Lab 9 – Manual Database Basics

Contents

[1 Introduction 1](#_Toc162016939)

[2 Database Primer 2](#_Toc162016940)

[3 Query Builder in Access 4](#_Toc162016941)

[4 Databinding: DataSource & GridView 10](#_Toc162016942)

[5 Explore Connection String and DataSource Structure 15](#_Toc162016943)

[6 Manual Database – Select Statement 16](#_Toc162016944)

[7 Manual Database – Insert Statement 21](#_Toc162016945)

[8 Manual Database – Update & Delete Statements 24](#_Toc162016946)

[9 Submission 27](#_Toc162016947)

# Introduction

This lab assumes you have completed Lab 8 successfully. The objectives of this lab are:

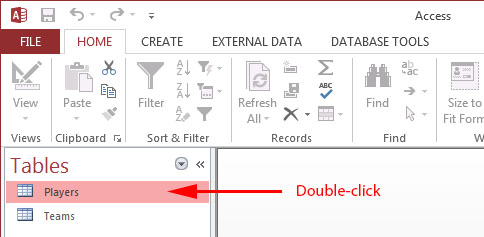
1. Be able to describe what these are: database, table, row/record, column/field, AutoNumber.
2. Configure a *DataSource* object and bind it to a *GridView.*
3. Use *Connection, Command,* and *DataReader* objects to select data from a database manually
4. Use *Connection* and *Command* objects to insert, update, and delete data from a database manually

To make this document easier to read, it is recommended that you turn off spell checking and grammar checking in Word: (a) choose: File, Option, Proofing, (b) at the very bottom, check: “Hide spelling errors…” and “Hide grammar errors…”

Note:

* The code you will copy/paste in some cases spans several pages. Make sure you get it all.
* Study the code. Get the big picture (see Lab objectives above). We will cover the details in class.
* (this note is for me) This Lab could be improved by incorporating: Inner Join, parameterized queries, filtering

# Database Primer

1. Download *players.zip* from the Schedule, unzip anywhere and you will find *players.mdb,* the *Players* database in MS Access format.
2. Do the following:
3. Double-click *players.mdb* to open.
4. As shown on the right, double-click the *Players* table to open it.
5. **(Read, no action required)** A database is a set of (usually) related *tables* that store information about something of interest. For example, below we have the *Players Table* as shown below and what is open in Access. Note the following:
6. A database *table* is organized as *rows* and *columns*. Each *row* (*record*) represents data about a player. Each *column* (*field)* is an attribute about a player. For instance, we see fields: *LName* and *FName* the represent the last name and first name of a player. The *PNumber* field is poorly named. It stands for player number, but would be clearer with the name *JerseyNum*.
7. Every database table we consider will have a *primary key* field. A primary key uniquely identifies a row. The value of a primary key is created by the database itself when we insert a new row. The primary key cannot be changed. The *foreign key* will be discussed later.

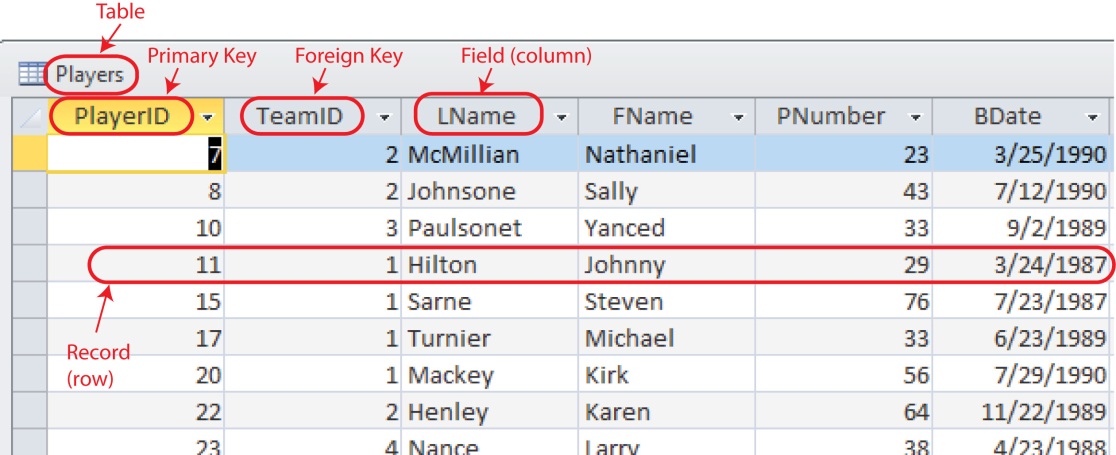


Figure 1 – Players Table

1. We can obtain more information on the fields, such as their data types. Do the following:

|  |  |
| --- | --- |
| 1. Choose: View, Design View   E:\Data-Classes\CS 3340 - Web Programming\Topics\04_Database1\Labs\b2.jpg | 1. As shown below, the primary key, *PlayerID* is of type *AutoNumber*. This datatype means that Access will automatically assign a unique value. We will discuss this in class. Select some of the other fields and see their *Field Properties*.   E:\Data-Classes\CS 3340 - Web Programming\Topics\04_Database1\Labs\b3.jpg |

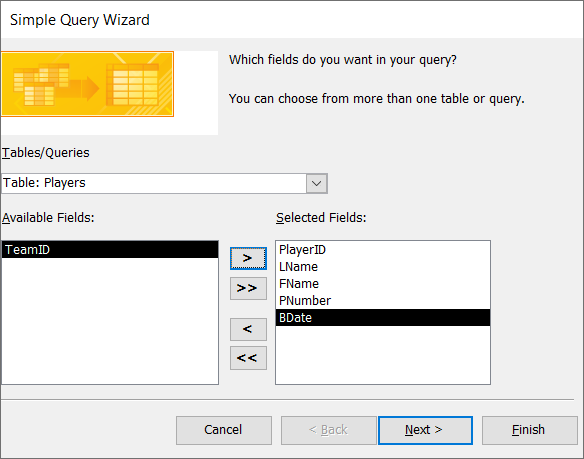
1. Choose: View, Datasheet View to return to displaying the data.
2. We will add a row (Player) to the *Players* table to see how the *PlayerID* field is “automatically” (*e.g. AutoNumber*) filled. Do the following:

|  |  |
| --- | --- |
| 1. Scroll to the bottom of *Players* table, click in the *TeamID* field, type the value: 2.   E:\Data-Classes\CS 3340 - Web Programming\Topics\04_Database1\Labs\b6.jpg | 1. Note that the *PlayerID* value is automatically supplied (your value will be different than that shown below). Add values for the rest of the fields.   E:\Data-Classes\CS 3340 - Web Programming\Topics\04_Database1\Labs\b7.jpg |
|  |  |

1. Pick any row, and change some field values in a row. Try to supply bad data. For example, type “a” for *PNumber*. Notice that the value for the key, *PlayerID* has gaps. For instance, in the figure above on the right, 121 is missing. This simply means that a record with key 121 was deleted. The database itself keeps up with the keys and recycles them when necessary.
2. Delete a row.
3. A convenient way to show all the tables and fields in a database is to display an *entity relationship* (*ER*) *diagram*. As you will see, and ER diagram is very similar to a class diagram.

|  |  |
| --- | --- |
| 1. Choose: Database Tools, Relationships   E:\Data-Classes\CS 3340 - Web Programming\Topics\04_Database1\Labs\b4.jpg | 1. You will see a diagram of the tables in this database. You will probably need to stretch them to make the look as shown below. We can see that there is a 1-many relationship between the *Teams* and *Players* tables. Thus, each team has many players, and each player has exactly one team. For homework and tests I will give you a diagram like this so that you will know how the database is organized.   E:\Data-Classes\CS 3340 - Web Programming\Topics\04_Database1\Labs\b5.jpg |

# Query Builder in Access

1. **(Read, no action required)** *Filtering* data refers to the selection of a portion of a larger collection of data according to some criteria. To filter a database table(s), we build and execute a *SQL statement* (*query*). *SQL* is the language we use to communicate with a database. We will learn about this in class. Here, we will build a query using *query builder* in Access. Later in this lab, we will write a query in the codebehind for a page so that we can display the returned data.
2. Build initial query. Do the following:
3. Choose: Create, Simple Query Wizard, OK
4. See the figure on the right. Select the *Players* table. Then select all the fields except *TeamID* and move them to the *Selected Fields*.
5. Choose: Next, Next.
6. Note the name of the query: *Player’s Query* (probably). This can be changed, of course, but we will leave it.
7. Choose: Finish. All the players will be displayed. The *TeamID* field is not displayed.
8. Choose: View, SQL View. The SQL statement that generated the data is shown.

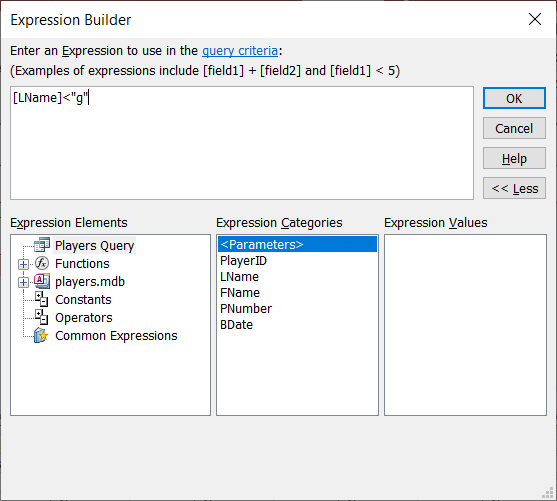
SELECT Players.[PlayerID], Players.[LName], Players.[FName], Players.[PNumber], Players.[BDate]

FROM Players;

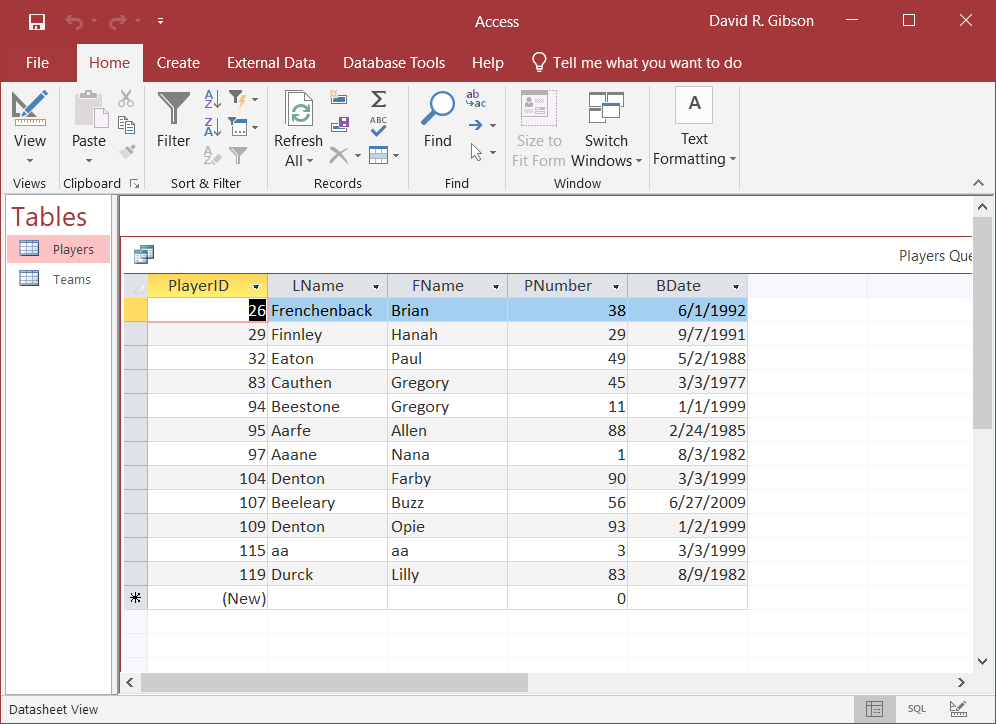
1. Modify query to filter on last names that begin with the letters a-f. Do the following:
2. Choose: View, Design View. Then, place your cursor in the *Criteria* row for the *LName* column, right-click, and choose: Build (see figure below).



1. Enter this into text area as shown below: [LName]<"g". Then, press OK.



1. Choose: Design, Run (the red exclamation point icon in tool bar). The data is displayed (it might not be the same as what is shown below)



1. Choose: View, SQL View. Note the addition (highlighted below) to the SQL statement:

SELECT Players.[PlayerID], Players.[LName], Players.[FName], Players.[PNumber], Players.[BDate]

FROM Players

WHERE (([LName]<"g"));

1. Modify query to sort on last names. Do the following:
2. Choose: View, Design View. The select the *Sort* column for the *LName* field and choose: Ascending.



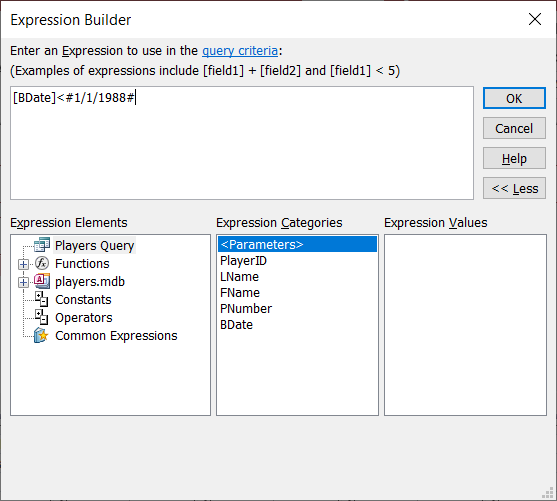
1. Run the query and note the order of the records.
2. Choose: View, SQL View and note the addition to the SQL statement:

SELECT Players.[PlayerID], Players.[LName], Players.[FName], Players.[PNumber], Players.[BDate]

FROM Players

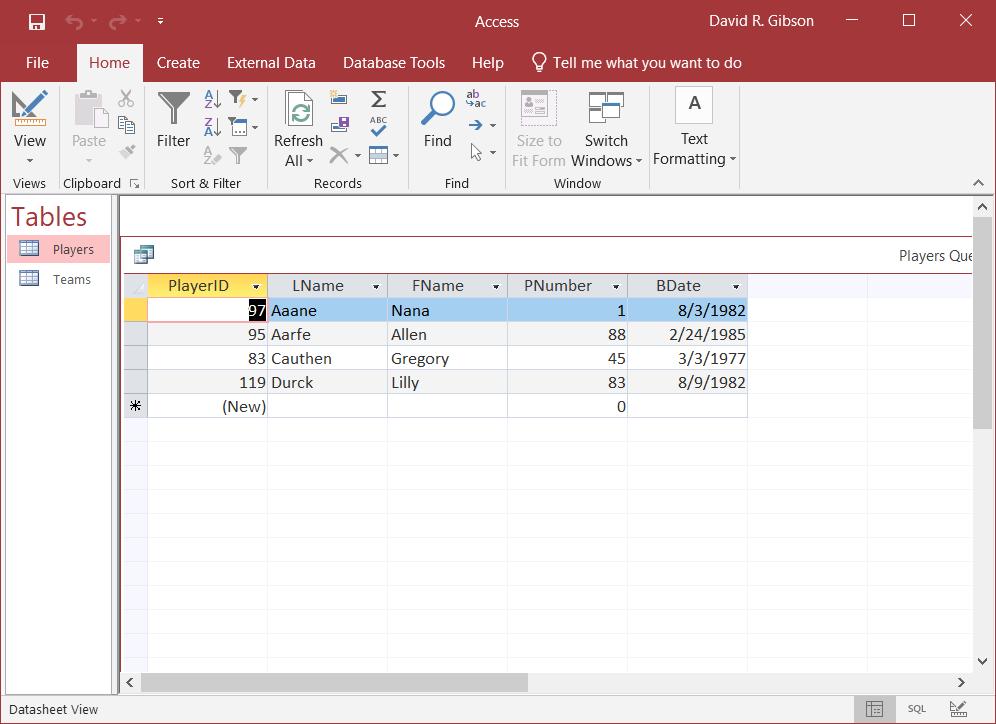
WHERE (([LName]<"g"))

ORDER BY Players.[LName];

1. Modify query to also filter on birth dates before 1/1/1988. Do the following:
2. Choose: View, Design View. Then, place your cursor in the *Criteria* row for the *BDate* column, right-click, and choose: Build. Type this into the dialog (shown on the right also)

