

Hands-On Lab

Introduction to SQL Azure

Lab version: 2.0.0

Last updated: 8/5/2011



# Contents

[Overview 3](#_Toc300312854)

[Exercise 1: Preparing Your SQL Azure Account 5](#_Toc300312855)

[Task 1 – Retrieving your SQL Azure Server Name 5](#_Toc300312856)

[Exercise 2: Working with Data Basic DDL and DML 11](#_Toc300312857)

[Task 1 – Creating a New Database 12](#_Toc300312858)

[Task 2 – Managing your Database with the Database Manager for SQL Azure 13](#_Toc300312859)

[Task 3 – Managing your Database with SQL Server Management Studio 19](#_Toc300312860)

[Task 4 – Creating Logins and Database Users 22](#_Toc300312861)

[Task 5 – Creating Tables, Indices, and Queries 26](#_Toc300312862)

[Exercise 3: Build a Windows Azure Application that Accesses SQL Azure 30](#_Toc300312863)

[Task 1 – Loading the Sample Database into SQL Azure 30](#_Toc300312864)

[Task 2 – Creating the Visual Studio Project 31](#_Toc300312865)

[Exercise 4: Connecting via Client Libraries 43](#_Toc300312866)

[Task 1 – Opening the Begin Solution and Exploring the Common Functionalities 44](#_Toc300312867)

[Task 2 – Connecting to SQL Azure Using ADO.NET 46](#_Toc300312868)

[Task 3 – Connecting to SQL Azure Using ODBC 51](#_Toc300312869)

[Task 4 – Connecting to SQL Azure Using OLEDB 56](#_Toc300312870)

[Task 5 – Connecting to SQL Azure Using Entity Framework 60](#_Toc300312871)

[Task 6 – Connecting to SQL Azure via Non-Microsoft Technologies 68](#_Toc300312872)

[Summary 69](#_Toc300312873)

Overview

* 1. SQL Azure makes the power of Microsoft SQL Server available in a Cloud Hosted offering. Working with SQL Azure should be a familiar experience for most developers because, for the most part, it supports the same tooling and development practices currently used for on premise SQL Server applications.
  2. However, there are some small differences between working with SQL Azure and working with on-premise SQL Server. Some of these differences are inherent in the way that SQL Azure has been architected and some will only apply during the Community Technical Preview phase.
  3. This hands-on lab will walk through a series of simple use cases for SQL Azure such as provisioning your account, creating, and using a database. You will create a simple Windows Azure application to allow you to manipulate the data in the Contact table of a database running in SQL Azure.

# Objectives

* 1. In this hands-on lab, you will:
  + Use SQL Azure as a cloud hosted database platform for your applications.
  + Learn how to:
    - Provision a new account in SQL Azure
    - Create new databases within the context of your account
    - Create new users and grant them appropriate permissions
  + Work with SQL Azure to:
    - Execute Data Definition Language statements to create tables and indexes
    - Execute Data Manipulation Language to insert and query rows
  + Build a simple data driven ASP.NET page using the graphical controls in Visual Studio.
  + Connect to SQL Azure Database via Client Libraries.



# Prerequisites

* 1. The following is required to complete this hands-on lab:
  + [Microsoft .NET Framework 4.0](http://go.microsoft.com/fwlink/?linkid=186916)
  + [Microsoft Visual Studio 2010](http://msdn.microsoft.com/vstudio/products/)
  + [SQL Server Management Studio 2008 R2 Express Edition](http://www.microsoft.com/downloads/details.aspx?familyid=56AD557C-03E6-4369-9C1D-E81B33D8026B)
  + [Windows Azure Tools for Microsoft Visual Studio 1.4](http://go.microsoft.com/fwlink/?LinkID=128752)
  + Access to a **SQL Azure** account with a server created
    - **SQL Azure Firewall** enabled for machine running this lab
  1. **Note:** This hands-on lab has been designed to use the latest release of the Windows Azure Tools for Visual Studio 2010 (version 1.4) and the new Windows Azure Platform Management Portal experience.
  2. To complete this hands-on lab, you need to have a SQL Azure account. To sign up, visit <http://www.microsoft.com/windowsazure/sqlazure/>.

# Setup

* 1. In order to execute the exercises in this hands-on lab you need to set up your environment.
  2. Open a Windows Explorer window and browse to the lab’s **Source** folder.
  3. Double-click the **Setup.cmd** file in this folder to launch the setup process that will configure your environment and install the Visual Studio code snippets for this lab.
  4. If the User Account Control dialog is shown, confirm the action to proceed.
  5. **Note:** Make sure you have checked all the dependencies for this lab before running the setup.

# Using the Code Snippets

* 1. Throughout the lab document, you will be instructed to insert code blocks. For your convenience, most of that code is provided as Visual Studio Code Snippets, which you can use from within Visual Studio 2010 to avoid having to add it manually.
  2. If you are not familiar with the Visual Studio Code Snippets, and want to learn how to use them, you can refer to the **Setup.docx** document in the **Assets** folder of the training kit, which contains a section describing how to use them.

