 - `RecordsAffected`

Przykłady

- DataAdapterUpdates
- DataAdapterUpdates1
- DataAdapterUpdates2
- DataAdapterUpdates3

Transakcje

- Przypomnienie pojęcia transakcji
 - przykład: przelew z konta na konto
- Własność ACID
 - Atomowość, czyli wszystko w transakcji albo nic
 - Spójność, czyli baza cały czas zawiera poprawne dane
 - Izolacja, czyli transakcje są od siebie odseparowane
 - Trwałość, czyli dane cały czas są dostępne
- Niektóre z reguł ACID będziemy trochę naruszać, np.
 - Atomość (tworząc transakcje zagnieżdżone, savepoints)
 - Izolacja (poprzez różne poziomy izolacji)
 - Spójność (przy np. większych importach danych)

Transakcje

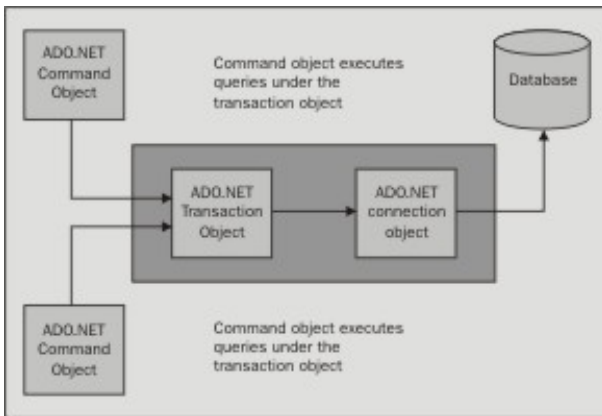
- Trzy główne polecenia do zarządzania transakcjami:
 - BEGIN,
 - COMMIT,
 - ROLLBACK
- ADO.NET dostarcza dwa modele dostępu do danych
 - połączeniowego
 - bezpołączeniowego
- Transakcje są obsługiwane w obu powyższych modelach

Typowy scenariusz w środowisku połączeniowym

- Otwarcie połączenia do bazy danych
- Rozpoczęcie transakcji
- Wykonanie zapytań w kontekście połączenia i transakcji, wykorzystując obiekt `Command`
- Zatwierdzenie lub wycofanie transakcji
- Zamknięcie połączenia

Transakcje

Typowy scenariusz w środowisku połączeniowym



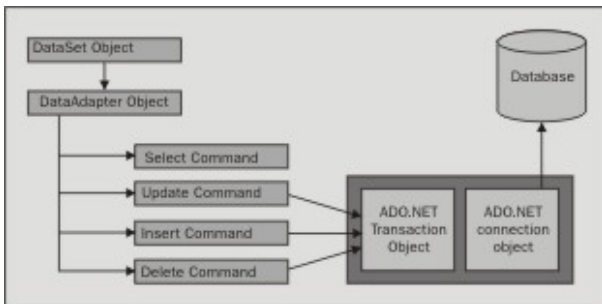
Źródło: Bipin Joshi, et al., *Professional ADO.NET Programming*, Wrox

Typowy scenariusz w środowisku bezpołączeniowym

- Otwarcie połączenia do bazy danych
- Pobranie danych do obiektu DataSet
- Zamknięcie połączenia do bazy danych
- Operacje na danych w obiekcie DataSet
- Ponowne otwarcie połączenia do bazy danych
- Rozpoczęcie transakcji
- Wykonanie aktualizacji bazy danych na podstawie zmian w obiekcie DataSet
- Zamknięcie połączenia do bazy danych

Transakcje

Typowy scenariusz w środowisku bezpołączeniowym



Źródło: Bipin Joshi, et al., *Professional ADO.NET Programming*, Wrox

Klasa **Transaction**

- Każdy *provider* ma swoją implementację klasy obsługującej transakcje
 - Wszystkie klasy transakcji implementują interfejs `System.Data.IDbTransaction`
 - Nas będzie interesować obiekt `SqlTransaction`
- Mamy dwie podstawowe metody: `Commit()` i `Rollback()`
- W ramach połączenia może być tylko jedna transakcja
- Wszystkie komendy skojarzone z danym połączeniem muszą mieć ustawioną transakcję skojarzoną z tym połączeniem

Tworzenie transakcji

- Transakcja jest tworzona w ramach połączenia:
 - `SqlConnection tx = connection.BeginTransaction()`
- Przy tworzeniu możemy przekazać parametry:
 - poziom izolacji transakcji,
 - nazwę transakcji

Poziomy izolacji

- Do dyspozycji mamy poziomy izolacji:
 - `ReadUncommitted, ReadCommitted, RepeatableRead, Serializable`
 - Dostępne są w enumeracji `System.Data.IsolationLevel`
- Poziom izolacji możemy odczytać: `tx.IsolationLevel`

Punkty bezpieczeństwa (*savepoints*)

- Pozwalają na „zatwierdzenie” części transakcji i ewentualnie wycofanie części „niezatwierdzonej”
- Służą do tego metody `tr.Save(name)` i `tr.Rollback(name)`

Zagnieżdżenie transakcji

- Tworzymy poprzez konstrukcję

```
OleDbConnection conn = new OleDbConnection();  
OleDbTransaction tx = conn.BeginTransaction();  
OleDbTransaction nested_tx = tx.Begin();
```

- Niedostępne dla `SqlConnection`

Kwestie wydajności

- Transakcji powinno się używać kiedy naprawdę trzeba
- Transakcje powinny być jak najkrótsze
- Należy unikać instrukcji SELECT
 - Jeśli już jej użyjemy, należy pobierać dokładnie te wiersze i kolumny, które są nam potrzebne, w szczególności `SELECT * FROM T` jest bardzo niedobrym pomysłem
- Czasami warto przenieść obsługę transakcji na poziom procedur składowanych
- Należy unikać umieszczenia kilku wsadów w jednej transakcji
 - każdy wsad powinien być w osobnej transakcji

Przykłady

- TransactionsExample
- TransactionsExample2
- TransactionsExample3
- TransactionsIsolationLevel
- TransactionsSavePoints
- TransactionsDataSet