[BDate]<#1/1/1988#

1. Choose: OK, and then run the query. The results are similar to what is shown below:

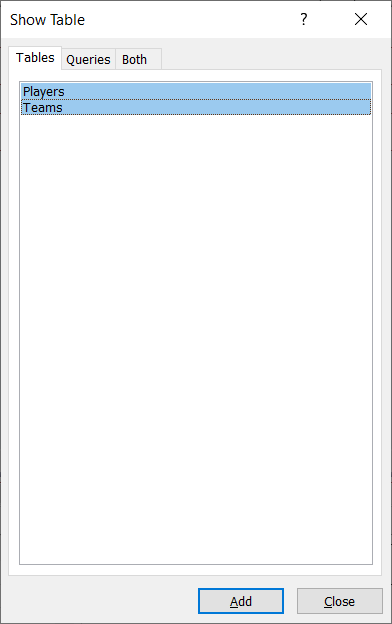


1. Choose: View, SQL View and note the SQL statement:

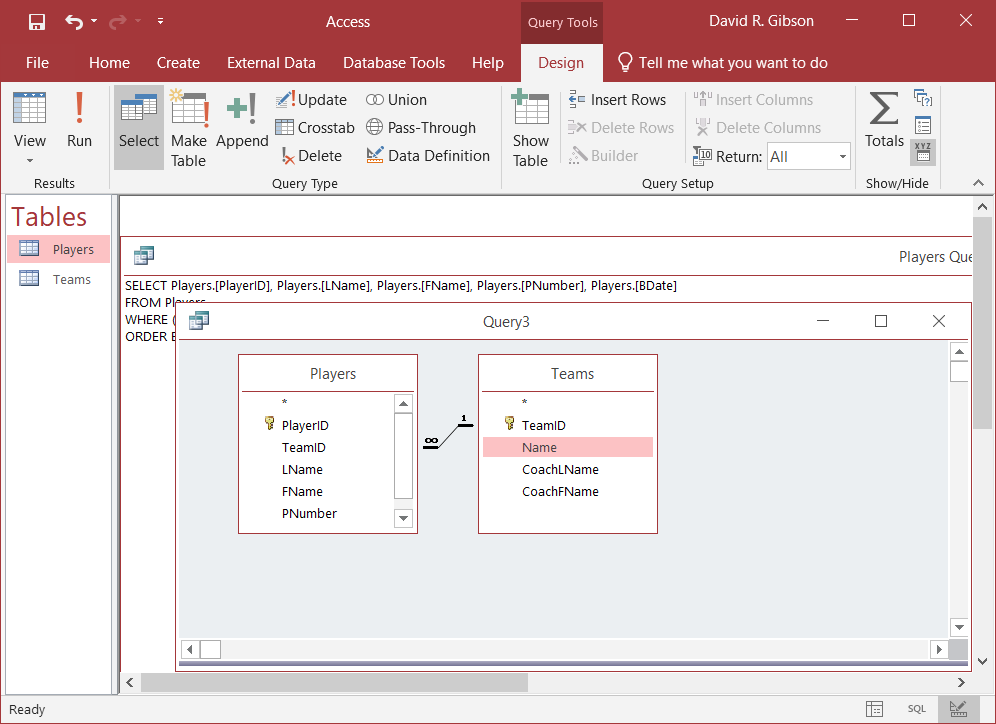
SELECT Players.[PlayerID], Players.[LName], Players.[FName], Players.[PNumber], Players.[BDate]

FROM Players

WHERE (([LName]<"g") AND ([BDate]<#1/1/1988#))

ORDER BY Players.[LName];

1. Next, you are going to build another query, but instead of using the *Query Wizard,* you will use the *Query Designer*.
2. Choose: Create, Query Design
3. Select both tables as shown on the right and choose: Add, and then close.
4. Double click: *LName* in the *Players* table (nothing will appear to happen, but it has just put that in the SQL statement). Then, Double click: *FName*, then double click *Name* in the *Teams* table.

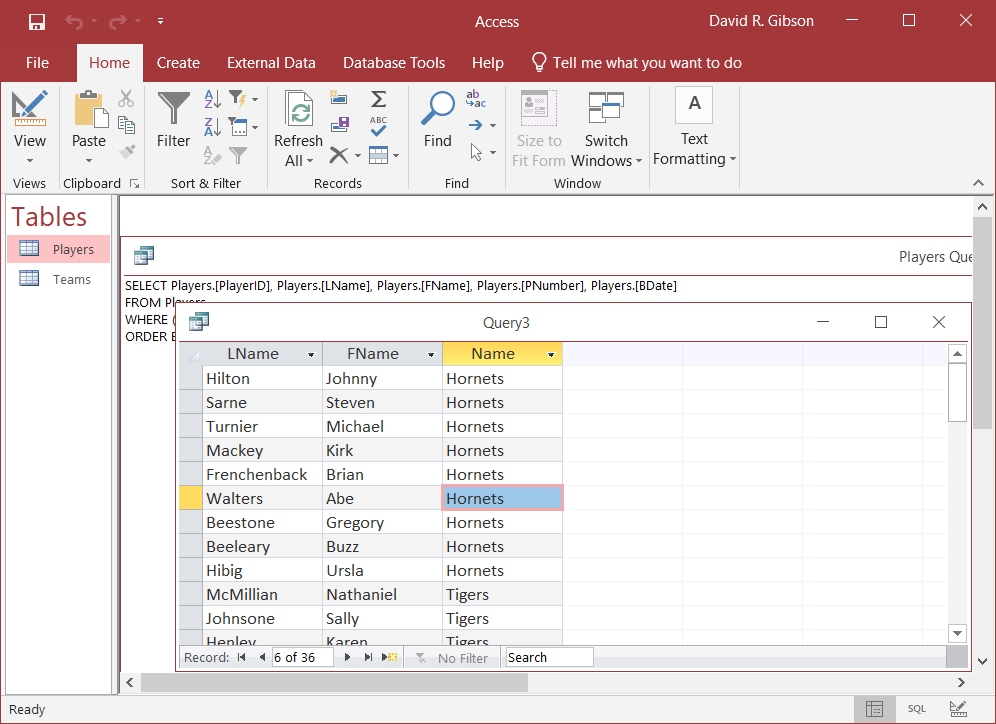


1. Choose: View, SQL View. Inspect the SQL statement. You probably won’t understand it completely. We will talk about this in class.

SELECT Players.LName, Players.FName, Teams.Name

FROM Teams INNER JOIN Players ON Teams.TeamID = Players.TeamID;

1. Run the query, note the players are grouped by team as shown below:



1. Close the query window. It will ask you if you want to save (yes), and give it a name: “by teams”.
2. To see your queries, do the following:

Right-click in the navigation pane (where the tables are displayed) on the left.

Choose: Category, Tables and Related Views. Then, they should be displayed as shown the left in the figure below. Double-click to open one of them. Then close the query.



1. **Create a query on your own, whatever you want.**
2. We are done with this stage. You leave Access open, or close it. I find it convenient to leave it open when I am working with database programming, which we do next.

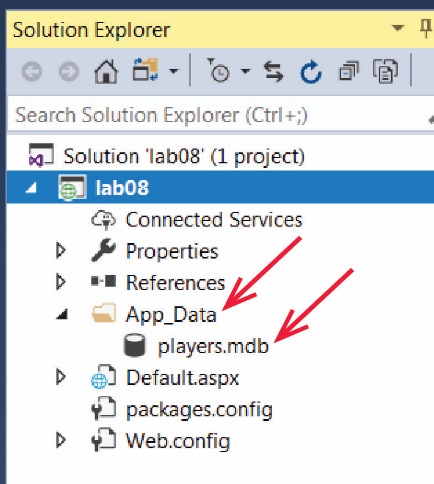
# Databinding: DataSource & GridView

(Read, no action required) In the remainder of the lab we consider web applications that display and manipulate information in a database. In this section we briefly consider *data binding*, which is an approach where we will essentially use wizards to connect to a database and display the data in a table on a Web Form.

This *data binding* approach is actually the next major Topic in this course.

In the remaining stages of this lab, we will learn *manual database*. Manual database means that we will write code using C# API classes to retrieve and display data.

Here, we consider databinding, briefly, as an aid to the work we do in the remainder of this lab and a technique that might help you debug your manual database homework assignment.

1. Create your *lab09\_lastName* project.
2. Do the following:
3. Right-click the project node in the SE and choose: Add, Add ASP.NET Folder, App\_Data

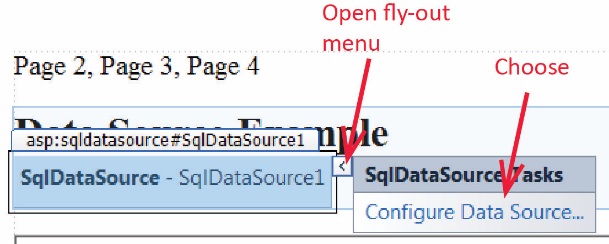
Note: Do not add a regular folder with this name. Make sure you add an *ASP.NET Folder.*

1. Drag the database, *players.mdb* into the *App\_Data* folder.
2. Add a web form named, *Default.aspx*, open in Source mode and add this markup inside the *div* tags

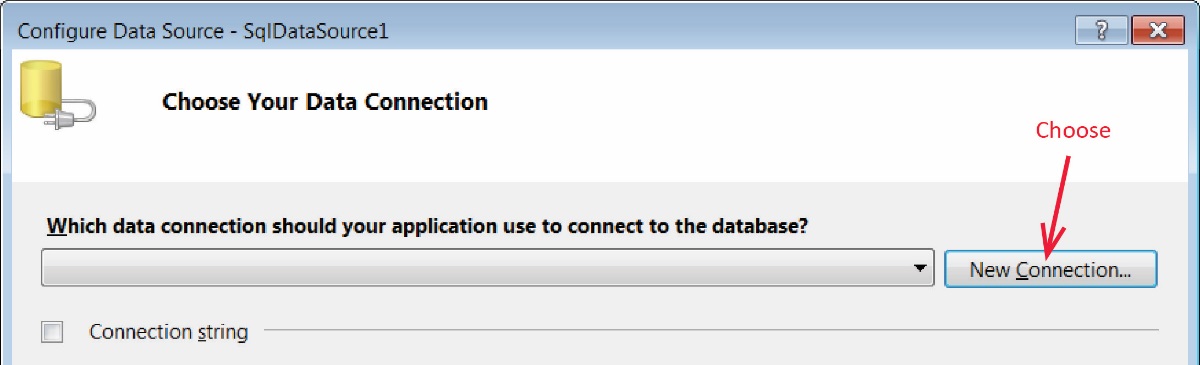
<p>Page 2, Page 3, Page 4</p>

<h2>Data Source Example</h2>

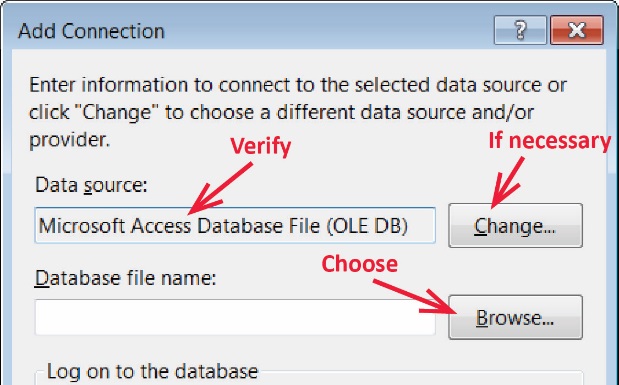
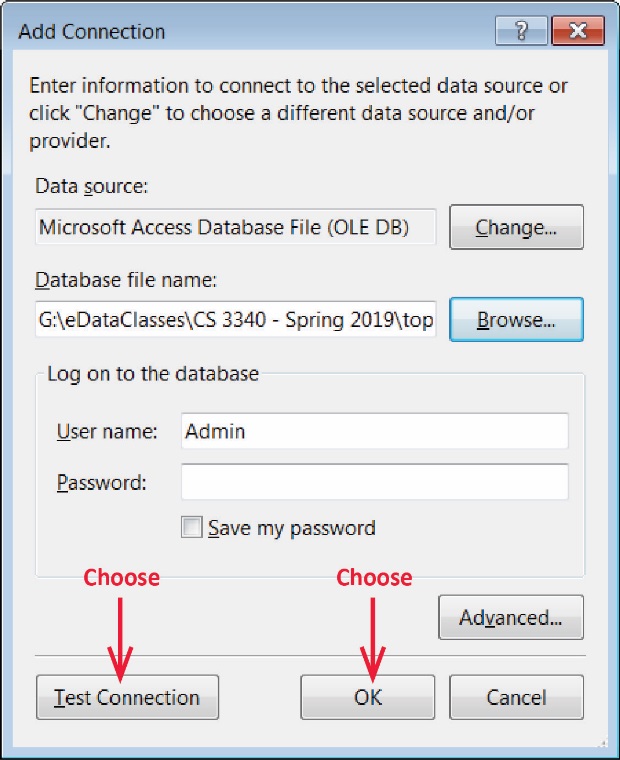
We will add links to “Page 2”, etc as we go along.

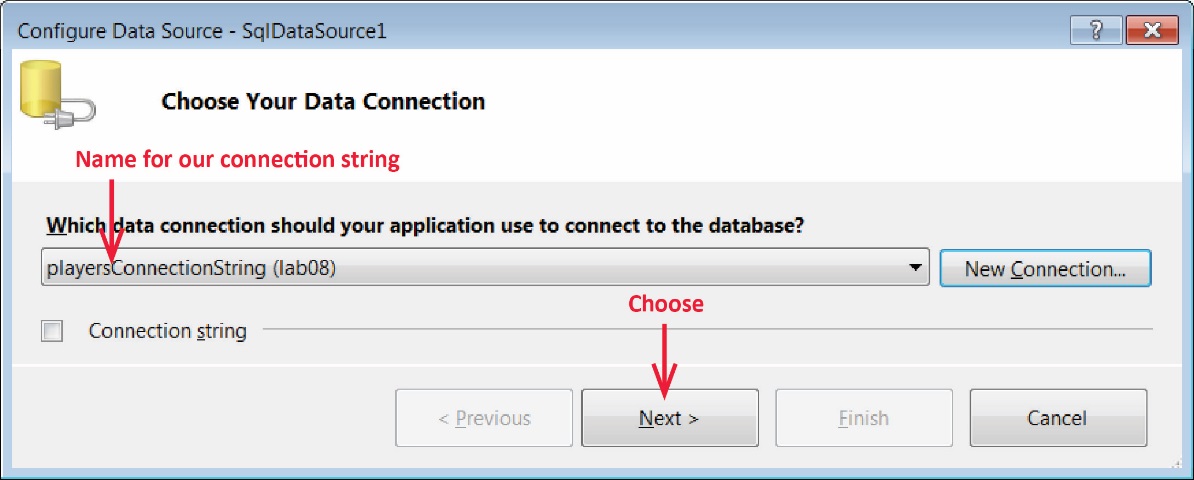
1. Do the following:
2. Open your page in Design mode, position your cursor at the end of the text, “Data Source Example”.
3. In the Toolbox, expand the *Data* group, and double-click, *SqlDataSource*

Note: A *SqlDataSource* is a component and is not displayed at runtime.

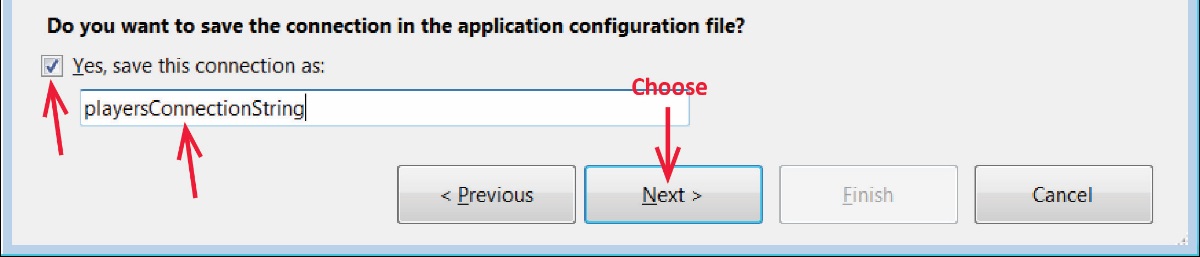
1. Open the fly-out menu and choose: *Configure Data Source*
2. Choose:

*New Connection*

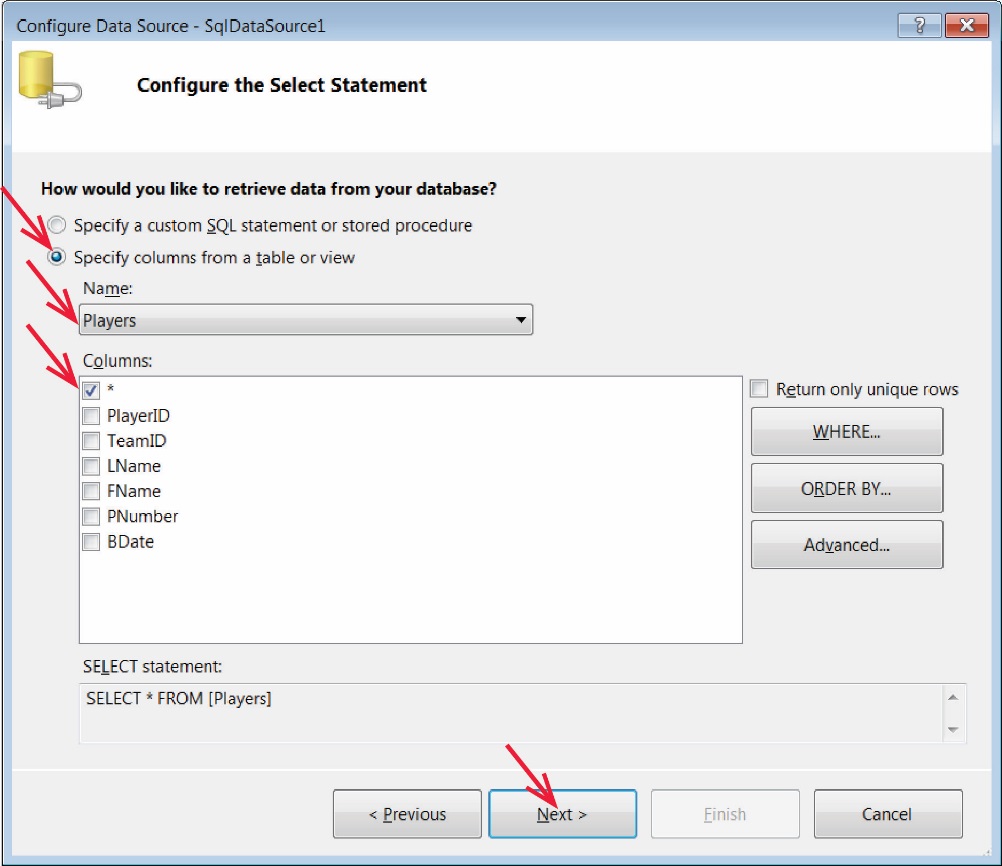
1. Do the following:
2. Verify that the *Data Source* is, “Microsoft Access Database…”. If not, choose: Change and select it from the dialog.
3. Choose: Browse and then navigate to your database, select it, and choose: OK.
4. Choose: Test Connection and verify that it succeeded, and then choose: OK
5. Note the name for the connection string (yours will probably say, *player.mdb*) and then choose: Next.



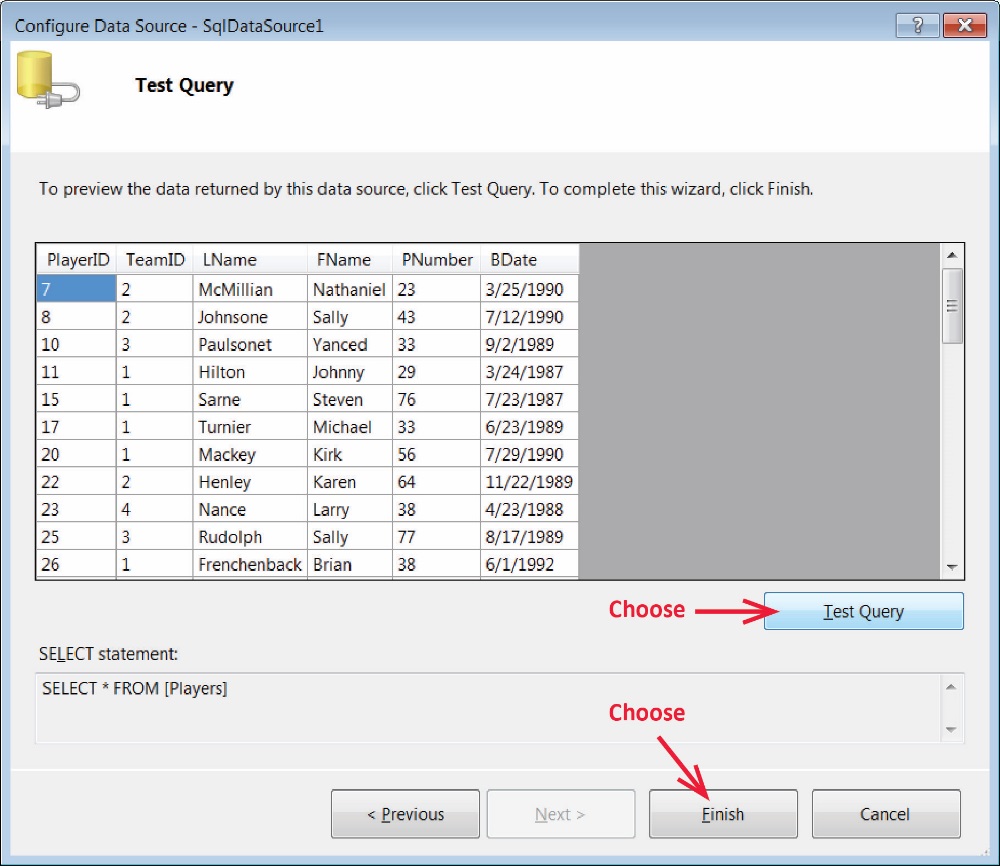
1. Select the checkbox to save the connection string, name the connection string, *playersConnectionString* and choose: Next.



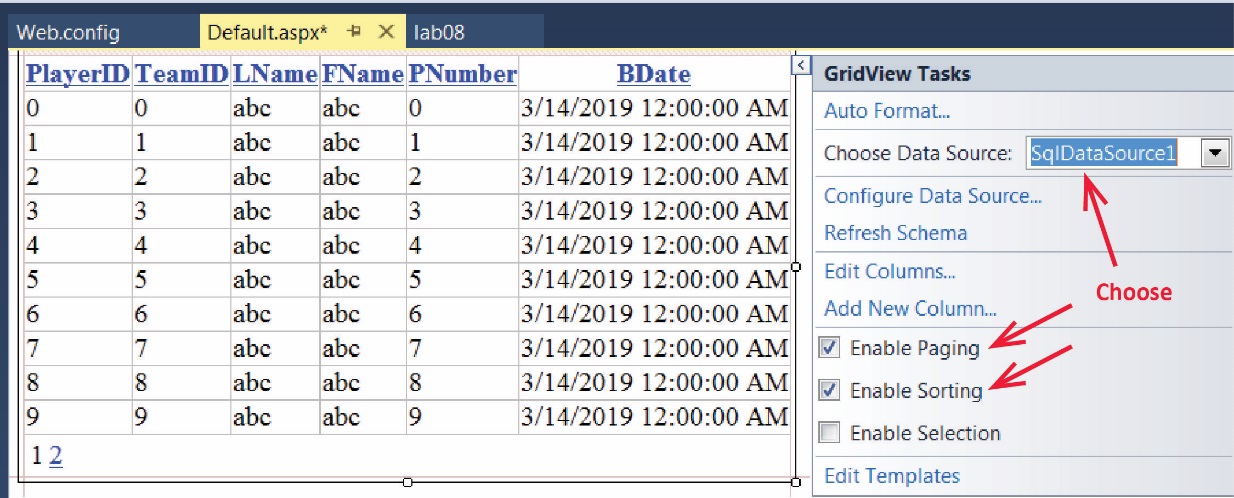
1. Make sure the items below are selected and then choose: Next



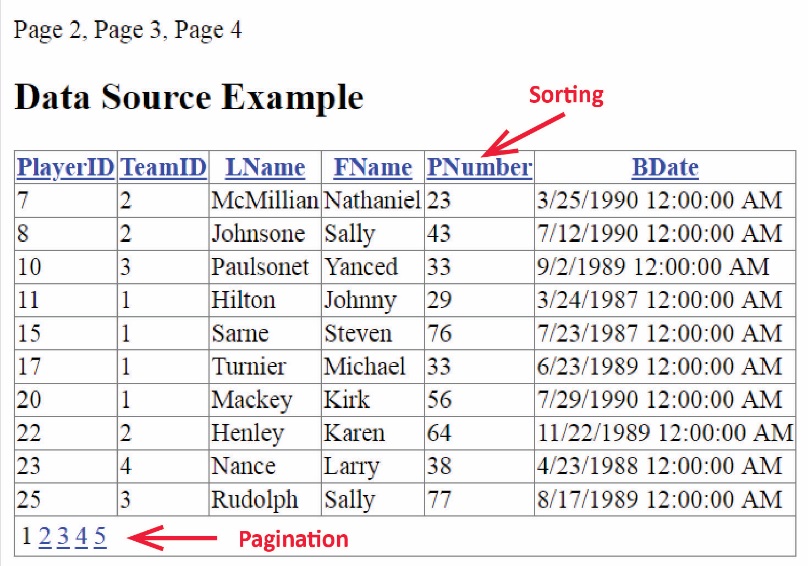
1. Choose: Test Query and verify that the data is displayed. Then choose: Finish

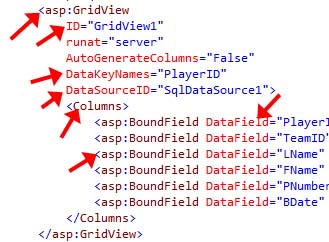


1. Do the following:
2. Position your cursor below the *SqlDataSource*.
3. In the Toolbox, open the *Data* group and double-click *GridView*. This will place a grid structure on your page.
4. Open the *GridView’s* fly-out menu and select the items shown below.



1. Run your page and it should appear as shown below. You can press the links in the column headers to sort on that field.



1. Open your page in Source view. Find the GridView tag and note its structure.Find the following attributes in the markup shown on your page and note the definitions:

* *ID* – the name of this instance of the GridView
* Not shown in the figure, but present in your code are the *AllowPaging* and *AllowSorting* attributes.
* *DataKeyNames* – the key field in the database table. This will be discussed in class.
* *DataSourceID* – the ID of the data source
* Columns – the GridView has a collection of columns
* *BoundField* – the *type* of column (we will study other types later). Used to bind a field in the database to this field in the GridView
* *DataField* – The name of the field in the database that is being bound

1. (Read, no action required) As stated above, we will use the *DataSource* and *GridView* as an aid to the manual database work we do in the subsequent stages. As you can see, the process we just completed is very easy to do. That is in contrast to the manual database work we do next which will be much more challenging in practice. It is convenient to display the data in a *GridView* to verify work that we do later is correct.

# Explore Connection String and DataSource Structure

1. Open *Web.config* and find the *connectionStrings* node as shown below. Do the following:
2. The *Data Source* attribute (highlighted yellow) specifies the location of the database. It should appear exactly as shown below (highlighted in yellow). It should not be hard-coded with a path, but rather should use the relative path, “|DataDirectory|”.

<connectionStrings>

<add name="playersConnectionString"

connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=|DataDirectory|\players.mdb"

providerName="System.Data.OleDb" />

</connectionStrings>

1. Run your page and verify that it still works.
2. Run your page and verify that it still works.
3. Open your page in Source mode and find the *SqlDataSource* tag. Note that the *ConnectionString* and *ProviderName* are read from *Web.config* (highlighted in green below). Note, you can create a line break to view more easily, as I have done below.

<asp:SqlDataSource ID="SqlDataSource1"

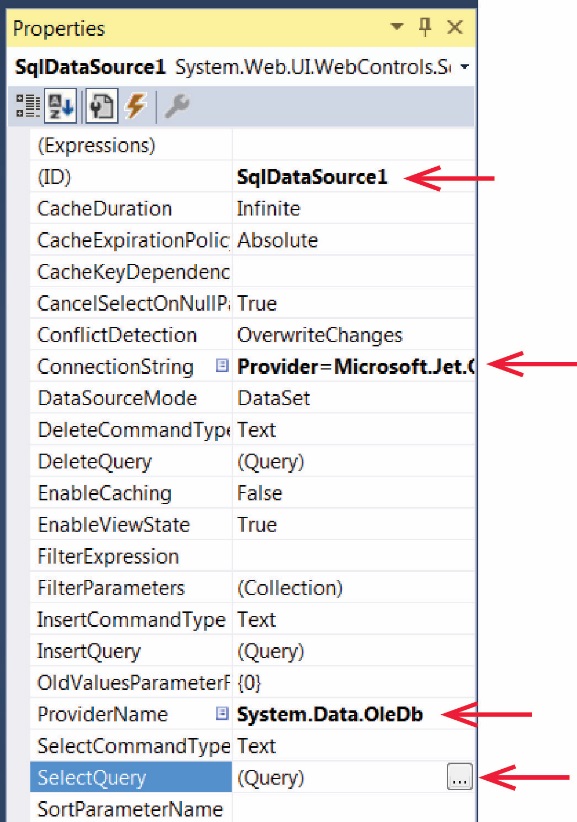
runat="server"

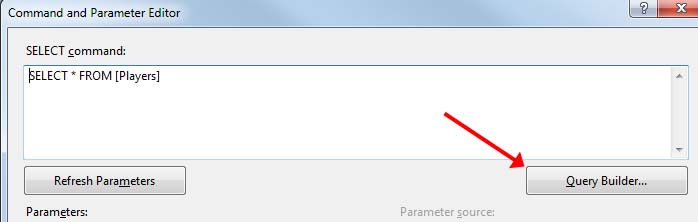
ConnectionString="<%$ ConnectionStrings:playersConnectionString %>"

ProviderName="<%$ ConnectionStrings:playersConnectionString.ProviderName %>"

SelectCommand="SELECT \* FROM [Players]">

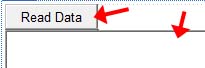
</asp:SqlDataSource>

1. Display your page in Design mode. Select the data source and note the items below in the Properties window.
2. Still in the Properties Window, expand the *SelectQuery* choice. We will use the *Query Builder* later. This is literally the same as the one we used with Access in Stage 1. Close this dialog.



# Manual Database – Select Statement

The approach we used in [Section 4](#_Databinding:_DataSource_&) is very easy to do. I call this the *Wizard* approach as we used wizards to do all the work. Sometimes this approach is sufficient. However, sometimes we need to have more control over the configuration, or the ability to directly touch the data with code. I call this the *manual* approach. In this stage we will work with the same *Players* table; however, we will not use the *DataSource*. Instead, we will use classes from the .NET API to connect to the database and read data *manually*.

1. **Set-Up New Page** – Do the following:
2. Create a new web form named: *Page2.aspx*
3. Open *Default.aspx* and add a link from your Lab 8 homepage (*Default.aspx*) to this new page (Select the text, “Page 2”, then choose: Format, Convert to Hyperlink, Browse).
4. Return to *Page2.aspx* in Design mode. Add a Button with *ID* set to: *btnReadData* as shown on the right.
5. Add a multiline TextBox with *ID* set to: *txtMsg* as shown on the right (be sure and set the *TextMode* property to “MultiLine”). Stretch the text box larger.
6. **Create Event-Handler for Button** – Do the following:
7. Double-click the button to create the event-handler stub.
8. Study the code below and then add it to the event-handler. You need to add some using statements (see part *c* below).

// Create connection object

IDbConnection con = new OleDbConnection();

// Create command object

IDbCommand cmd = new OleDbCommand();

// Get connection string from web.config.

string conn = ConfigurationManager.ConnectionStrings["playersConnectionString"].ConnectionString;

// Attach connection string to connection object.

con.ConnectionString = conn;

// Attach the connection object to the command object

cmd.Connection = con;

// Display information about the connection.

txtMsg.Text = "\*\*\*Connection String: \n" + con.ConnectionString + "\n\n";

txtMsg.Text += "\*\*\*Connection State: " + con.State + "\n\n";

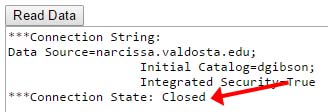
1. Resolve the compiler errors manually, or, supply these using statements at the top of your file:

using System;

using System.Configuration;

using System.Data;

using System.Data.OleDb;

1. View your page and press the “Read Data” button. The resulting page should look as shown on the right (your connection string will be different). The connection is *Closed* as we have not opened it yet. This is just to show you that that we can query the state of the connection as shown in the code above: con.State.
2. **Set the SQL Statement** – We will write code to build a SQL statement that will be used later to read data from the database. Study the code below and then add it at the **end**of the button click event:

// Build sql string

string sql =

"SELECT " +

"Players.PlayerID, " +

"Players.TeamID, " +

"Players.LName, " +

"Players.FName, " +

"Players.PNumber, " +

"Players.BDate " +

"FROM " +

"Players " +

"ORDER BY " +

"Players.LName Asc, " +

"Players.FName Asc";

// Add sql to command object

cmd.CommandText = sql;

// Display SQL statement. Very useful debugging technique.

txtMsg.Text += "\*\*\*CommandText: " + cmd.CommandText + "\n\n";

1. **Open Connection to Database** – Study the code below and then add it to the **end** of the code in the button click event.

try {

// Open the connection.

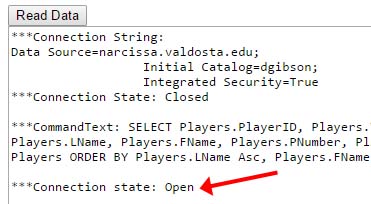
cmd.Connection.Open();

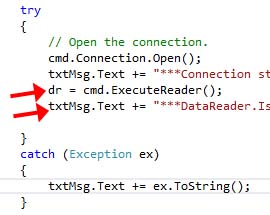
txtMsg.Text += "\*\*\*Connection state: " + cmd.Connection.State + "\n\n";

}

catch (Exception ex) {

txtMsg.Text += ex.ToString();

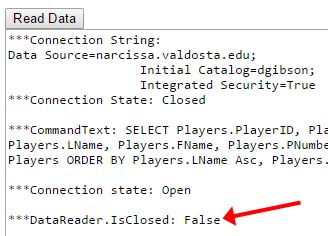
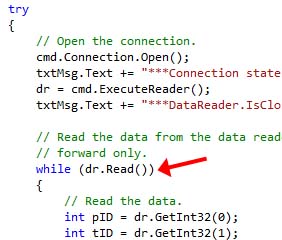
 }

1. View your page. Verify that the Connection state is “Open” as shown on the right.
2. **Create the DataReader** – Add the lines below to the **end** of the code **inside the try block** (as shown on the right):

// Read the data and put the results in data reader.

IDataReader dr = cmd.ExecuteReader();

txtMsg.Text += "\*\*\*DataReader.IsClosed: " + dr.IsClosed + "\n\n";

1. View your page. Verify that DataReader.IsClosedis “False” as shown on the right. This means it is ready to have data extracted from it.
2. **Extract the Data from the DataReader** – Study the code below and also read the notes after the code. When finished, add the code to the **end** of the code **inside the *try* block** (just below the lines from the previous step) as shown on the right.

// Read the data from the data reader.

// Note that this is one-pass, forward only.

while (dr.Read())

{

// Read the data. Must specify the data type you

// are reading (e.g. GetInt32) and the position

// in the reader: 0, 1, etc. Note that this order

// is the same as the order in the SQL statement.

int pID = dr.GetInt32(0);

int tID = dr.GetInt32(1);

string lName = dr.GetString(2);

string fName = dr.GetString(3);

int jerNum = dr.GetInt32(4);

DateTime bDate = dr.GetDateTime(5);

// Do something with the data, i.e. put it in a textbox.

txtMsg.Text += pID + " " + tID + " " + lName + " " + fName + " " + jerNum + " " +

bDate.ToShortDateString() + "\n";

}

// Close data reader and connection

dr.Close();

cmd.Connection.Close();

txtMsg.Text += "\*\*\*DataReader.IsClosed: " + dr.IsClosed + "\n\n";

txtMsg.Text += "\*\*\*Connection state: " + cmd.Connection.State + "\n\n";

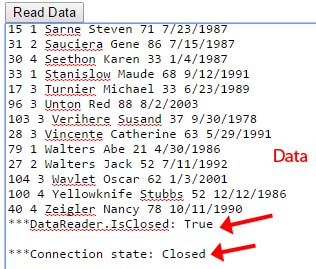
1. (Read, no action required)

* The *IDataReader* object is similar to the *Scanner* object in Java. The API for the *IDataReader* interface is [here](https://docs.microsoft.com/en-us/dotnet/api/system.data.idatareader?view=netframework-4.8).

|  |  |
| --- | --- |
| ***DataReader* (C#)** | ***Scanner* (Java)** |
| *ExecuteReader* | *n/a* |
| *Read():Boolean* | *hasNext* |
| *GetString(pos):String* | *next* |
| *GetInt32(pos):int* | *nextInt* |
| *GetDouble(pos):int* | *nextDouble* |
| *GetDateTime(pos):DataTime* | *n/a* |
| *GetValue(pos):Object* | *n/a* |
| *Close()* | *close()* |

* The *IDataReader* methods accept an integer which is the column number for the current row. For example, our SQL statement selected the fields in the order shown, and thus the *DataReader* extracts them in that order.

|  |  |
| --- | --- |
| Select Statement | DataReader |
| "SELECT " +  "Players.PlayerID, " +  "Players.TeamID, " +  "Players.LName, " +  "Players.FName, " +  "Players.PNumber, " +  "Players.BDate " + | int pID = dr.GetInt32(0);  int tID = dr.GetInt32(1);  string lName = dr.GetString(2);  string fName = dr.GetString(3);  int jerNum = dr.GetInt32(4);  DateTime bDate = dr.GetDateTime(5); |

1. View your page. Verify (as shown on the right):
2. The data is displayed.
3. At the end, the DataReader.IsClosedis “True”
4. The Connection state is “Closed”.
5. **Review** – Review your button click code and verify the algorithm below for obtaining data from a database. This is frequently a test question: Suppose you are given a connection string, *playersConnectionString* to an Access database with table Players (an ER diagram would be provide). Write the code required to display the *LName* of all players in a textbox, *txtPlayers*.
6. Create a *Connection* object and a *Command* object.

IDbConnection con = new OleDbConnection();

IDbCommand cmd = new OleDbCommand();

1. Retrieve the connection string from *Web.config*.

string conn = ConfigurationManager.ConnectionStrings["playersConnectionString"].ConnectionString;

1. Put the connection string in the *Connection* object.

con.ConnectionString = conn;

1. Put the *Connection* object in the *Command* object.

cmd.Connection = con;

1. Build SQL string and put in *Command* object.

string sql =

"SELECT " +

"Players.PlayerID, " + ...

cmd.CommandText = sql;

1. Open the connection.

cmd.Connection.Open();

1. Create a *DataReader* object from the *Command* object (*e.g.* read the data into a data reader)

IDataReader dr = cmd.ExecuteReader();

1. Loop through the *DataReader* object and extract the data. (*e.g.* extract data from data reader)

while (dr.Read()) {

int pID = dr.GetInt32(0);

...

}

1. Close the *DataReader* and *Connection*.

dr.Close();

cmd.Connection.Close();

# Manual Database – Insert Statement

Here we demonstrate the insert statement. As well, we modularize the code to display the data.

1. **Set-Up New Page** – Do the following:
2. Create a new web form named: *Page3.aspx* and create a link to it on *Default.aspx.*
3. Add this markup to the Source of *Page3.aspx* inside the *div* tags.

<p>Insert Sql Statements</p>

<p>

<asp:Button ID="btnInsert" runat="server" Text="Insert" />

&nbsp;Team ID\*<asp:TextBox ID="txtTeamID" runat="server" Width="22px"></asp:TextBox>

&nbsp;LName<asp:TextBox ID="txtLName" runat="server" Width="98px"></asp:TextBox>

&nbsp;FName<asp:TextBox ID="txtFName" runat="server" Width="98px"></asp:TextBox>

&nbsp;PNum<asp:TextBox ID="txtPNum" runat="server" Width="38px"></asp:TextBox>

&nbsp;BDate<asp:TextBox ID="txtBDate" runat="server" Width="98px"></asp:TextBox>

</p>

<p>

\*Team ID must be valid value from the Teams table in the TeamID column (1,2,3,4, or 8, unless you have deleted a team)

</p>

<asp:GridView ID="gvPlayers" runat="server">

</asp:GridView>

<p>

<asp:TextBox ID="txtMsg" runat="server" Height="314px" TextMode="MultiLine" Width="593px"></asp:TextBox>

</p>

1. **Display Data** – Do the following:
2. (Read, no action required) First, we will display all the data when the page is first loaded. We will do this in a modularized fashion by writing helper methods. This will be helpful when we implement the insert feature, so that we can verify that it was indeed inserted.
3. Add these methods to the code behind file (you’ll need to resolve the missing *using* statements). Notice in the first method below, we are binding the *DataReader* (*dr*) to a *GridView* (*gvPlayers*).

/// <summary>

/// Displays the data from the Players table in a GridView.

/// </summary>

private void displayData() {

IDbCommand cmd = getCommand();

cmd.CommandText = getSelectSql();

try {

// Open the connection.

cmd.Connection.Open();

// Load data into the reader

IDataReader dr = cmd.ExecuteReader();

// Link the Gridview to the reader

gvPlayers.DataSource = dr;

// Bind the reader to GridView, i.e. put the data into the GridView.

gvPlayers.DataBind();

dr.Close();

cmd.Connection.Close();

}

catch (Exception ex) {

txtMsg.Text += ex.ToString();

}

}

/// <summary>

/// Returns a Command object that is connected to a Connection object. The Connection

/// object has its ConnectionString property set to the value in web.config.

/// </summary>

/// <returns>Command object</returns>

private IDbCommand getCommand() {

IDbConnection con = new System.Data.OleDb.OleDbConnection();

IDbCommand cmd = new System.Data.OleDb.OleDbCommand();

string connectionString = System.Configuration.ConfigurationManager.ConnectionStrings["playersConnectionString"].ConnectionString;

con.ConnectionString = connectionString;

cmd.Connection = con;

return cmd;

}

/// <summary>

/// Returns a SQL statement that selects all the data from the Players table

/// sorted on last name, then first name.

/// </summary>

/// <returns>Select SQL statement</returns>

private string getSelectSql() {

string sql =

"SELECT " +

"Players.PlayerID, " +

"Players.TeamID, " +

"Players.LName, " +

"Players.FName, " +

"Players.PNumber, " +

"Players.BDate " +

"FROM " +

"Players " +

"ORDER BY " +

"Players.LName Asc, " +

"Players.FName Asc";

return sql;

}

1. Add this line of code to *Page\_Load* to call *displayData* when the page is first loaded.

// Display the data if this is the first time on the page.

if (!Page.IsPostBack) {

displayData();

}

1. Run your page and verify that the data is displayed.
2. **Implement Insert Feature** – Do the following:
3. Double-click the Insert button in Design view to create the click event handler.
4. Add these helper methods to the code behind file:

/// <summary>

/// Returns a SQL statement that inserts a row into the Players table, using the

/// values from the text boxes for the field values in the table.

/// </summary>

/// <returns>Insert SQL statement</returns>

private string getInsertSql() {

string sql =

"INSERT INTO Players " +

"(TeamID, LName, FName, PNumber, BDate) " +

"VALUES ( " +

txtTeamID.Text + ", " +

"'" + txtLName.Text + "', " +

"'" + txtFName.Text + "', " +

txtPNum.Text + ", " +

"'" + txtBDate.Text + "'" +

")";

return sql;

}

/// <summary>

/// Clear the text boxes.

/// </summary>

private void clearTextBoxes() {

txtTeamID.Text = String.Empty;

txtLName.Text = String.Empty;

txtFName.Text = String.Empty;

txtPNum.Text = String.Empty;

txtBDate.Text = String.Empty;

}

1. Add this code to the *btnInsert\_Cllick*

// Create the Command object

IDbCommand cmd = getCommand();

// Set the Insert SQL statement into the Command

cmd.CommandText = getInsertSql();

// Display the SQL statement

txtMsg.Text = "INSERT Sql statement:\n" + cmd.CommandText + "\n";

try {

// Open the connection.

cmd.Connection.Open();

// Inserts the row into the Players table and returns how many rows are affected.

// If successful, this will be 1, as 1 row has been added. If not successfull,

// then 0 will be returned.

int rowsAffected = cmd.ExecuteNonQuery();

txtMsg.Text += "Rows affected=" + rowsAffected + "\n";

cmd.Connection.Close();

// Display the updated GridView, which includes the row that was added.

displayData();

clearTextBoxes();

}

catch (Exception ex) {

txtMsg.Text += ex.ToString();

}

1. Run your page and experiment. Be sure (a) validate that your data was inserted (the GridView is sorted on LName, so the addition will probably not be at the end of the table), (b) and check the multiline textbox to see any output there. Type in an invalid value for TeamID and look in the multiline textbox and you will see an error. (See the note on the page itself about valid values).

# Manual Database – Update & Delete Statements

Here we demonstrate the update and delete Sql statements. You will copy the entire code below. You should read through the code.

1. **Set-Up New Page** – Do the following:
2. Create a new web form named: *Page4.aspx* and create a link to it on *Default.aspx.*
3. Add this markup to the Source of *Page4.aspx* inside the *div* tags.

<p>Update &amp; Delete Sql Statements</p>

<p>

<asp:Button ID="btnUpdate" runat="server" Text="Update" OnClick="btnUpdate\_Click" />

&nbsp;Player ID<asp:TextBox ID="txtPlayerID" runat="server" Width="22px"></asp:TextBox>

&nbsp;Team ID\*<asp:TextBox ID="txtTeamID" runat="server" Width="22px"></asp:TextBox>

&nbsp;LName<asp:TextBox ID="txtLName" runat="server" Width="98px"></asp:TextBox>

&nbsp;FName<asp:TextBox ID="txtFName" runat="server" Width="98px"></asp:TextBox>

&nbsp;PNum<asp:TextBox ID="txtPNum" runat="server" Width="38px"></asp:TextBox>

&nbsp;BDate<asp:TextBox ID="txtBDate" runat="server" Width="98px"></asp:TextBox>

</p>

<p>

\*Team ID must be valid value from the Teams table in the TeamID column (1,2,3,4, or 8, unless you have deleted a team)

</p>

<p>

<asp:Button ID="btnDelete" runat="server" OnClick="btnDelete\_Click" Text="Delete" />

&nbsp;Player ID<asp:TextBox ID="txtPlayerIDDelete" runat="server" Width="22px"></asp:TextBox>

&nbsp;</p>

<asp:GridView ID="gvPlayers" runat="server">

</asp:GridView>

<p>

<asp:TextBox ID="txtMsg" runat="server" Height="314px" TextMode="MultiLine" Width="593px"></asp:TextBox>

</p>

1. Add this code to the code-behind file (spans this page and next 2). Note: you will be replacing everything inside the *page4* class:

protected void Page\_Load(object sender, EventArgs e) {

// Display the data if this is the first time on the page.

if (!Page.IsPostBack) {

displayData();

}

}

/// <summary>

/// Deletes a row from the Players table.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

protected void btnDelete\_Click(object sender, EventArgs e) {

// Create the Command object

IDbCommand cmd = getCommand();

// Set the Delete SQL statement into the Command

cmd.CommandText = getDeleteSql();

// Display the SQL statement

txtMsg.Text = "DELETE Sql statement:\n" + cmd.CommandText + "\n";

try {

// Open the connection.

cmd.Connection.Open();

// Deletes the row from the Players table and returns how many rows are affected.

// If successful, this will be 1, as 1 row has been deleted. If not successfull,

// then 0 will be returned.

int rowsAffected = cmd.ExecuteNonQuery();

txtMsg.Text += "Rows affected=" + rowsAffected + "\n";

cmd.Connection.Close();

// Display the updated GridView.

displayData();

clearTextBoxes();

}

catch (Exception ex) {

txtMsg.Text += ex.ToString();

}

}

/// <summary>

/// Updates a row in the Players table.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

protected void btnUpdate\_Click(object sender, EventArgs e) {

// Create the Command object

IDbCommand cmd = getCommand();

// Set the Update SQL statement into the Command

cmd.CommandText = getUpdateSql();

// Display the SQL statement

txtMsg.Text = "UPDATE Sql statement:\n" + cmd.CommandText + "\n";

try {

// Open the connection.

cmd.Connection.Open();

// Updates the row in the Players table and returns how many rows are affected.

// If successful, this will be 1, as 1 row has been added. If not successfull,

// then 0 will be returned.

int rowsAffected = cmd.ExecuteNonQuery();

txtMsg.Text += "Rows affected=" + rowsAffected + "\n";

cmd.Connection.Close();

// Display the updated GridView

displayData();

clearTextBoxes();

}

catch (Exception ex) {

txtMsg.Text += ex.ToString();

}

}

/// <summary>

/// Displays the data from the Players table in a GridView.

/// </summary>

private void displayData() {

IDbCommand cmd = getCommand();

cmd.CommandText = getSelectSql();

try {

// Open the connection.

cmd.Connection.Open();

// Load data into the reader

IDataReader dr = cmd.ExecuteReader();

// Link the Gridview to the reader

gvPlayers.DataSource = dr;

// Bind the reader to GridView, i.e. put the data into the GridView.

gvPlayers.DataBind();

dr.Close();

cmd.Connection.Close();

}

catch (Exception ex) {

txtMsg.Text += ex.ToString();

}

}

/// <summary>

/// Returns a Command object that is connected to a Connection object. The Connection

/// object has its ConnectionString property set to the value in web.config.

/// </summary>

/// <returns>Command object</returns>

private IDbCommand getCommand() {

IDbConnection con = new System.Data.OleDb.OleDbConnection();

IDbCommand cmd = new System.Data.OleDb.OleDbCommand();

string connectionString = System.Configuration.ConfigurationManager.ConnectionStrings["playersConnectionString"].ConnectionString;

con.ConnectionString = connectionString;

cmd.Connection = con;

return cmd;

}

/// <summary>

/// Returns a SQL statement that selects all the data from the Players table

/// sorted on last name, then first name.

/// </summary>

/// <returns>Select SQL statement</returns>

private string getSelectSql() {

string sql =

"SELECT " +

"Players.PlayerID, " +

"Players.TeamID, " +

"Players.LName, " +

"Players.FName, " +

"Players.PNumber, " +

"Players.BDate " +

"FROM " +

"Players " +

"ORDER BY " +

"Players.LName Asc, " +

"Players.FName Asc";

return sql;

}

/// <summary>

/// Returns a SQL statement that updates a row in the Players table, using the

/// values from the text boxes for the field values in the table.

/// </summary>

/// <returns>Insert SQL statement</returns>

private string getUpdateSql() {

string sql =

"UPDATE Players SET " +

"TeamID=" + txtTeamID.Text + ", " +

"LName='" + txtLName.Text + "', " +

"FName='" + txtFName.Text + "', " +

"PNumber=" + txtPNum.Text + ", " +

"BDate='" + txtBDate.Text + "' " +

"WHERE PlayerID=" + txtPlayerID.Text;

return sql;

}

/// <summary>

/// Clear the text boxes.

/// </summary>

private void clearTextBoxes() {

txtPlayerID.Text = String.Empty;

txtTeamID.Text = String.Empty;

txtLName.Text = String.Empty;

txtFName.Text = String.Empty;

txtPNum.Text = String.Empty;

txtBDate.Text = String.Empty;

txtPlayerIDDelete.Text = String.Empty;

}

/// <summary>

/// Returns a SQL statement that deletes a row in the Players table.

/// </summary>

/// <returns>Insert SQL statement</returns>

private string getDeleteSql() {

string sql =

"DELETE FROM Players WHERE PlayerID=" + txtPlayerIDDelete.Text;

return sql;

}

1. Run your page and experiment. Study the code to see how it works.

# Submission

1. Close VS and zip your *lab09\_lastName* project folder and submit on Blazeview in the *Lab 09* dropbox.