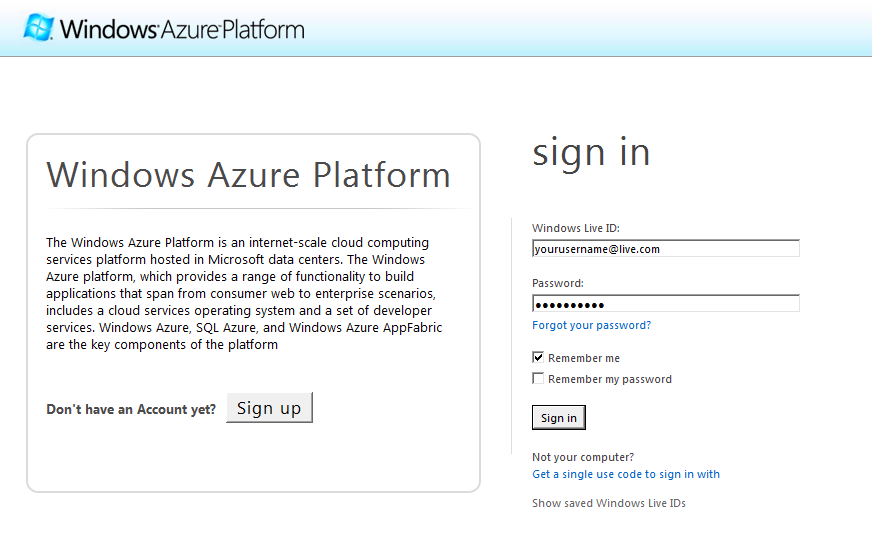
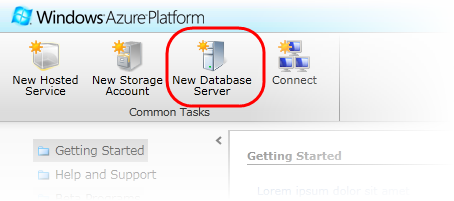
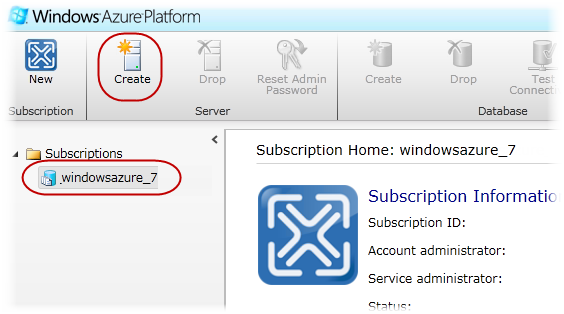
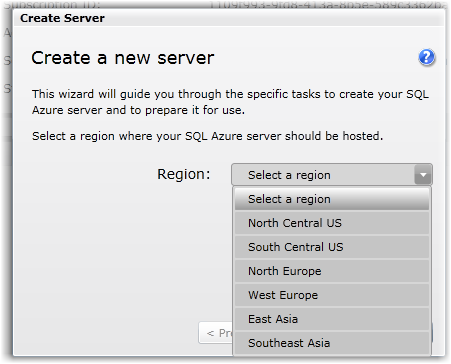
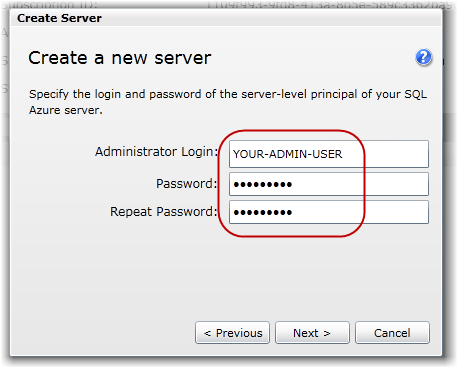
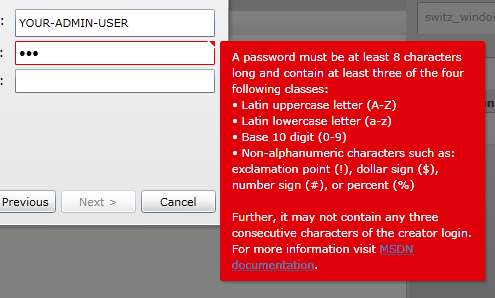
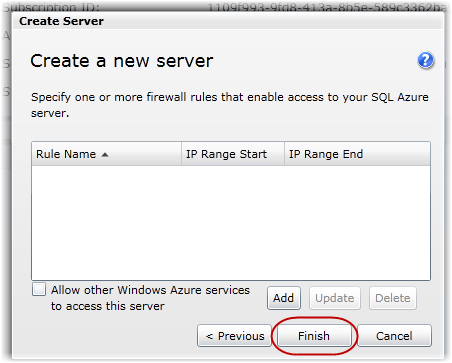
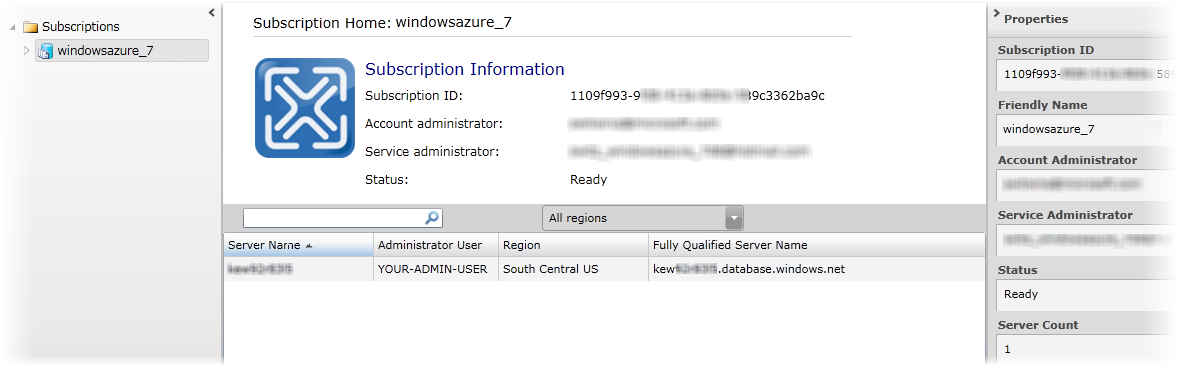
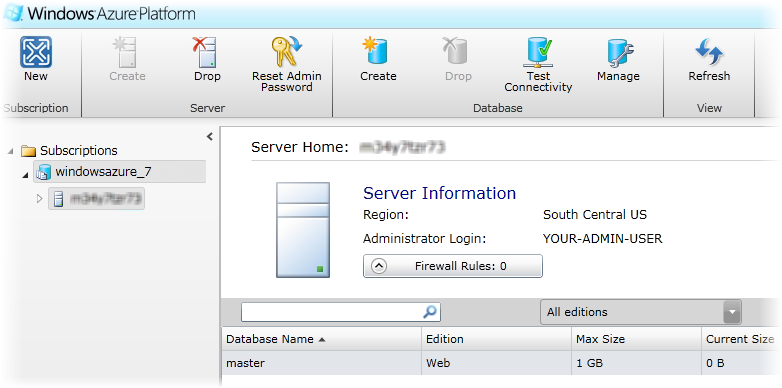
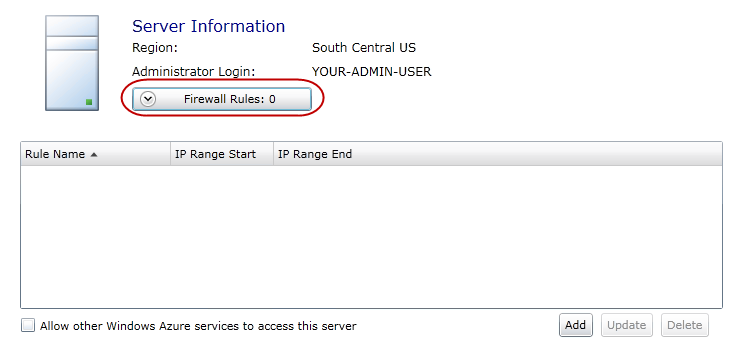
# Exercises

* 1. This hands-on lab includes the following exercises:
  2. Preparing Your SQL Azure Account
  3. Basic DDL and DML - Creating Tables and Indexes
  4. Build a Windows Azure Application that Accesses SQL Azure
  5. Connecting via Client Libraires
  6. Estimated time to complete this lab: **60 minutes**.
  7. **Note:** When you first start Visual Studio, you must select one of the predefined settings collections. Every predefined collection is designed to match a particular development style and determines window layouts, editor behavior, IntelliSense code snippets, and dialog box options. The procedures in this lab describe the actions necessary to accomplish a given task in Visual Studio when using the **General Development Settings** collection. If you choose a different settings collection for your development environment, there may be differences in these procedures that you need to take into account.

Exercise 1: Preparing Your SQL Azure Account

* 1. In this exercise, you will connect to your SQL Azure account and create a database server, add a new user and then reconnect to SQL Azure so that you can begin working with your new database.

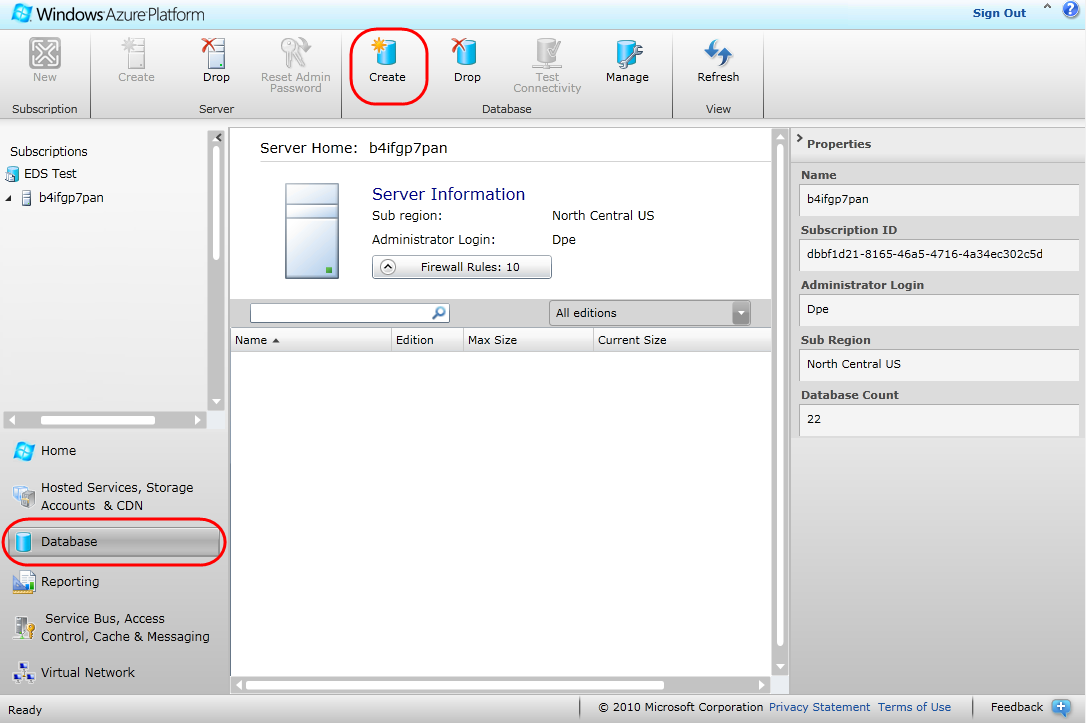
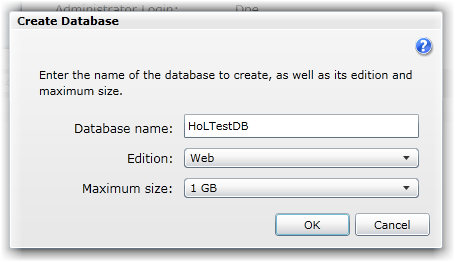
Task 1 – Retrieving your SQL Azure Server Name

* 1. In this exercise, you will log into the SQL Azure portal to obtain the name of the SQL Azure server assigned to your account.
  2. Go to the Windows Azure Platform portal at <https://windows.azure.com>.
  3. Log in to your Windows Live account.
     1. 
     2. Figure 1
     3. Logging into the Azure Services Portal
  4. At the Windows Azure Platform portal home page, click **New Database Server** on the ribbon.
     1. 
     2. Figure 2
     3. Creating a new SQL Azure database server
  5. If you have not previously created a server, you will need to do so now; otherwise, you may skip this exercise.
  6. To create a server, select your subscription on the left pane. Click on **Create** on the **Server** ribbon.
     1. 
     2. Figure 3
     3. Creating a new SQL Azure database server
  7. Select a region from the **Region** drop down list, and then click **Next**. The location determines which datacenter the database will reside in.
     1. 
     2. Figure 4
     3. Choosing region
  8. Enter an administrator account name and password and click **Next**.
     1. 
     2. Figure 5
     3. Entering administrator login and password
     4. **Note:** An administrator account is a master account used to manage the new server. You should avoid using this account in connection strings where the username and password may be exposed.
     5. The password policy requires that the password follows certain rules.
     6. 
  9. Click **Finish** to create the new server. You will configure firewall rules later on this exercise.
     1. 
     2. Figure 6
     3. Firewall rules dialog
  10. Finally, the new server information, including **Fully Qualified Server Name**, is shown.
      1. 
      2. Figure 7
      3. SQL Azure projects list
      4. **Note:** The fully qualified domain name of the server uses the following format:
      5. *<ServerName>.database.windows.net*
      6. where *<ServerName>* identifies the server, for example, *a9ixtp7pux.database.windows.net*.
  11. Expand the subscription node located on the left pane, and then click on the server name you have created. The **Server** **Information** page allows you to perform basic administration of the database server.
      1. 
      2. Figure 8
      3. SQL Azure server information page
  12. The **Firewall Rules** allows you to specify a list of IP addresses that can access your SQL Azure Server. The firewall will deny all connections by default, so **be sure to configure your allow list** so that existing clients can connect to the server.
      1. 
      2. Figure 9
      3. Configuring the firewall settings for SQL Azure
      4. **Note:** Changes to your firewall settings can take some time to become effective.
      5. You now have a database server created and ready for the next steps in this lab. This database can be connected to from anywhere in the world.

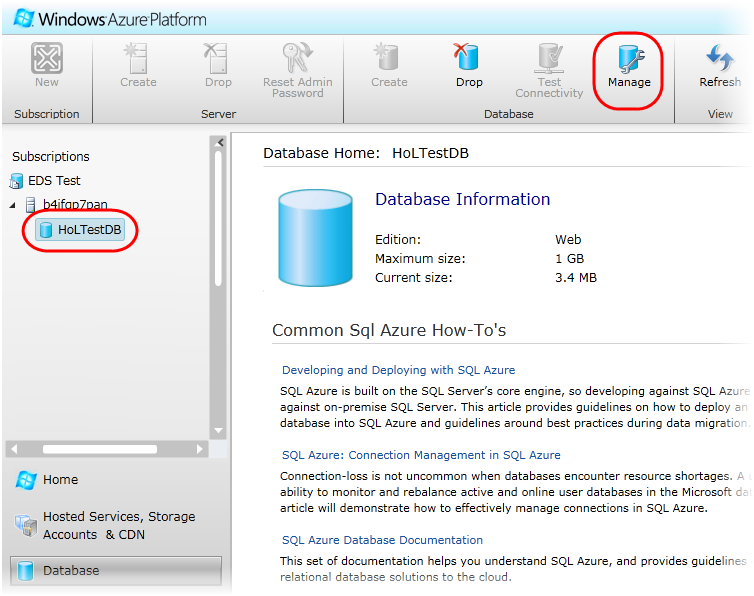
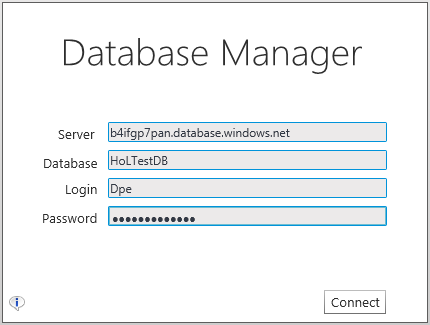
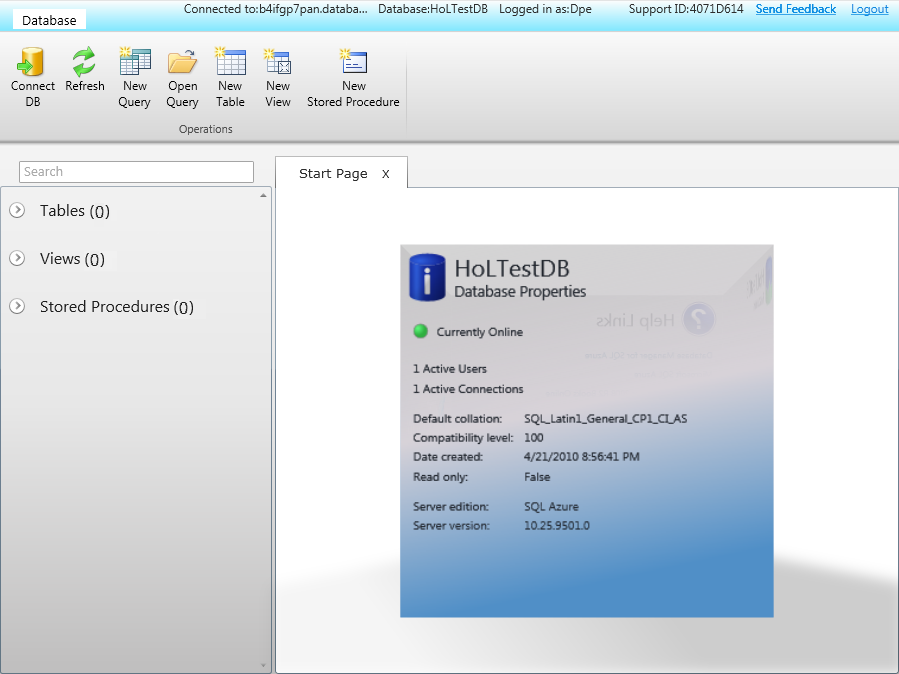
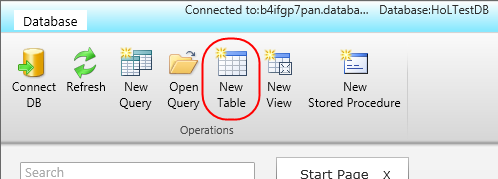
Exercise 2: Working with Data Basic DDL and DML

* 1. In this exercise, you will create a new database and work with its data. This means you will create some tables, index those tables appropriately, and then insert and query data. For this purpose, you will use two different tools. The first tool, the Database Manager for SQL Azure, is a browser based Silverlight database administration tool that you can access from the Windows Azure Platform portal. The other tool is SQL Server Management Studio, a tool normally associated with SQL Server management. You will see that this tool is equally useful for managing your SQL Azure databases.

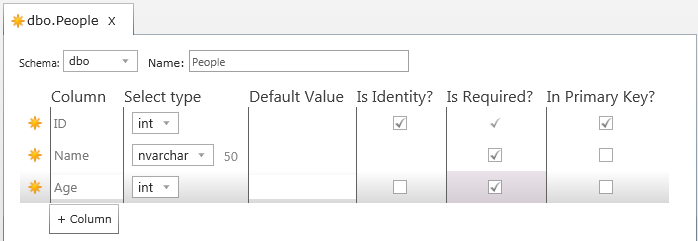
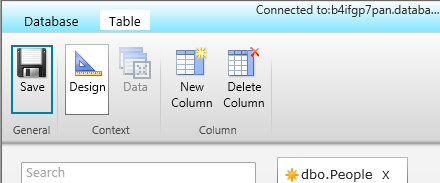
Task 1 – Creating a New Database

* 1. In the Windows Azure Management portal UI, select the **Database** option.
  2. Under **Subscriptions**, expand your project in the tree view on the left, select the server name where you wish to create a database, and then, in the **Database** group of the ribbon, click **Create**.
     1. 
     2. Figure 10
     3. Creating a new database
  3. In the **Create Database** dialog, set the **Database name** to *HoLTestDB*, select the *Web Edition* and set the **Maximum size** to *1 GB*.
     1. 
     2. Figure 11
     3. Choosing database features
     4. **Note:** In this hands-on lab, you create a database using the SQL Azure portal. Databases can also be created by executing a DDL query against your assigned server using the T-SQL CREATE DATABASE statement, specifying which SQL Azure database edition (Web or Business) to create as well as its maximum size. For example, to create a Business Edition database with a maximum size of 30GB, use the following T-SQL command:
     5. **CREATE DATABASE HolTestDB (MAXSIZE = 30GB)**
     6. Once a database reaches its maximum size, you cannot insert additional data until you delete some data to free storage space or increase its maximum size.

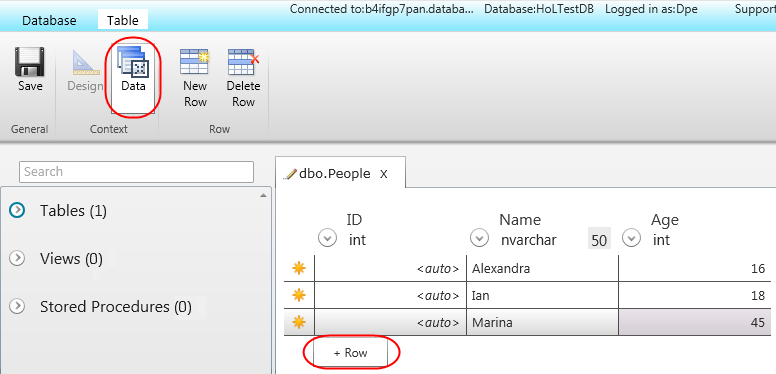
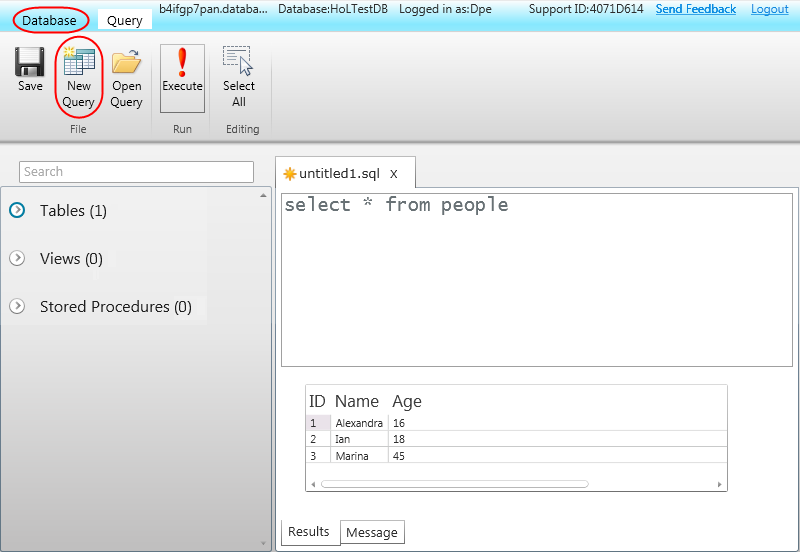
Task 2 – Managing your Database with the Database Manager for SQL Azure

* 1. In this task, you use the Database Manager for SQL Azure, a Silverlight client that runs in your browser, to connect to your SQL Azure database, create and populate a table, and then query its contents.
  2. Expand the server node under your subscription, click the *HoLTestDB* database to select it, and then click **Manage** on the ribbon.
     1. 
     2. Figure 12
     3. Managing a database
  3. In the database manager for SQL Azure sign-in page, enter your password and click **Connect**.
     1. 
     2. Figure 13
     3. Signing in to the SQL Azure database manager
  4. Wait for the manager to connect to your database and show its start page.
     1. 
     2. Figure 14
     3. Database manager start page
  5. In the ribbon, in the **Operations** group, click **New Table**.
     1. 
     2. Figure 15
     3. Creating a new table
  6. In the table creation UI, set the **Name** of the table to *People*.
  7. Next, define three table columns using the information shown below. Click **+ Column** to add new columns as needed.

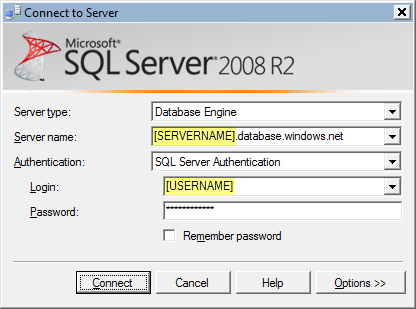
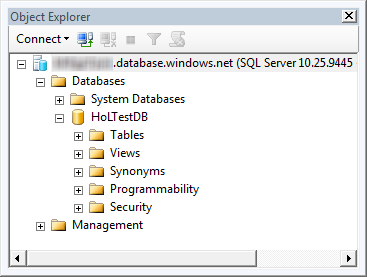
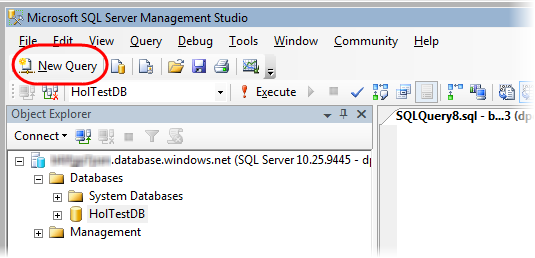
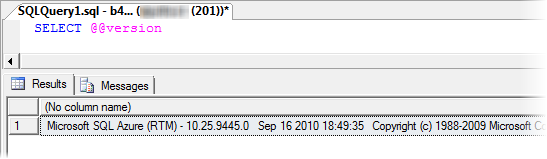
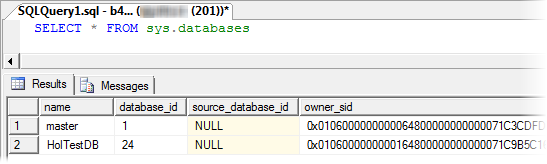
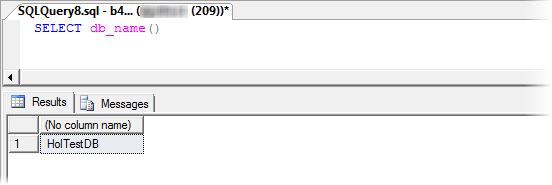
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Column | Type | Is Identity? | Is Required? | Is Primary Key? |
| ID | Int | Yes | Yes | Yes |
| Name | nvarchar(50) | No | Yes | No |
| Age | Int | No | Yes | No |

* + 1. 
    2. Figure 16
    3. Defining the table schema
  1. In the ribbon, in the **Operations** group, click **Save**.
     1. 
     2. Figure 17
     3. Saving the table schema
  2. Once the table is saved, in the **Context** group of the ribbon, click **Data**.
  3. Now, click **+ Row** and enter sample data for the *Name* and *Age* columns.

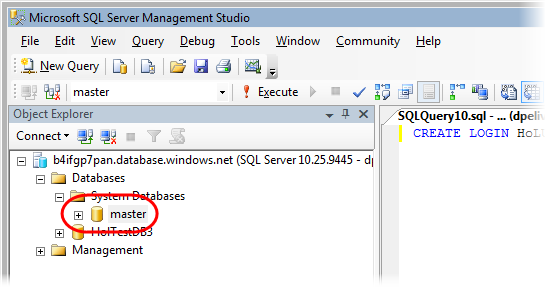
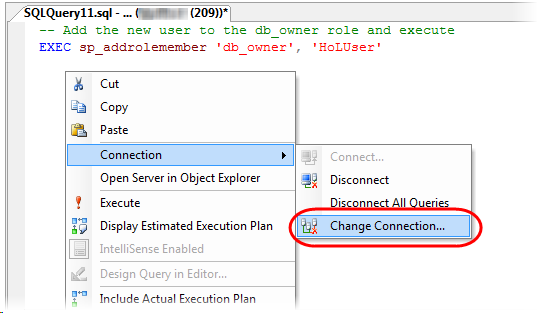
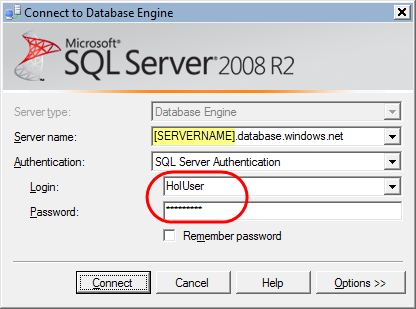
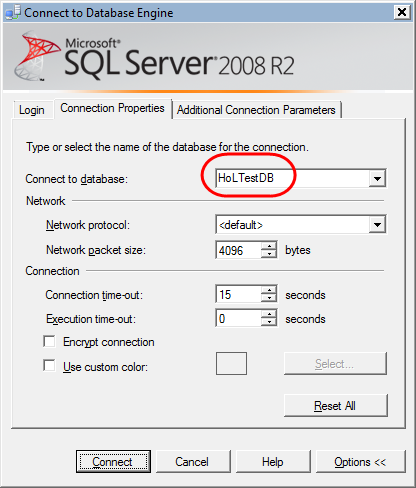
|  |  |
| --- | --- |
| Name | Age |
| Alexandra | 16 |
| Ian | 18 |
| Marina | 45 |

* + 1. 
    2. Figure 18
    3. Adding rows to the table
  1. Repeat the previous step to add another two rows and then click **Save** in the **Operations** group of the ribbon to commit the data to the table.
  2. Next, select the **Database** context in the upper left corner of the page and then click **New Query** in the **Operations** group of the ribbon.
  3. In the query window, enter the following T-SQL statement to select all the rows in the *People* table and then click **Execute** in the **Run** group of the ribbon. Verify that the results grid shows the rows that you entered previously.
     1. T-SQL
     2. **select \* from People**
     3. 
     4. Figure 19
     5. Querying the database

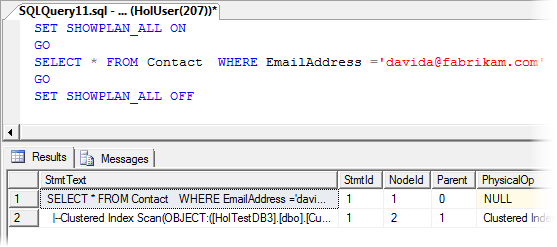
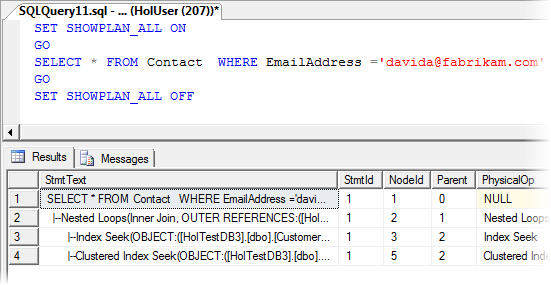
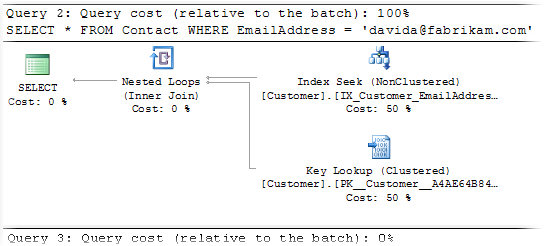
Task 3 – Managing your Database with SQL Server Management Studio

* 1. In this task, you use SQL Server Management Studio, a tool typically used for managing SQL Server, to connect to your SQL Azure server and administer it.
  2. Open SQL Server Management Studio from **Start | All Programs | Microsoft SQL Server 2008 R2 | SQL Server Management Studio**. You will be presented with a logon dialog.
  3. In the **Connect to Server** dialog, enter your login information ensuring that you select ***SQL Server Authentication***. SQL Azure currently only supports SQL Server Authentication.
     1. **Note:** Please replace server name with your server (e.g. REPLACE\_SERVER\_NAME.database.windows.net.)
     2. 
     3. Figure 20
     4. Connecting to SQL Azure with SQL Server Management Studio
  4. Click **Connect**.
  5. You should now see in your **Object Explorer** the structure of your database. Notice that your SQL Azure database is no different to an on-premise relational database.
     1. 
     2. Figure 21
     3. Object Explorer showing the HoLTestDB database
  6. In Object Explorer, select the **HoLTestDB** database in the tree view and then click **New Query** on the toolbar.
     1. 
     2. Figure 22
     3. Creating a new query window
  7. You now have a query window with an active connection to your account. You can test your connection by display the result of the **@@version** scalar function. To do this, type the following statement into the query window and press the **Execute** button. You will get back a scalar result that indicates the edition as Microsoft SQL Azure.
     1. T-SQL
     2. SELECT @@version
     3. 
     4. Figure 23
     5. Retrieving the SQL Azure version
  8. Replace the previous query with the statement shown belowand click **Execute.** Notice that the results grid shows the databases currently accessible.
     1. T-SQL
     2. SELECT \* FROM sys.databases
     3. 
     4. Figure 24
     5. Query results showing the list of databases in your subscription
  9. You can check that you are now in the context of your user database by executing the following query. Make sure that you replace the previous query.
     1. T-SQL
     2. SELECT db\_name()
     3. 
     4. Figure 25
     5. Querying the database currently in use
  10. Do not close the query window. You will need it during the next task.

Task 4 – Creating Logins and Database Users

* 1. Much like SQL Server, SQL Azure allows you to create additional logins and then assign those logins as users with permissions on a database. In this task, you will create a new login and then create a user that uses the new login in your *HoLTestDB* database.
  2. Open a new query window connected to the *master* database. To do this, in **Object Explorer**, expand the **System Databases** node inside **Databases** and then select *master*. Then, click **New Query** on the toolbar.
     1. 
     2. Figure 26
     3. Querying the master database
     4. **Note:** You cannot reuse the previous query window connected to the *HoLTestDB* database because you cannot change the database context without closing the current connection. The USE <database\_name> command does not work with SQL Azure. Therefore, you need to open a new query window or disconnect and reconnect in order to change from the *HoLTestDB* to the *master* database.
  3. Create a new login by executing the following statement:
     1. T-SQL
     2. **CREATE LOGIN HoLUser WITH password='Password1'**
     3. **Note:** You should choose your own password for this login account and use it where appropriate throughout the lab. If you do not choose a unique password, you should ensure that you remove this login when you finish the lab. To do this, execute the following statement in the *master* database:
     4. **DROP LOGIN HoLUser**
  4. Go back to the query window connected to the *HoLTestDB* database. If you closed this window, open it again by selecting the *HoLTestDB* database in **Object Explorer** and then click **New Query**.
  5. In the query window, execute the following statement to create a new user in the HoLTestDB database for the login *HoLUser*.
     1. T-SQL
     2. **-- Create a new user from the login and execute**
     3. **CREATE USER HoLUser FROM LOGIN HoLUser**
  6. Next, add the user to the **db\_owner** role of your *HoLTestDB* database by executing the following:
     1. T-SQL
     2. **-- Add the new user to the db\_owner role and execute**
     3. **EXEC sp\_addrolemember 'db\_owner', 'HoLUser'**
     4. **Note:** By making your user a member of the **db\_owner** role, you have granted a very extensive permission set to the user. In a real world scenario, you should be careful to ensure that you grant users only the smallest privilege set possible.
  7. Change the user associated with the current connection to the newly created *HoLUser*. To do this, right-click the query window, point to **Connection**, and then select **Change Connection**.
     1. 
     2. Figure 27
     3. Changing the database connection properties
  8. In the **Connect to Database Engine** dialog, replace the **Login** name with *HoLUser* and set the **Password** to the value that you chose earlier when you created the database user.
     1. 
     2. Figure 28
     3. Connecting to the database as a different user
  9. Click **Options** to show additional connection settings. Switch to the **Connection Properties** tab and ensure that the name of the database for the connection is *HoLTestDB*. If the current value is different, you will need to type this rather than use the drop down list, then press the **Connect** button.
     1. 
     2. Figure 29
     3. Connecting to a specific database
     4. **Note:** You are now connected to the database as the *HoLUser* database user. You will continue with this user for the remaining steps of this exercise.

Task 5 – Creating Tables, Indices, and Queries

* 1. In the query window, replace the current content with the following SQL query to create a *Contact* table and execute it.
     1. T-SQL
     2. **CREATE TABLE [Contact](**
     3. **[ContactID] [int] IDENTITY(1,1) NOT NULL PRIMARY KEY CLUSTERED,**
     4. **[Title] [nvarchar](8) NULL,**
     5. **[FirstName] [nvarchar](50) NOT NULL,**
     6. **[LastName] [nvarchar](50) NOT NULL,**
     7. **[EmailAddress] [nvarchar](50) NULL,**
     8. **[Phone] [nvarchar](30) NULL,**
     9. **[Timestamp] [timestamp] NOT NULL**
     10. **)**
     11. **Note:** SQL Azure requires that every table have a clustered index. If you create a table without a clustered index, you will not be able to insert rows into the table until you have created one.
     12. Because the clustered index determines the order of rows on disk, and thus affects certain queries, you may choose to place the clustered index on a column other than the primary key column.
  2. You will add an index on the *EmailAddress* field. To do this, execute the following query:
     1. T-SQL
     2. **CREATE INDEX IX\_Contact\_EmailAddress**
     3. **ON Contact(EmailAddress)**
  3. Execute the following query to add a row to the new *Contact* table:
     1. T-SQL
     2. **INSERT INTO [Contact]**
     3. **([Title],[FirstName],[LastName],[EmailAddress],[Phone])**
     4. **VALUES**
     5. **('Mr','David','Alexander','davida@fabrikam.com','555-1234-5555')**
  4. Now, query the data back out, but start by enabling the SHOWPLAN*\_ALL* option to show the execution plan. To do that, execute the following query:
     1. T-SQL
     2. **SET SHOWPLAN\_ALL ON**
     3. **GO**
     4. **SELECT \* FROM Contact WHERE EmailAddress ='davida@fabrikam.com'**
     5. **GO**
     6. **SET SHOWPLAN\_ALL OFF**
     7. 
     8. Figure 30
     9. Query execution plan for a small set
  5. Next, you will add a large number of rows to the database and then examine the query plan again. To do this, execute the following query to create a stored procedure named *AddData*. This stored procedure will loop incrementing a counter each time through and add a new record with an email address with the pattern [Counter][davida@fabrikam.com](mailto:davida@fabrikam.com):
     1. T-SQL
     2. **CREATE PROCEDURE AddData**
     3. **@NumRows int**
     4. **AS**
     5. **DECLARE @counter int**
     6. **SELECT @counter = 1**
     7. **WHILE (@counter < @NumRows)**
     8. **BEGIN**
     9. **INSERT INTO [Contact]**
     10. **([Title],[FirstName],[LastName],[EmailAddress],[Phone])**
     11. **VALUES**
     12. **('Mr','David','Alexander',CAST(@counter as nvarchar)+'davida@fabrikam.com','555-1234-5555')**
     13. **SELECT @counter = @counter + 1**
     14. **END**
  6. Now, insert 10,000 rows into the *Contact* table by executing the new stored procedure. Each row will have a unique email address. To do this, execute the following T-SQL statement:
     1. T-SQL
     2. **EXEC AddData 10000**
     3. **Note:** It may take some time to generate the 10,000 rows.
  7. Execute the following query again and examine the generated plan. Compare the result with the one obtained previously, when the table contained few rows.
     1. T-SQL
     2. **SET SHOWPLAN\_ALL ON**
     3. **GO**
     4. **SELECT \* FROM Contact WHERE EmailAddress ='davida@fabrikam.com'**
     5. **GO**
     6. **SET SHOWPLAN\_ALL OFF**
     7. 
     8. Figure 31
     9. Query execution plan for a large set
     10. **Note:** Notice that the second time around the query optimizer is likely to use the index that you defined: This is the Index Seek line in the query plan.
  8. For the most part, you can use any tool that you might have previously used with SQL Server on-premise. For an example of this, look at the query plan graphically. In SQL Server Management Studio press **Ctrl-L** to display the **Estimated Execution Plan**.
     1. 
     2. Figure 32
     3. Showing the query execution plan graphically

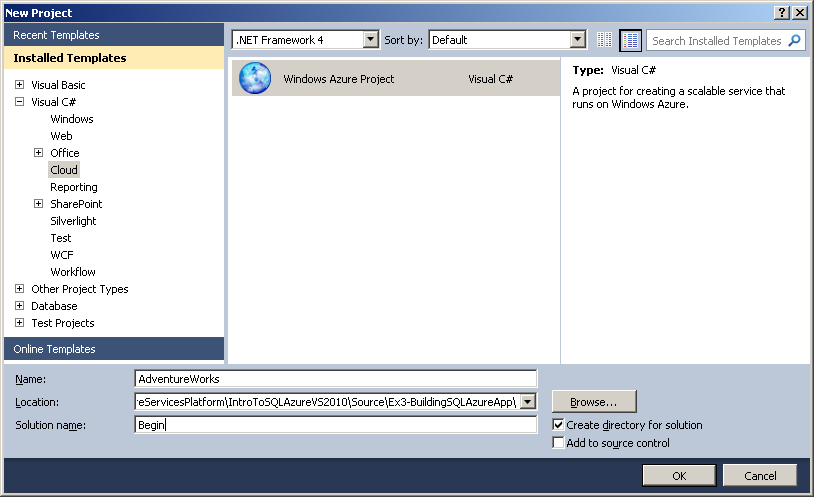
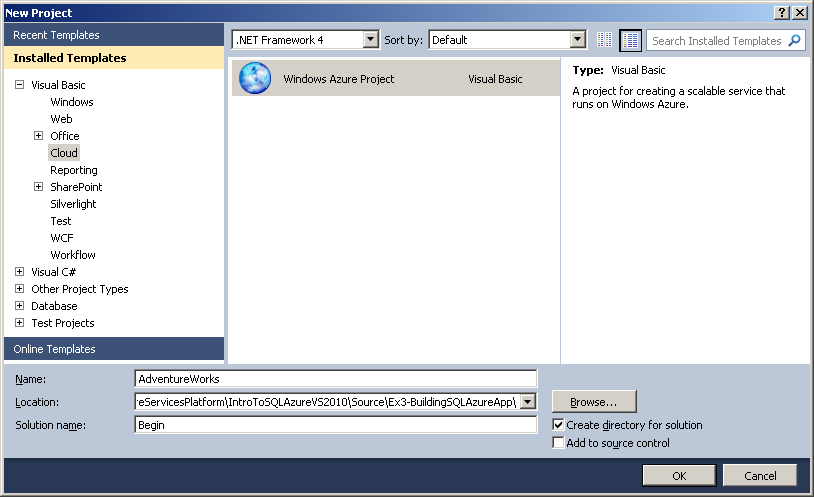
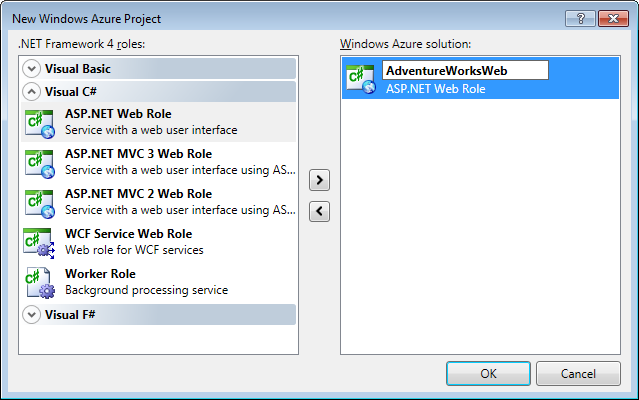
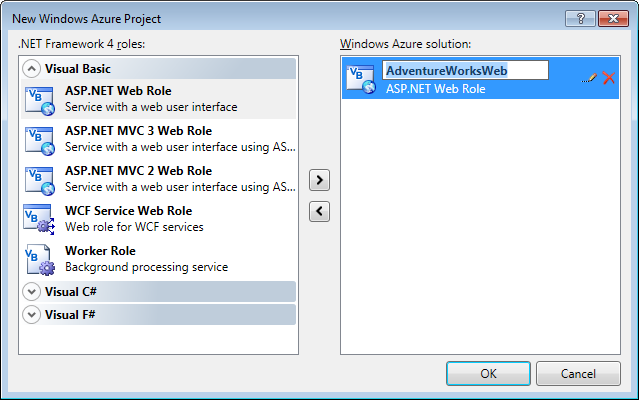
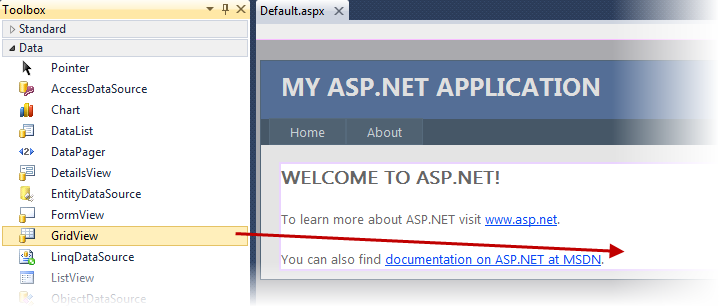
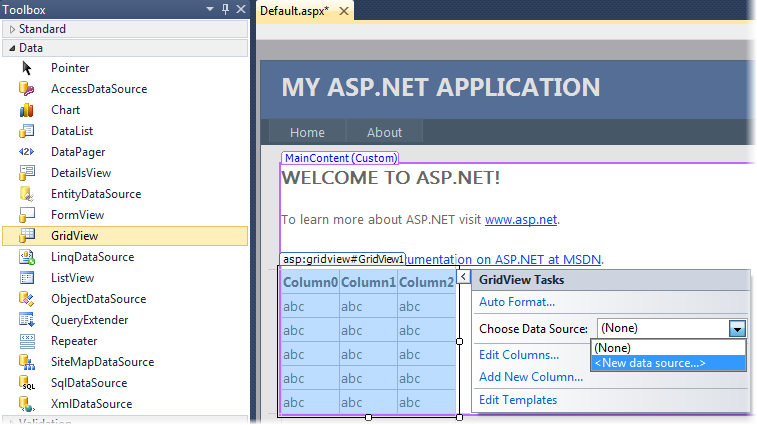
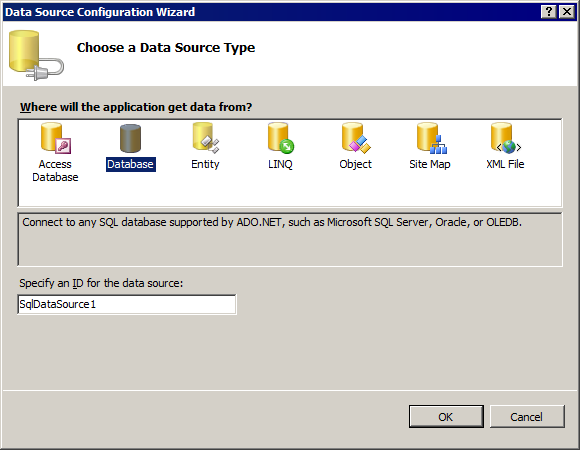
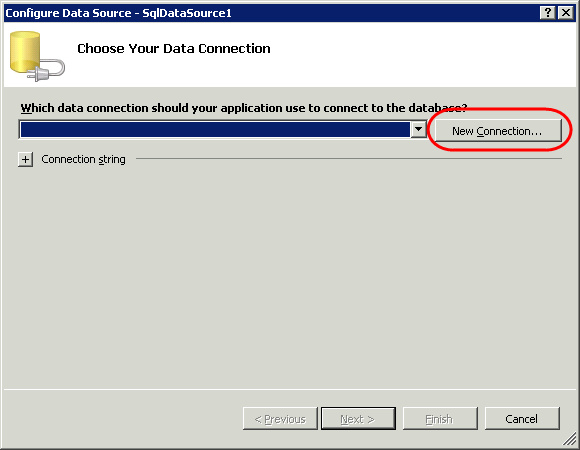
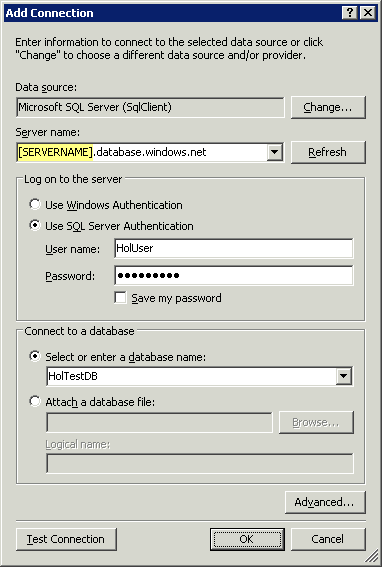
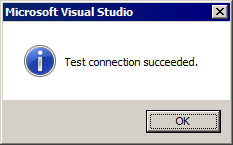
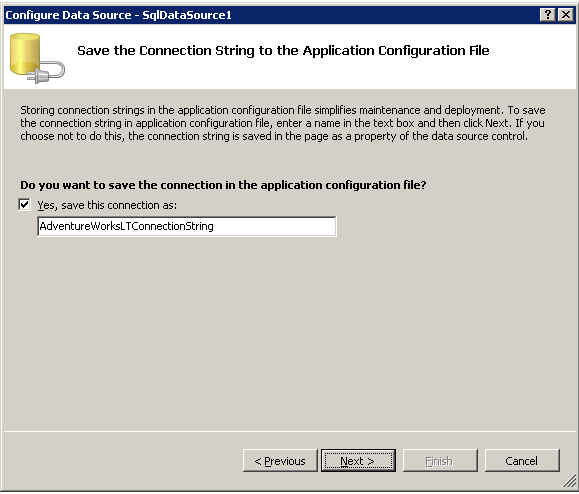
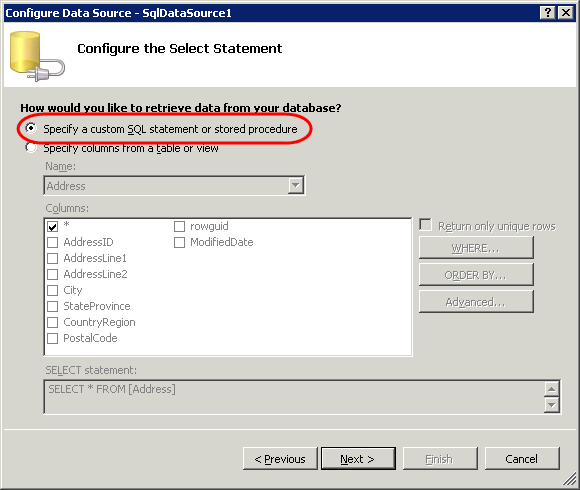
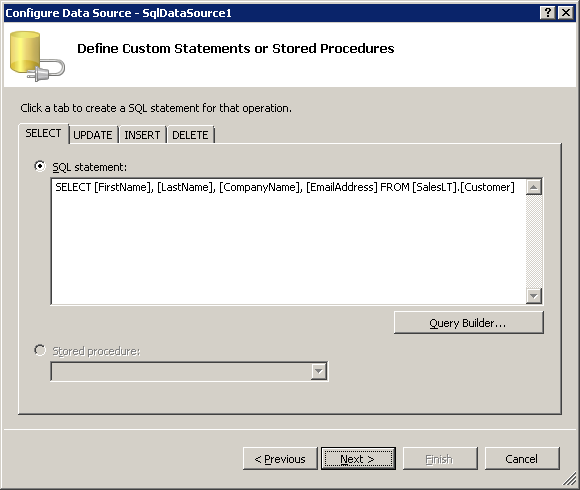
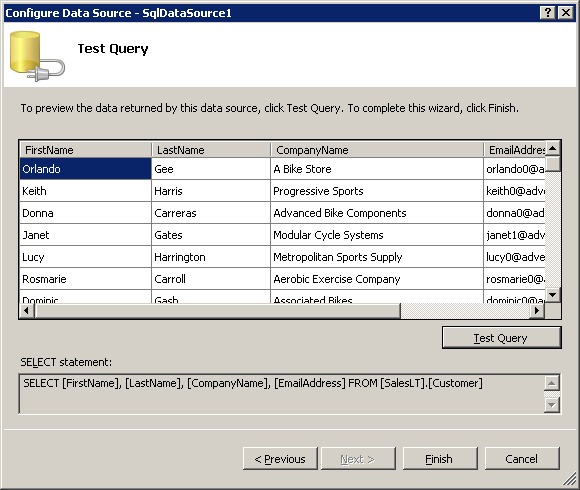
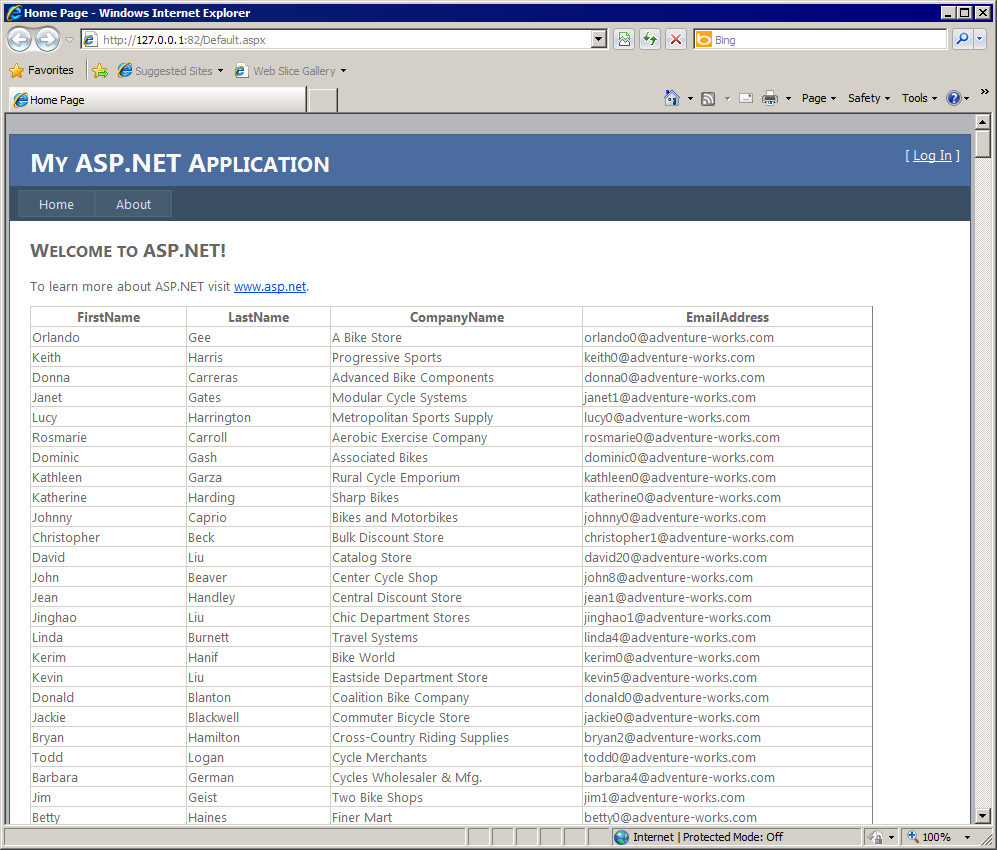
Exercise 3: Build a Windows Azure Application that Accesses SQL Azure

1. In this exercise, you will create a simple Windows Azure application to manipulate the data in the *Customer* table of the *AdventureWorksLT2008* database.
   1. The purpose of this exercise is to demonstrate just how simple it is to work with SQL Azure and Windows Azure using the graphical Visual Studio 'drag and drop' approach.

Task 1 – Loading the Sample Database into SQL Azure

* 1. Connect to the *HoLTestDB* database on your SQL Azure server using the *HoLUser* login that you created in the previous exercises. You can use either SQL Server Management Studio or the Database Manager for SQL Azure to perform these steps.
  2. If you are using SQL Server Management Studio, in the **File** menu, select **Open** | **File** and then navigate to the **Assets** folder inside the **Source** folder of this lab; if you use the Database Manager for SQL Azure, select the **Database** context on the upper left corner and click the **Open Query** button in the **File** group. Select the **AdventureWorks2008LT\_Azure.sql** file and click **Open**.
     1. **Note:** This script contains a cleaned up export script from the *AdventureWorksLT2008* sample database available for download in the [Codeplex sample databases project site](http://msftdbprodsamples.codeplex.com/). SQL Azure sample databases will be available for downloading that you could use to replace this script file.
  3. Execute the query. This may take a few minutes, as you are creating a subset of the Adventure Works database.

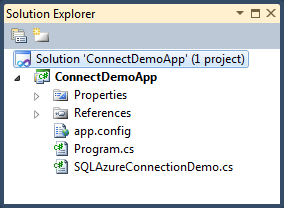
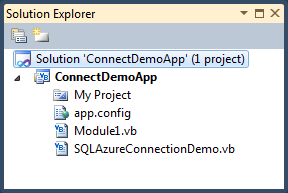
Task 2 – Creating the Visual Studio Project

* 1. In this task, you create a new Visual Studio project for a Windows Azure Web Site.
  2. Open Microsoft Visual Studio 2010 in elevated administrator mode. To do this, in **Start** | **All Programs** | **Microsoft Visual Studio 2010**, right-click the **Microsoft Visual Studio 2010** shortcut and choose **Run as Administrator**.
  3. If the **User Account Control** dialog appears, click **Continue**.
  4. From the **File** menu, choose **New** and then **Project**.
  5. In the **New Project** dialog, expand the language of your preference (Visual C# or Visual Basic) in the **Installed Templates** list and select **Cloud**.
  6. In the **Templates** list, select **Windows Azure** **Project**. Set the name of the project to “**AdventureWorks**” and the location inside **Ex3-BuildingSQLAzureApp** in the **Source** folder of the lab. Ensure that **Create directory for solution** is checked and then set the name of the solution to “**Begin**”. Click **OK** to create the project.
     1. 
     2. Figure 33
     3. Creating a new Web Cloud Service (C#)
     4. 
     5. Figure 34
     6. Creating a new Web Cloud Service (Visual Basic)
  7. In the **New Windows Azure Project** dialog, inside the **Roles** panel, expand the tab for the language of your choice (Visual C# or Visual Basic), select **ASP.NET Web Role** from the list of available roles and click the right arrow (>) to add an instance of this role to the solution. Before closing the dialog, select the new role in the right panel, click the pencil icon and rename the role as **AdventureWorksWeb**. Click **OK** to create the cloud service solution.
     1. 
     2. Figure 35
     3. Adding a Web Role to the Solution (C#)
     4. 
     5. Figure 36
     6. Adding a Web Role to the Solution (VB)
  8. When the project template has finished creating items, you should be presented with the **Default.aspx** page. If not, openthis file.
  9. Ensure that you are viewing the Default.aspx page in Design View by clicking the **Design** button.
  10. Drag and drop a **GridView** control from the *Data* section of the Toolbox onto the design canvas.
      1. 
      2. Figure 37
      3. Adding a GridView control
  11. From the *SmartTag* on the upper right corner of the newly created GridView, choose the ***New data source*** option on the **Choose Data Source** combo box.
      1. 
      2. Figure 38
      3. Creating a new data source
  12. In the **Data Source Configuration Wizard**, choose a data source type of **Database** and leave the default ID. Click **OK**.
      1. 
      2. Figure 39
      3. Choosing a Data Source
  13. In the **Configure Data Source** dialog, click **New Connection**.
      1. 
      2. Figure 40
      3. