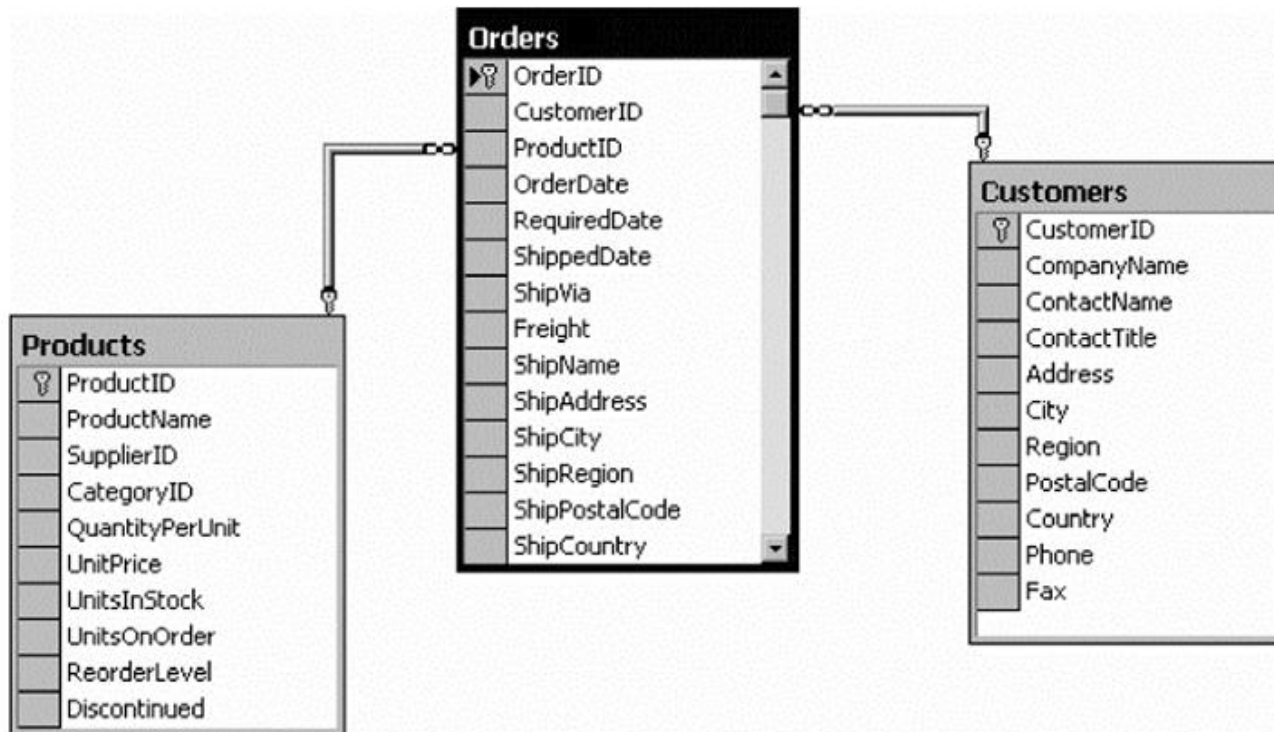


ADO .NET Fundamentals

Understanding Databases

- Database for a sales program consists of a list of customers, a list of products, and a list of sales



Understanding Databases

- Web application needs a full relational database management system (RDBMS), such as SQL Server
- Typically databases are used in applications like,
 - E-commerce sites (like Amazon)
 - Search engines (like Google)
 - Knowledge bases (like Microsoft Support)
 - Media sites (like The New York Times)

Configuring Your Database

- SQL Server Express: scaled-down version of SQL Server
- View ► Server Explorer
- Using the Data Connections node in the Server Explorer, you can connect to existing databases or create new ones.

The sqlcmd Command-Line Tool

- Command-line tool named - `sqlcmd.exe`
- Often, sqlcmd is used in a batch file—for example, to create database tables as part of an automated setup process.
- Open Visual Studio Command Line tool
 - Enter: `sqlcmd -?`

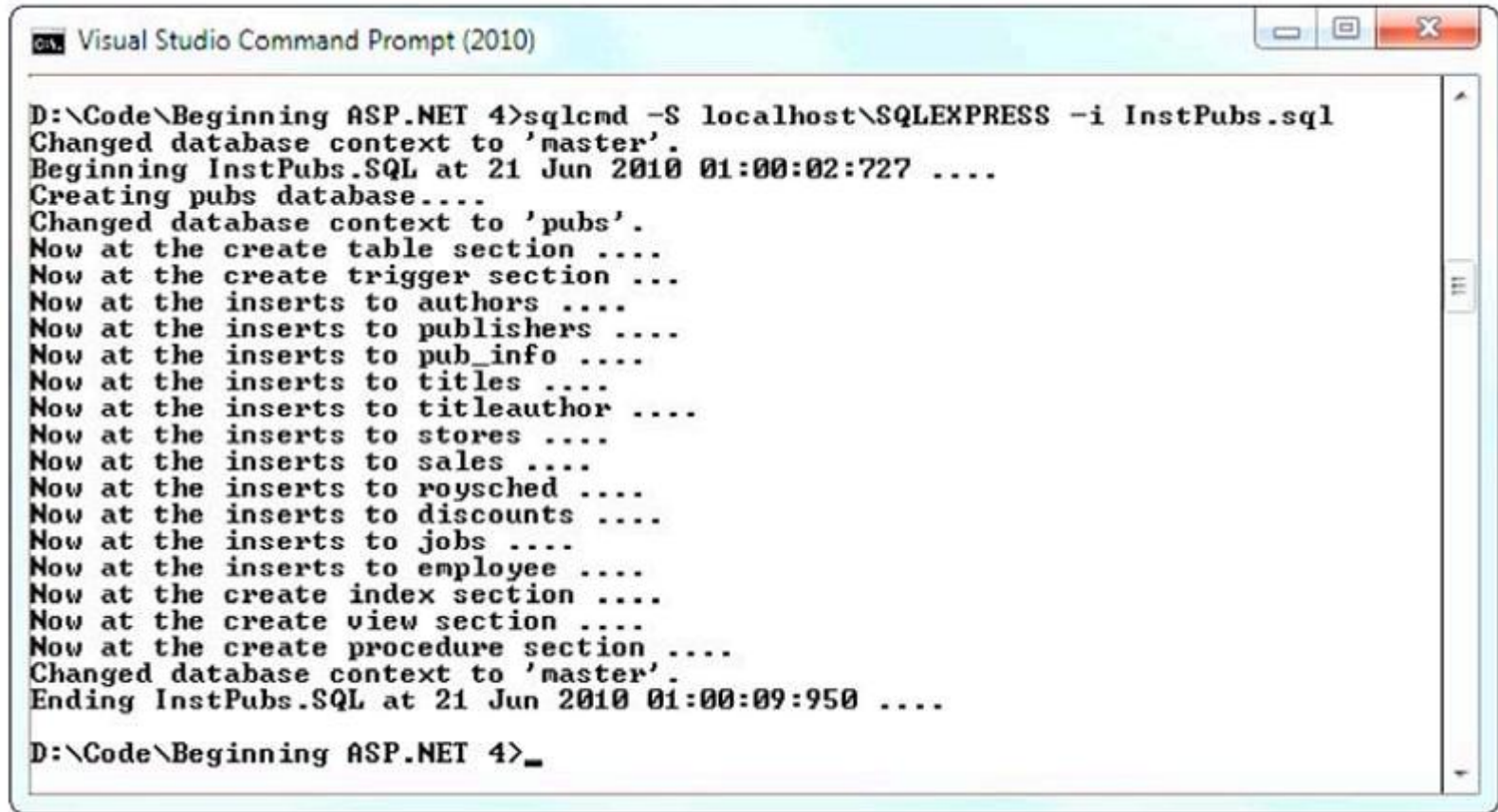
The sqlcmd Command-Line Tool

- `—S` - specifies the location of your database server
- `—i` - supplies a script file with SQL commands that you want to run
- Example:

```
sqlcmd -S localhost\SQLEXPRESS -i InstPubs.sql
```

```
sqlcmd -i InstPubs.sql
```

The sqlcmd Command-Line Tool



```
Visual Studio Command Prompt (2010)

D:\Code\Beginning ASP.NET 4>sqlcmd -S localhost\SQLEXPRESS -i InstPubs.sql
Changed database context to 'master'.
Beginning InstPubs.SQL at 21 Jun 2010 01:00:02:727 ....
Creating pubs database....
Changed database context to 'pubs'.
Now at the create table section ....
Now at the create trigger section ...
Now at the inserts to authors ....
Now at the inserts to publishers ....
Now at the inserts to pub_info ....
Now at the inserts to titles ....
Now at the inserts to titleauthor ....
Now at the inserts to stores ....
Now at the inserts to sales ....
Now at the inserts to roysched ....
Now at the inserts to discounts ....
Now at the inserts to jobs ....
Now at the inserts to employee ....
Now at the create index section ....
Now at the create view section ....
Now at the create procedure section ....
Changed database context to 'master'.
Ending InstPubs.SQL at 21 Jun 2010 01:00:09:950 ....

D:\Code\Beginning ASP.NET 4>_
```

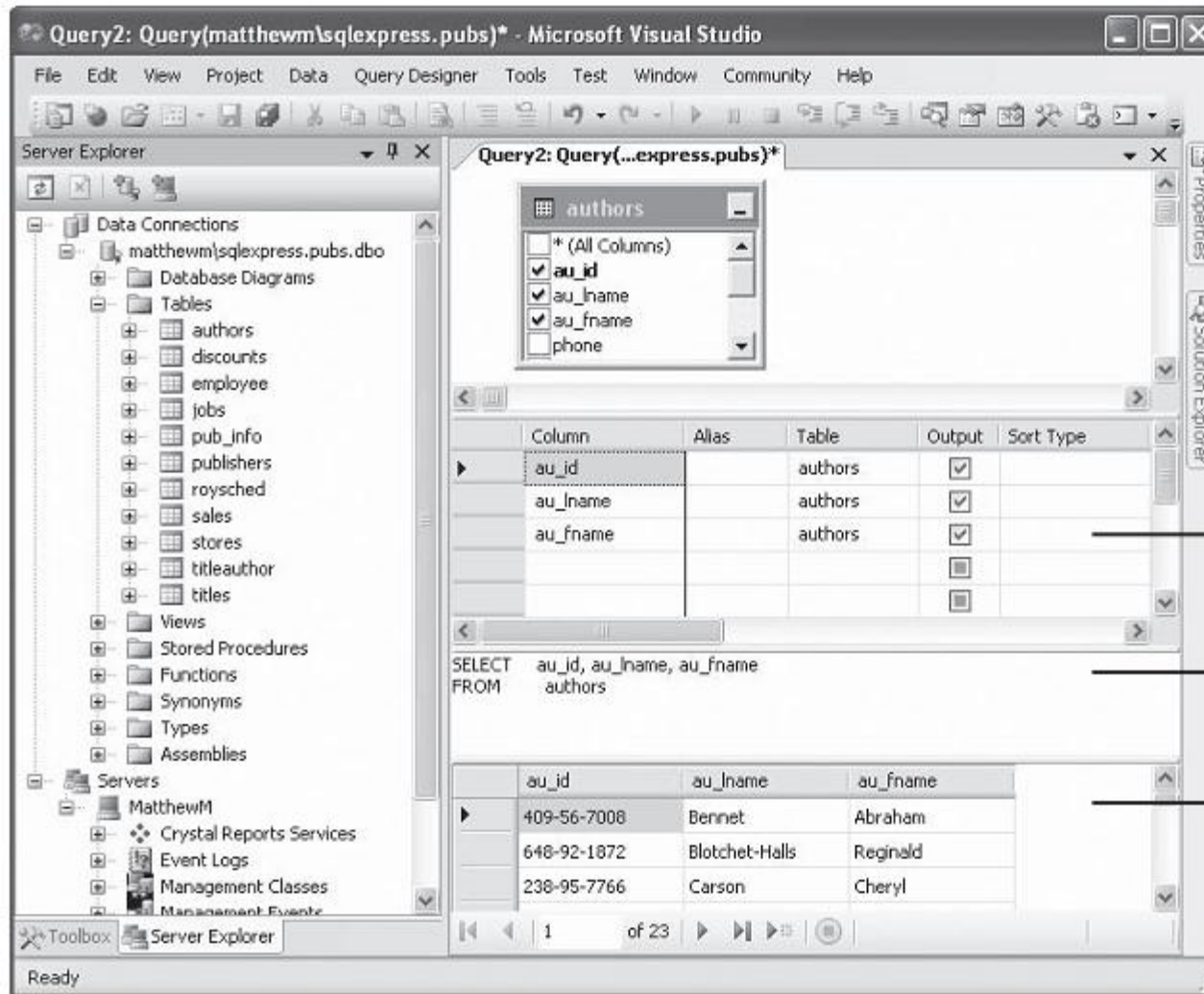
SQL Basics

- SQL (Structured Query Language)
- select, add, and modify data
- SQL scripts for stored procedures and triggers
- SQL includes: Select, Update, Insert and Delete

Running Queries in Visual Studio

- Use
 - SQL Server Management Studio
 - Server Explorer
- Demo
 - Right-click your connection, and choose New Query
 - query-building window
 - Query Designer ➤ Execute SQL

Running Queries in Visual Studio



Configure
the query
graphically

Edit the SQL
by hand

Query Results

The Select Statement

- Select Statement

```
SELECT [columns]
FROM [tables]
WHERE [search_condition]
ORDER BY [order_expression ASC | DESC]
```

- Update, Insert and Delete with examples

The Data Provider Model

- ADO.NET relies on the functionality in small set of core classes:
 - To contain and manage data (such as DataSet, DataTable, DataRow, and DataRelation)
 - To connect to a specific data source (such as Connection, Command, and DataReader)
- DataSet: Its data container, customized for relational data.
- Data providers are customized so that each one uses the best-performing way of interacting with its data source.
 - Ex: SQL Data Provider, Oracle data provider.
- Sql server – Tabular data stream protocol

The Data Provider Model

- **SqlConnection** and **SqlCommand** classes
- **OracleConnection** and **OracleCommand**

Namespace	Purpose
System.Data.SqlClient	Contains the classes you use to connect to a Microsoft SQL Server database and execute commands (like SqlConnection and SqlCommand).
System.Data.SqlTypes	Contains structures for SQL Server–specific data types such as SqlMoney and SqlDateTime. You can use these types to work with SQL Server data types without needing to convert them into the standard .NET equivalents (such as System.Decimal and System.DateTime). These types aren't required, but they do allow you to avoid any potential rounding or conversion problems that could adversely affect data.
System.Data	Contains fundamental classes with the core ADO.NET functionality. This includes DataSet and DataRelation, which allow you to manipulate structured relational data. These classes are totally independent of any specific type of database or the way you connect to it.

Direct Data Access

- Direct data access and Disconnected data access
- Direct Data Access:
 - Don't need to keep a copy of their data in memory for long periods of time
- Import ADO.Net Namespaces
 - Imports System.Data
 - Imports System.Data.SqlClient

Direct Data Access

To query information with simple data access, follow these steps:

1. Create Connection, Command, and DataReader objects.
2. Use the DataReader to retrieve information from the database, and display it in a control on a web form.
3. Close your connection.
4. Send the page to the user. At this point, the information your user sees and the information in the database no longer have any connection, and all the ADO.NET objects have been destroyed.

To add or update information, follow these steps:

1. Create new Connection and Command objects.
2. Execute the Command (with the appropriate SQL statement).

Direct Data Access

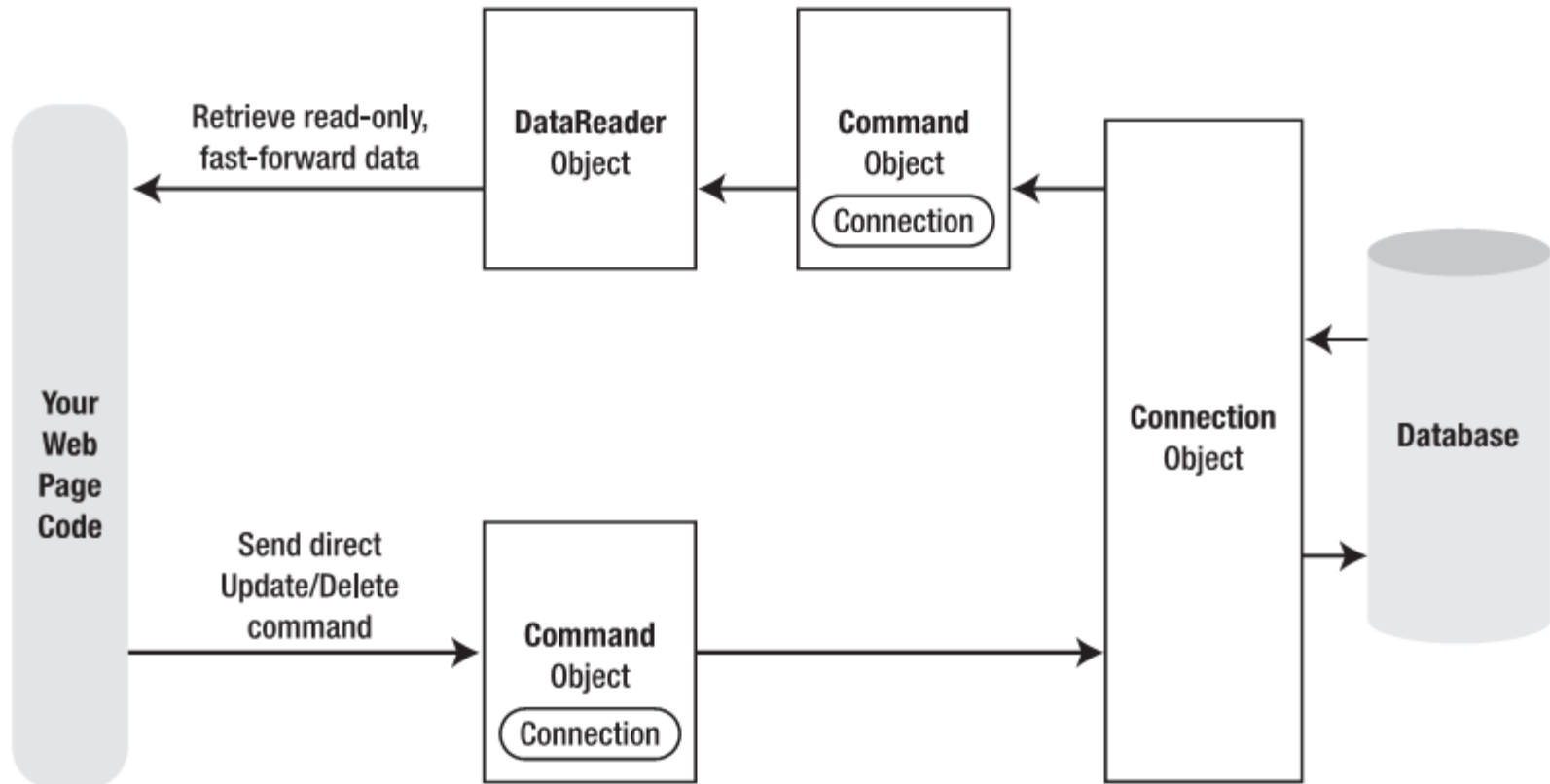


Fig: Direct data access with ADO.NET

Creating a Connection

- Create connection to the data source
- Connections are fixed – keep it short
- Write database code inside a Try/Catch error handling structure
- To create connection, use `ConnectionString` property
- `ConnectionString`: data source, log in, and choose an initial database.

Creating a Connection

- Example:

```
SqlConnection myConnection = new SqlConnection();  
myConnection.ConnectionString = "Data Source=localhost;" +  
    "Initial Catalog=Pubs;Integrated Security=SSPI";
```

```
SqlConnection myConnection = new SqlConnection();  
myConnection.ConnectionString = @"Data Source=(localdb)\v11.0;" +  
    "Initial Catalog=Pubs;Integrated Security=SSPI";
```

The Connection String

- Series of distinct pieces of information separated by semicolons (;).
 - ***Data source***: name of the server where the data source is located, “(localdb)\v11.0”
 - ***Initial catalog***: Accessing database, change with `Connection.ChangeDatabase()`
 - ***Integrated security***: Windows user account, SSPI (Security Support Provider Interface)
 - ***ConnectionTimeout***: How long your code will wait, in seconds, before generating an error if it cannot establish a database connection

Windows Authentication

- *Integrated Windows authentication*
 - SQL Server automatically uses the Windows account information for the currently logged-in process.
- *SQL Server authentication*
 - SQL Server uses its own user account information in the database.

Storing the Connection String

- Create a Connection object and supply the connection string

```
SqlConnection myConnection = new SqlConnection(connectionString);  
// myConnection.ConnectionString is now set to connectionString.
```

- <connectionStrings> section of the web.config file

```
<configuration>  
  <connectionStrings>  
    <add name="Pubs" connectionString=  
"Data Source=localhost;Initial Catalog=Pubs;Integrated Security=SSPI"/>  
  </connectionStrings>  
  ...  
</configuration>
```

Storing the Connection String

- Retrieve your connection string by name
- Import the System.Web.Configuration namespace

```
string connectionString =  
    WebConfigurationManager.ConnectionStrings["Pubs"].ConnectionString;
```

Making the Connection

```
// Define the ADO.NET Connection object.
string connectionString =
    WebConfigurationManager.ConnectionStrings["Pubs"].ConnectionString;
SqlConnection myConnection = new SqlConnection(connectionString);

try
{
    // Try to open the connection.
    myConnection.Open();
    lblInfo.Text = "<b>Server Version:</b> " + myConnection.ServerVersion;
    lblInfo.Text += "<br /><b>Connection Is:</b> " +
        myConnection.State.ToString();
}
catch (Exception err)
{
    // Handle an error by displaying the information.
    lblInfo.Text = "Error reading the database. ";
    lblInfo.Text += err.Message;
}
finally
{
    // Either way, make sure the connection is properly closed.
    // (Even if the connection wasn't opened successfully,
    // calling Close() won't cause an error.)
    myConnection.Close();
    lblInfo.Text += "<br /><b>Now Connection Is:</b> ";
    lblInfo.Text += myConnection.State.ToString();
}
```

Making the Connection

Another Approach

```
// Define the ADO.NET Connection object.
string connectionString =
    WebConfigurationManager.ConnectionStrings["Pubs"].ConnectionString;
SqlConnection myConnection = new SqlConnection(connectionString);

try
{
    using (myConnection)
    {
        // Try to open the connection.
        myConnection.Open();
        lblInfo.Text = "<b>Server Version:</b> " + myConnection.ServerVersion;
        lblInfo.Text += "<br /><b>Connection Is:</b> " +
            myConnection.State.ToString();
    }
}
catch (Exception err)
{
    // Handle an error by displaying the information.
    lblInfo.Text = "Error reading the database. ";
    lblInfo.Text += err.Message;
}

lblInfo.Text += "<br /><b>Now Connection Is:</b> ";
lblInfo.Text += myConnection.State.ToString();
```


The Select Command

- To retrieve data,
 - A SQL statement that selects the information you want
 - A Command object that executes the SQL statement
 - A DataReader or DataSet object to access the retrieved records

```
SqlCommand myCommand = new SqlCommand();  
myCommand.Connection = myConnection;  
myCommand.CommandText = "SELECT * FROM Authors ORDER BY au_lname ";  
SqlCommand myCommand = new SqlCommand(  
    "SELECT * FROM Authors ORDER BY au_lname ", myConnection);
```

The DataReader

- Allows you to quickly retrieve all your results
- Uses a live connection and should be used quickly and then closed

```
myConnection.Open();  
  
// You don't need the new keyword, as the Command will create the DataReader.  
SqlDataReader myReader;  
myReader = myCommand.ExecuteReader();  
  
myReader.Read();    // The first row in the result set is now available.  
lstNames.Items.Add(myReader["au_lname"] + ", " + myReader["au_fname"]);  
  
myReader.Close();  
myConnection.Close();
```

Creating More Robust Commands

- Two potentially serious drawbacks
 - Users may Accidentally enter characters that will affect your SQL statement - value contains an apostrophe (')
 - Users might deliberately enter characters that will affect your SQL statement
- Solution

```
string authorID = txtID.Text.Replace("'", "");
```

Creating More Robust Commands

- Use parameterized command
- A parameterized command is one that replaces hard-coded values with placeholders.

```
SELECT * FROM Customers WHERE CustomerID = 'ALFKI'
```



```
SELECT * FROM Customers WHERE CustomerID = @CustomerID
```

Creating More Robust Commands

```
protected void cmdInsert_Click(Object sender, EventArgs e)
{
    // Perform user-defined checks.
    if (txtID.Text == "" || txtFirstName.Text == "" || txtLastName.Text == "")
    {
        lblStatus.Text = "Records require an ID, first name, and last name.";
        return;
    }

    // Define ADO.NET objects.
    string insertSQL;
    insertSQL = "INSERT INTO Authors (";
    insertSQL += "au_id, au_fname, au_lname, ";
    insertSQL += "phone, address, city, state, zip, contract) ";
    insertSQL += "VALUES (";
    insertSQL += "@au_id, @au_fname, @au_lname, ";
    insertSQL += "@phone, @address, @city, @state, @zip, @contract)";

    SqlConnection con = new SqlConnection(connectionString);
    SqlCommand cmd = new SqlCommand(insertSQL, con);

    // Add the parameters.
    cmd.Parameters.AddWithValue("@au_id", txtID.Text);
    cmd.Parameters.AddWithValue("@au_fname", txtFirstName.Text);
    cmd.Parameters.AddWithValue("@au_lname", txtLastName.Text);
}
```

```

cmd.Parameters.AddWithValue("@phone", txtPhone.Text);
cmd.Parameters.AddWithValue("@address", txtAddress.Text);
cmd.Parameters.AddWithValue("@city", txtCity.Text);
cmd.Parameters.AddWithValue("@state", txtState.Text);
cmd.Parameters.AddWithValue("@zip", txtZip.Text);
cmd.Parameters.AddWithValue("@contract", chkContract.Checked);

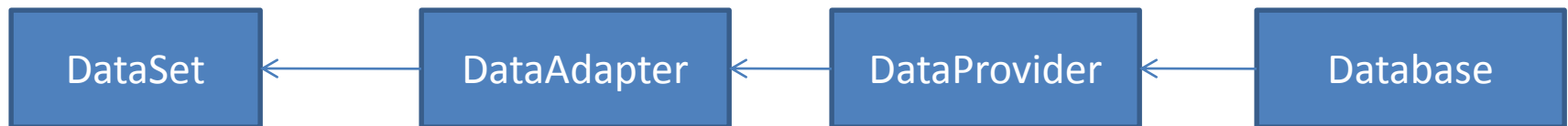
// Try to open the database and execute the update.
int added = 0;
try
{
    con.Open();
    added = cmd.ExecuteNonQuery();
    lblStatus.Text = added.ToString() + " record inserted.";
}
catch (Exception err)
{
    lblStatus.Text = "Error inserting record. ";
    lblStatus.Text += err.Message;
}
finally
{
    con.Close();
}

// If the insert succeeded, refresh the author list.
if (added > 0)
{
    FillAuthorList();
}
}

```

Disconnected Data Access

- Datasets store a copy of data from the database tables.
- Datasets can not directly retrieve data from Databases.
- DataAdapters are used to link Databases with DataSets.
- If we see diagrammatically,

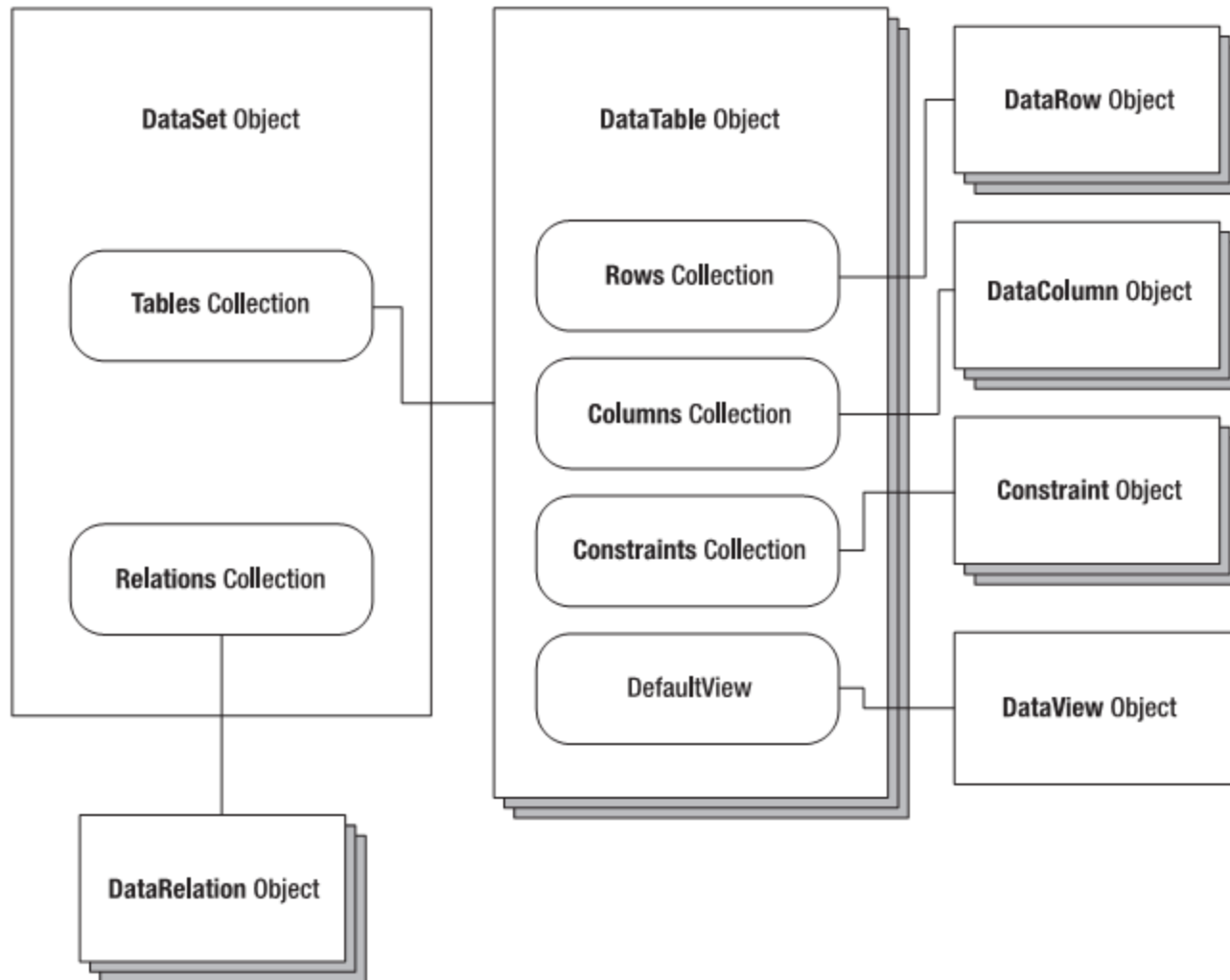


Disconnected Data Access

Some Reasons for Using Disconnected Data Access

- Time-consuming with the data.
- To use ASP.NET data binding to fill a web control
- Navigate backward and forward through your data
- Navigate from one table to another.
- Save the data to a file for later use

Selecting Disconnected Data



Selecting Disconnected Data

```
private void FillAuthorList()
{
    lstAuthor.Items.Clear();

    // Define ADO.NET objects.
    string selectSQL;
    selectSQL = "SELECT au_lname, au_fname, au_id FROM Authors";
    SqlConnection con = new SqlConnection(connectionString);
    SqlCommand cmd = new SqlCommand(selectSQL, con);
    SqlDataAdapter adapter = new SqlDataAdapter(cmd);
    DataSet dsPubs = new DataSet();

    // Try to open database and read information.
    try
    {
        con.Open();

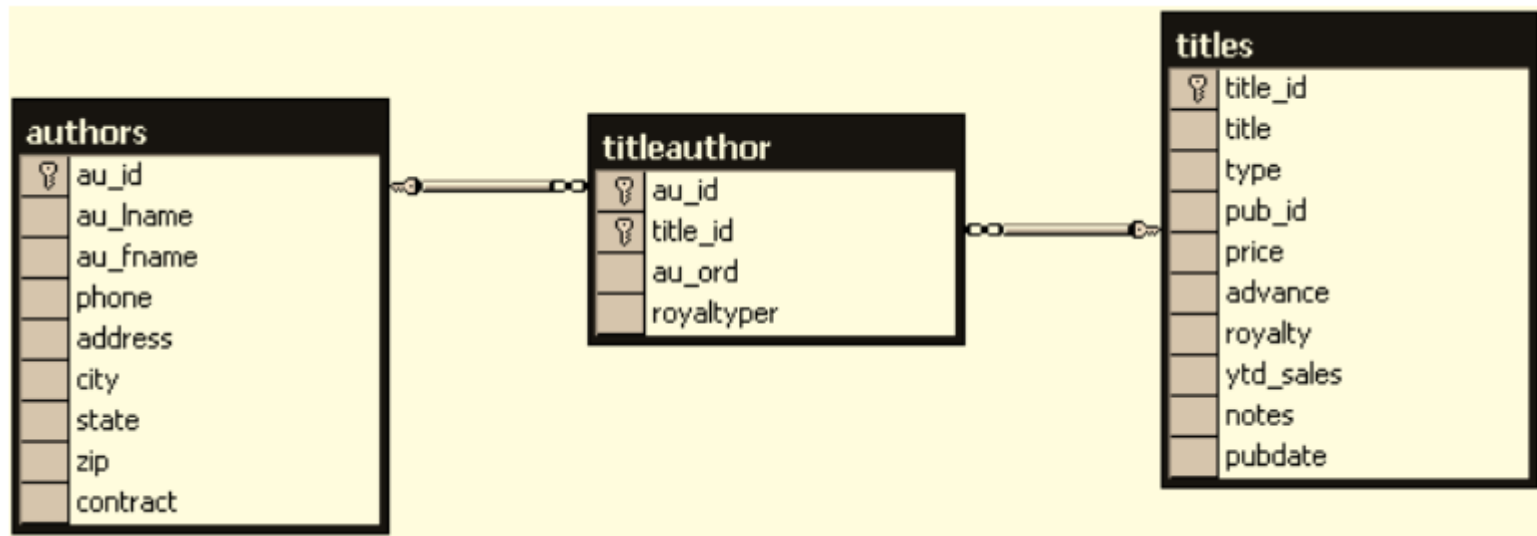
        // All the information is transferred with one command.
        // This command creates a new DataTable (named Authors)
        // inside the DataSet.
        adapter.Fill(dsPubs, "Authors");
    }
}
```

Selecting Disconnected Data

```
catch (Exception err)
{
    lblStatus.Text = "Error reading list of names. ";
    lblStatus.Text += err.Message;
}
finally
{
    con.Close();
}

foreach (DataRow row in dsPubs.Tables["Authors"].Rows)
{
    ListItem newItem = new ListItem();
    newItem.Text = row["au_lname"] + ", " +
        row["au_fname"];
    newItem.Value = row["au_id"].ToString();
    lstAuthor.Items.Add(newItem);
}
}
```

Selecting Multiple Tables



Selecting Multiple Tables

```
// Define the ADO.NET objects.
string connectionString =
    WebConfigurationManager.ConnectionStrings["Pubs"].ConnectionString;
SqlConnection con = new SqlConnection(connectionString);

string selectSQL = "SELECT au_lname, au_fname, au_id FROM Authors";
SqlCommand cmd = new SqlCommand(selectSQL, con);
SqlDataAdapter adapter = new SqlDataAdapter(cmd);
DataSet dsPubs = new DataSet();
try
{
    con.Open();
    adapter.Fill(dsPubs, "Authors");

    // This command is still linked to the data adapter.
    cmd.CommandText = "SELECT au_id, title_id FROM TitleAuthor";
    adapter.Fill(dsPubs, "TitleAuthor");

    // This command is still linked to the data adapter.
    cmd.CommandText = "SELECT title_id, title FROM Titles";
    adapter.Fill(dsPubs, "Titles");
}
catch (Exception err)
{
    lblList.Text = "Error reading list of names. ";
    lblList.Text += err.Message;
}
finally
{
    con.Close();
}
```

END OF LECTURE