Acceso a Datos ADO.NET

Integración de Sistemas

Diseño e implementación con .NET

Introducción a ADO.NET

- ADO.NET es el modelo de acceso a datos para las aplicaciones basadas en .NET
- Se puede utilizar para acceder a sistemas de base de datos relacionales. Ejemplos:
  - SQL Server, Oracle, etc.
  - Muchas otras fuentes de datos (para las cuales existe un proveedor OLE DB u ODBC)
- Soporte intrínseco para XML
- Programación Orientada a Componentes

ADO.NET
Evolución histórica

- ODBC (Open DataBase Connectivity)
  - Interoperatibilidad con amplio rango de SGBD
  - API acceso ampliamente aceptada
  - Usa SQL como lenguaje de acceso a datos
- DAO (Data Access Objects)
  - Interfaz de programación para bases de datos JET/ISAM (e.g. MS Access)

ADO.NET
Evolución histórica

- RDO (Remote Data Objects)
  - Estrechamente ligado a ODBC
  - Orientado a aplicaciones cliente/servidor
- OLE DB (Object Linking and Embedding for Databases)
  - No restringrido a acceso a datos relacionales
    - No limitado a SQL como lenguaje de recuperación de datos
  - Tecnología desarrollada por Microsoft
    - Construido sobre COM (Component Object Model)
  - Proporciona una interfaz a bajo nivel en C++

ADO.NET
Evolución histórica

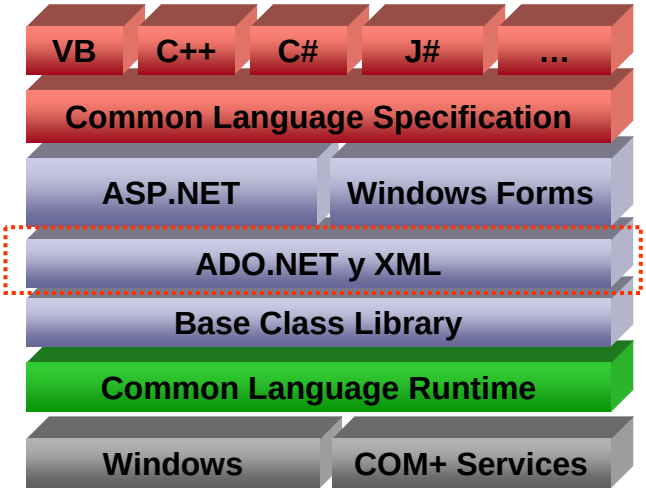
- ADO (ActiveX Data Objects)
  - Ofrece una interfaz orientada a objetos
  - Proporciona un modelo de programación para OLE DB accesible desde lenguajes diferentes a C++
  - Diseñado como modelo conectado, altamente acoplado
    - Indicado para arquitecturas cliente/servidor
  - No pensado para arquitecturas multicapa en entornos distribuidos
  - Diseño no correctamente factorizado
    - Demasiadas maneras de hacer lo mismo
    - Algunos objectos acaparan demasiada funcionalidad

ADO.NET
¿Qué es ADO.NET?

- Colección de clases, interfaces, estructuras y enumeraciones que permiten acceder a diversas fuentes de datos (BD, XML, etc.) desde la plataforma .NET
  - Organizada en *namespaces*:
    - **`System.Data`**
    - **`System.Data.Common`**
    - **`System.Data.OleDb`**
    - **`System.Data.SqlClient`**
    - Etc.
- Evolución de ADO
  - No comparte con éste su jerarquía de clases
  - Sí comparte su funcionalidad
  - Usa internamente XML

ADO.NET

Arquitectura Framework .NET



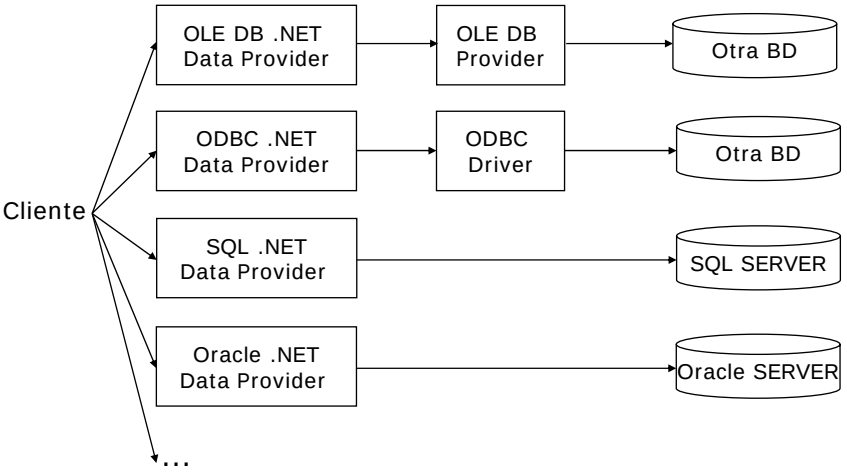
ADO.NET

Conceptos Básicos y Arquitectura

- Modelo de objetos
  - Objetos de **`System.Data`**
  - Proveedores de acceso a datos .NET
- Jerarquía de espacio de nombres (*namespace*)
  - Organiza el modelo de objetos
  - Incluye:
    - **`System.Data`**
    - **`System.Data.Common`**
    - **`System.Data.OleDb`**
    - **`System.Data.SqlClient`**
    - **`System.Data.SqlTypes`**

ADO.NET

Proveedores de Acceso a Datos (Data Providers)



ADO.NET

Proveedores de Acceso a Datos

- Proporcionan un conjunto de clases que implementan una serie de interfaces comunes
- ADO.NET
  - OLE DB
    - Acceso vía protocolo OLE DB a cualquier fuente de datos que lo soporte
    - **`System.Data.OleDb`**
  - ODBC
    - Acceso vía protocolo ODBC a cualquier fuente de datos que lo soporte
    - **`System.Data.Odbc`**
  - SQL Server
    - Acceso nativo a MS SQL Server 7.0 o superior y MS Access
    - **`System.Data.SqlClient`**
  - Oracle
    - Acceso nativo a Oracle Server
    - **`System.Data.OracleClient`**
- Otros provistos por terceros
    - MySQL, PostgreSQL, DB2, etc.

ADO.NET

Proveedores de Acceso a Datos

```
public static void ListProviders() {

   DataTable factoryTable = DbProviderFactories.GetFactoryClasses();
   foreach (DataRow dr in factoryTable.Rows) {
        Console.WriteLine("Name: {0}", dr["Name"]);
        Console.WriteLine("Description: {0}", dr["Description"]);
        Console.WriteLine("InvariantName: {0}", dr["InvariantName"]);
        Console.WriteLine("AssemblyQualifiedName: {0}",
                dr["AssemblyQualifiedName"]);
   }
}

// Sample Output

//Name: SqlClient Data Provider
//Description: .Net Framework Data Provider for SqlServer
//InvariantName: System.Data.SqlClient
//AssemblyQualifiedName: System.Data.SqlClient.SqlClientFactory,
//   System.Data, Version=2.0.0.0, Culture=neutral,
//   PublicKeyToken=b77a5c561934e089
```

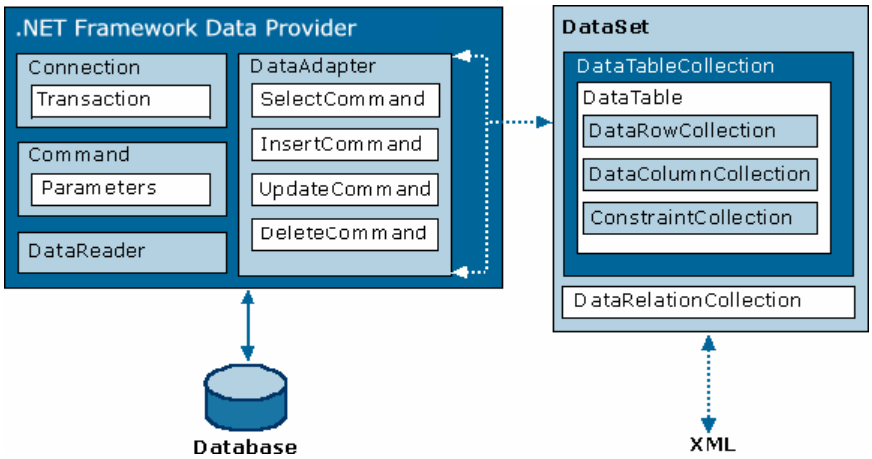
ADO.NET

ADO.NET 2.0 – API Independiente

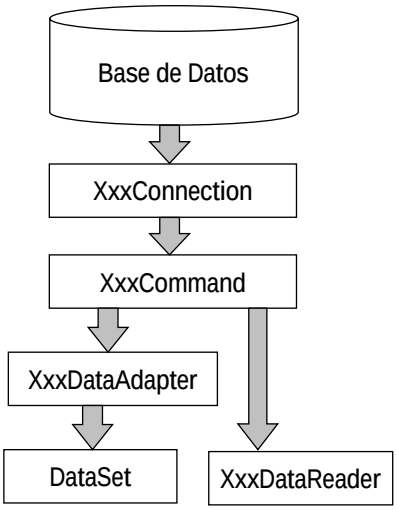
- Namespace **`System.Data.Common`**

DbCommand	**DbCommandBuilder**	**DbConnection**
DataAdapter	**DbDataAdapter**	**DbDataReader**
DbParameter	**DbParameterCollection**	**DbTransaction**
DbProviderFactory	**DbProviderFactories**	**DbException**

ADO.NET
Arquitectura



ADO.NET
Arquitectura



- **XxxConnection:** maneja la conexión a una BD
- **XxxCommand**: ejecuta comandos contra una BD
- **XxxDataAdapter**: intercambia datos entre un DataSet y una BD
- **DataSet**: copia local de datos relacionales (Entorno Desconectado)
- **XxxDataReader**: Proporciona acceso a datos Read-only, Forward-only (Entorno Conectado)

ADO.NET
Entornos de Acceso a Datos

- Conectado:
  - forward-only, read-only
  - Aplicación realiza una consulta y lee los datos conforme los va procesando
  - Cursor unidireccional
  - Objeto **`DataReader`**
- Desconectado
  - La aplicación ejecuta la consulta y almacena los resultados de la misma para procesarlos después
  - Minimiza el tiempo de conexión a la base de datos
  - Objetos *lightweight*
  - Objetos **`DataSet`** y **`DataAdapter`**

ADO.NET
Namespace **`System.Data`**

- Contiene la base de ADO.NET
- Proporciona el modo de trabajo sobre datos
  - Clases y métodos para la manipulación de datos
  - Posibilidad de creación de vistas
  - Representación lógica de los datos
  - Utilización de XML para visualizar, compartir y almacenar datos

ADO.NET
Namespace **`System.Data`**

- Acceso Independiente a Datos
  - **`System.Data.IDbConnection`**
  - **`System.Data.IDbCommand`**
  - **`System.Data.IDbDataParameter`**
  - **`System.Data.IDbTransaction`**
  - **`System.Data.IDataReader`**
  - **`System.Data.IDataAdapter`**

ADO.NET
`DbConnection`

- Establece una sesión con una fuente de datos
- Implementada por **`SqlDbConnection`**, **`ODBCConnection`**, **`OleDbConnection`**, etc.
- Funcionalidad
  - Abrir y Cerrar conexiones
  - Gestionar Transacciones
- Usada conjuntamente con objetos **`DbCommand`** y **`DataAdapter`**
- Propiedades adicionales, métodos y colecciones dependerán de la implementación específica del proveedor

ADO.NET
`DbConnection`

- Propiedades
  - **`ConnectionString`**
  - **`ConnectionTimeOut`**
  - **`DataBase`**
  - **`State`**
    - **`Open`**
    - **`Close`**
- Métodos
  - **`void Open()`**
  - **`void Close()`**
  - **`void ChangeDataBase(dbName);`**
  - **`DbCommand CreateCommand()`**

ADO.NET
`DbConnection`. Connection String

- Dependerá del proveedor de acceso a datos
- http://www.codeproject.com/database/connectionstrings.asp
- Ejemplos:
  - SqlServer:

```
"Data Source=localhost\SQLExpress; Initial
 Catalog=miniportal; User ID=user;Password=password"
```

  - MySQL ODBC Driver:

```
"DRIVER={MySQL ODBC 3.51 Driver}; SERVER=localhost;
 PORT=3306; UID=user; PWD=pwd; DATABASE=db;"
```

  - Access OLEDB:

```
"Provider=MSDASQL; Driver={Microsoft Access Driver
 (*.mdb)}; Dbq=drive:\path\file.mdb; Uid=user; Pwd=pwd";
```

ADO.NET

Connection Pooling

- Pool de Conexiones **habilitado automáticamente**
  - Pool se crea en base a la cadena de conexión
  - Al cerrar una conexión, ésta se devuelve al pool

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString = "Integrated Security=SSPI;Initial
  Catalog=northwind";
conn.Open();  // Pool A is created.

SqlConnection conn = new SqlConnection();
conn.ConnectionString = "Integrated Security=SSPI;Initial
  Catalog=pubs";
conn.Open(); // Pool B is created because the connection
  strings differ.

SqlConnection conn = new SqlConnection();
conn.ConnectionString = "Integrated Security=SSPI;Initial
  Catalog=northwind";
conn.Open(); // The connection string matches pool A.
```

ADO.NET

Ejemplo Creación de Conexiones

```
using System.Data;
using System.Data.SqlClient;
   <<...>>
   String connectionString = "DataSource=localhost\\SQLExpress;" +
                             "Initial Catalog=miniportal;" +
                             "UserID=user;Password=password";
   SqlConnection connection = new SqlConnection();
   connection.ConnectionString = sqlConnectionString;

   connection.Open();
   <<...>>
   connection.Close();
```

ADO.NET
Providers Factory

- Independizar código del proveedor de datos
  - Factoría de proveedores : **`DbProviderFactories`**
  - Crea instancias de la implementación de un proveedor de las clases de origen de datos
- Objeto **`DbProviderFactory`**
  - **`.CreateCommand()`**
  - **`.CreateConnection()`**
  - **`.CreateParameter()`**

ADO.NET
Providers Factory

```
String providerName = "System.Data.SqlClient";

String connectionString = "Data Source=localhost\\SQLExpress;" +
                      "Initial Catalog=miniportal;" +
                      "User ID=user;Password=password";

DbProviderFactory dbFactory =
   DbProviderFactories.GetFactory(providerName);

DbConnection connection = dbFactory.CreateConnection();

connection.ConnectionString = connectionString;

DbCommand command = connection.CreateCommand();

command.Connection = connection;

command.CommandText = "Select * from users";
```

ADO.NET
`DbCommand`

- Representa una sentencia que se envía a una fuente de datos
  - Generalmente, pero no necesariamente SQL
- Implementada por **`SqlCommand`**, **`OleDbCommand`**, etc.
- Funcionalidad
  - Definir la sentencia a ejecutar
  - Ejecutar la sentencia
  - Enviar y recibir parámetros
  - Crear una versión compilada
- Propiedades adicionales, métodos y colecciones dependerán de la implementación específica del proveedor

ADO.NET
`DbCommand`

- Propiedades
  - **`CommandText`**
  - **`CommandTimeOut`**
  - **`CommandType`**
    - **`CommandType.Text`**
    - **`CommandType.StoredProc`**
  - **`Connection`**
  - **`Parameters`**
  - **`Transaction`**

ADO.NET
Comandos

- Si se trabaja con comandos dependientes del Data Provider es posible disponer de varios constructores
  - **SqlCommand()**
  - **SqlCommand(cmdText)**
  - **SqlCommand(cmdText, connection)**
  - **SqlCommand(cmdText, connection, transaction)**

```
SqlCommand command = new SqlCommand("SELECT loginName " +
                    "FROM UserProfile ", connection);
```

ADO.NET
Comandos

- Si se trabaja con comandos genéricos (independientes del "Data Provider"), el comando debe crearse a partir de la conexión
  - Único constructor, sin parámetros
  - Inicialización mediante acceso a propiedades

```
// Create the command and set properties ...
DbCommand command = connection.CreateCommand();
command.CommandText =
    "SELECT loginName " +
    "FROM UserProfile ";
command.Connection = connection;
command.CommandTimeout = 15;
command.CommandType = CommandType.Text;
command.Prepare();
```

ADO.NET
Comandos Parametrizados

- Comandos poseen colección **`Parameters`**
- **`DbParameter`**
  - **`ParameterName`**
  - **`DbType`**
    - Enumeración: **`String`**, **`Int32`**, **`Date`**, **`Double`**, etc.
  - **`Value`**
  - **`Size`**
  - **`IsNullable`**

ADO.NET
Comandos Parametrizados

```
DbCommand command = connection.CreateCommand();
command.CommandText =
     "SELECT loginName FROM UserProfile " +
     "WHERE loginName = @loginName";

// Create and populate parameter
DbParameter loginNameParam = command.CreateParameter();
loginNameParam.ParameterName = "@loginName";
loginNameParam.DbType = DbType.String;
loginNameParam.Value = "jsmith";

command.Parameters.Add(loginNameParam);
```

ADO.NET
Mecanismos de Acceso a Datos

- **`DbCommand`**
  - **`ExecuteReader`**
  - **`ExecuteNonQuery`**
  - **`ExecuteScalar`**

  } Entorno Conectado

- **`DbDataAdapter`**
  - **`DataSet`**

  } Entorno Desconectado

ADO.NET
Entorno Conectado. DbCommand

- **`ExecuteReader`**
  - Sentencias que devuelven múltiples filas de resultados (**`DbDataReader`**)
- **`ExecuteNonQuery`**
  - Sentencias update, delete, etc. que no devuelven ninguna fila como resultado
- **`ExecuteScalar`**
  - Sentencias SQL que devuelven una fila con un único valor como resultado

ADO.NET
Entorno Conectado. **`DbDataReader`**

- Proporciona acceso secuencial de sólo lectura a una fuente de datos
- Creado a través de **`command.ExecuteReader()`**
- Al utilizar un objeto **`DbDataReader`**, las operaciones sobre la conexión **`DbConnection`** quedan deshabilitadas hasta que se cierre el objeto **`DbDataReader`**

ADO.NET
`DbDataReader`

- Propiedades de interés:
  - **`FieldCount:`** devuelve el número de campos en la fila actual
  - **`RecordsAffected`**: número de registros afectados
- Métodos
  - **`Read()`**
    - Avanza el **`DbDataReader`** al siguiente registro
    - Inicialmente se sitúa antes del primer registro del resultado
    - Devuelve **`false`** si ha llegado al final, **`true`** en caso contrario
  - **`Close`**()
    - Cierra el objecto **`DbDataReader`**
  - **`GetValues`**()
    - Obtiene la fila actual
  - Proporciona métodos para el tipado de los datos leídos (**`GetValue`**, **`GetString`**, etc.)
    - `Boolean b = myDataReader.GetBoolean(fieldNumber);`

ADO.NET
DbDataReader

```
try {
      << ... >>
      // Create the command and set properties ...
      DbCommand command = connection.CreateCommand();
      command.CommandText = "SELECT loginName, email " +
                  "FROM UserProfile" ;
      command.Connection = connection;
      command.Prepare();
      // Open the connection and execute the command ...
      connection.Open();
      DbDataReader dr = command.ExecuteReader();
      while (dr.Read()) {
              String loginName = dr.GetString(0);
              String email = dr.GetString(1);
              Console.WriteLine("loginName: " + loginName + ",
                       email: " + email);
      }
      // Close the DataReader ...
      dr.Close();
} catch (Exception e) {
      << ... >>
} finally {
      // Ensures connection is closed
      if (connection != null) connection.Close();
}
```

ADO.NET
Entorno Conectado. ExecuteNonQuery

- **ExecuteNonQuery**

```
<< ... >>

      // Create the command and set properties ...
      DbCommand command = connection.CreateCommand();
      command.CommandText = "UPDATE Account " +
                  "SET balance = 1.1 * balance ";
      command.Connection = connection;
      command.Prepare();

      // Open the connection and execute the command ...
      connection.Open();

      // Executes an SQL statement against the Connection object
      // of a .NET Framework data provider, and returns the
      // number of rows affected.
      int affectedRows = command.ExecuteNonQuery();
      Console.WriteLine("affectedRows: " + affectedRows);

<< ... >>
```

ADO.NET

Entorno Conectado. **`ExecuteScalar`**

- **`ExecuteScalar`**

```
<< ... >>

     // Create the command and set properties ...
     DbCommand command = connection.CreateCommand();
     command.CommandText = "SELECT count(*) " +
                           "FROM UserProfile ";
     command.Connection = connection;
     command.Prepare();

     // Open the connection and execute the command ...
     connection.Open();

     // Executes the query, and returns the first column of the
     // first row in the resultset returned by the query. Extra
     // columns or rows are ignored.
     Int32 numberOfUsers = (Int32) command.ExecuteScalar();
     Console.WriteLine("numberOfUsers: " + numberOfUsers);

<< ... >>
```

ADO.NET

Transacciones

- Transacción:
  - Conjunto sentencias SQL que constituyen una unidad lógica de trabajo
- Debe ajustarse a las propiedades ACID:
  - **A**tomicity:
    - Las sentencias se ejecutan todas o ninguna
  - **C**onsistency:
    - Una vez finalizada, los datos deben ser consistentes
  - **I**solation:
    - Transacciones se comporten como si cada una fuera la única transacción
  - **D**urability:
    - Una vez finalizada, los cambios son permanentes

ADO.NET
Transacciones

- Se crean a partir de la conexión
  - `connection.BeginTransaction();`
- Es obligatorio asociar los comandos a la transacción
  - Propiedad `command.Transaction`
- Métodos
  - `Commit();`
  - `Rollback();`

ADO.NET
Transacciones

- Niveles de Aislamiento
  - `IsolationLevel.ReadUncommitted`: pueden ocurrir “dirty reads”, “non-repeatable reads” y “phantom reads”
  - `IsolationLevel.ReadCommitted`: pueden ocurrir “non-repeatable reads” y “phantom reads
  - `IsolationLevel.RepeatableRead`: pueden ocurrir “phantom reads"
  - `IsolationLevel.Serializable`: elimina todos los problemas de concurrencia
- El nivel de aislamiento se fija en el momento de crear la transacción
  -
```
connection.BeginTransaction(
    IsolationLevel.Serializable);
```

ADO.NET
Transacciones

```
try {

   // Get the connection ...

   << ... >>

   // Open the connection ...

   connection.Open();

   // Starts a new transaction ...
   // transaction = connection.BeginTransaction(); //default
   transaction = connection.BeginTransaction(IsolationLevel.Serializable);

   // Create the command and set properties ...
   DbCommand selectCommand = connection.CreateCommand();
   selectCommand.Connection = connection;
   selectCommand.CommandText =
         "SELECT balance " +
         "FROM ACCOUNT " +
         "WHERE accId = 1";

   // Associate the command with the transaction
   selectCommand.Transaction = transaction;
   selectCommand.Prepare();
```

ADO.NET
Transacciones

```
   // Execute the selectCommand ...
   dataReader = selectCommand.ExecuteReader();

   if (!dataReader.Read()) {
         throw new Exception("Error reading from Data Base");
   }

   balance = dataReader.GetDouble(0);
   Console.WriteLine("Current balance: " + balance);
   balance = 1.1 * balance; Console.WriteLine("New balance must be: " +
   balance);

   // PAUSE: another process could try to change the balance value in BD
   Console.ReadLine();

   // The DataReader must be closed before executing updateCommand!!!
   dataReader.Close();

   // Create the updateCommand and set properties ...
   DbCommand updateCommand = connection.CreateCommand();
   updateCommand.Connection = connection;
   updateCommand.CommandText =
         "UPDATE ACCOUNT " +
         "SET balance = @balance " +
         "WHERE (accId = 1)";
```

ADO.NET
Transacciones

```
   updateCommand.Transaction = transaction;
   updateCommand.Prepare();

   DbParameter balanceParam = updateCommand.CreateParameter();
   balanceParam.ParameterName = "@balance";
   balanceParam.DbType = DbType.Decimal;
   balanceParam.Value = balance;
   updateCommand.Parameters.Add(balanceParam);

   // Execute the updateCommand ...
   int afectedRows = updateCommand.ExecuteNonQuery();
   transaction.Commit();
   commited = true;
   Console.WriteLine("Transaction COMMITED");

   } catch (DbException e) {

        Console.WriteLine(e.StackTrace);
        Console.WriteLine(e.Message);

   } catch (Exception e) {

        Console.WriteLine(e.StackTrace);
        Console.WriteLine(e.Message);
```

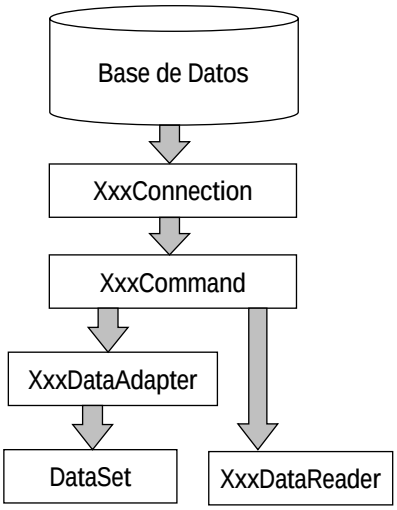
ADO.NET
Transacciones

```
    } finally {

        if (!commited) {
            if (transaction != null) {
                transaction.Rollback();
            }
        }

        // Ensures connection is closed
        if (connection.State.Equals(ConnectionState.Open)) {
            connection.Close();
        }
    }
}
```

ADO.NET
Excepciones

- **`System.Data.Common.DbException`**
  - Se lanza cuando ocurre algún problema en la capa de acceso a datos
  - Es una clase abstracta que implementa **`ExternalException`**
  - Cada "Data Provider" proporcionará una implementación específica
- Constructores:
  - **`DbException()`**
  - **`DbException(string message)`**
    - **`message`**: mensaje a mostrar
  - **`DbException(string message, Exception innerException)`**
    - **`innerException`**: la referencia de la excepción interna
  - **`DbException(string message, int errorCode)`**
    - **`errorCode`**: código de error para la excepción

ADO.NET
Entorno Desconectado



- **XxxConnection:** maneja la conexión a una BD
- **XxxCommand**: ejecuta comandos contra una BD
- **XxxDataAdapter**: intercambia datos entre un DataSet y una BD
- **DataSet**: copia local de datos relacionales (Entorno Desconectado)
- **XxxDataReader**: Proporciona acceso a datos Read-only, Forward-only (Entorno Conectado)

ADO.NET
Entorno Desconectado: **`DataSet`**

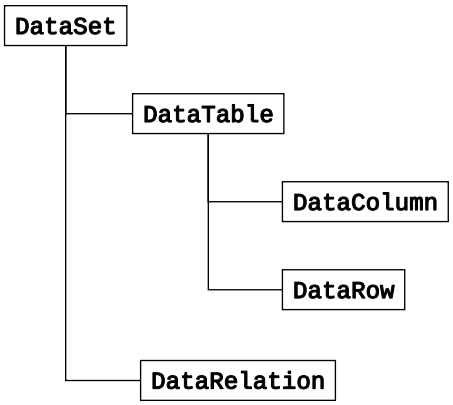
- Núcleo ADO.NET bajo entorno desconectado
- Representación en memoria del contenido de la base de datos
- Operaciones sobre los datos se realizan sobre el **`DataSet`**, no sobre el origen de datos
- Almacena
  - Tablas (**`DataTable`**)
  - Relaciones ente tablas (**`DataRelation`**)
- Independiente del proveedor de Datos
- Problemas
  - Sincronización datos
  - Acceso Concurrente

ADO.NET
`DataTable`

- Representación lógica de una tabla de la base de datos
- Propiedades de interés:
  - **`Columns:`**
    - Colección de tipo **`ColumnsCollection`** de objetos **`DataColumn`**
  - **`Rows`**:
    - Colección de tipo **`RowsCollection`** de objectos **`DataRow`**
  - **`ParentRelations`**:
    - **`RelationsCollection`**. Relaciones en las que participa la tabla
  - **`Constraints`**:
    - Returns the table's **`ConstraintsCollection`**
  - **`DataSet`**:
    - **`DataSet`** en el que está incluida la **`DataTable`**
  - **`PrimaryKey`**:
    - **`DataColumn`** que actúa como clave primaria de la tabla

ADO.NET

Entorno Desconectado: **`DataSet`**



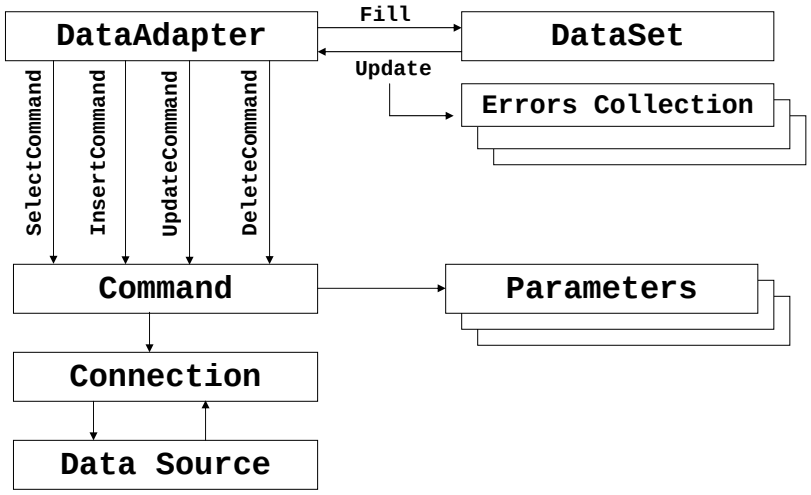
ADO.NET

Entorno Desconectado. **`SqlDataAdapter`**

- Bridge entre origen de datos y **`DataSet`**
- Implementa los métodos abstractos de la clase **`DataAdapter`**:
  - **`public abstract int Fill( DataSet dataSet );`**
  - **`public abstract int Update( DataSet dataSet );`**
- Propiedades de interés:
  - **`DeleteCommand:`** El comando de borrado, expresado en SQL
  - **`InsertCommand`**: Obtiene o establece el comando de inserción
  - **`SelectCommand`**: Obtiene o establece el comando de selección
  - **`UpdateCommand`**: Obtiene o establece el comando de actualización
  - **`TableMappings`**: Relaciona la tabla con el **`DataTable`**
- Debe especificarse siempre un comando de selección

ADO.NET

Entorno Desconectado



ADO.NET

Entorno Desconectado

```
// Create the DataAdapter
SqlDataAdapter sqlDataAdapter = new SqlDataAdapter();

// Establish Select Command (used to populate the DataSet)
SqlCommand selectCmd =
  new SqlCommand("Select * from UserProfile", connection);

sqlDataAdapter.SelectCommand = selectCmd;

// Create the DataSet
DataSet sqlDataSet = new DataSet("UserProfile");

// Fill DataSet
sqlDataAdapter.Fill(sqlDataSet, "UserProfile");

connection.Close();
```

ADO.NET

Entorno Desconectado

```
// Access Data

// First table in the data set
DataTable dataTable = sqlDataSet.Tables["UserProfile"];

// First row in the table
DataRow dr = dataTable.Rows[0]

Console.WriteLine( dr["loginName"] );

// Access Email Column
DataColumn emails = dataTable.Columns["email"];
```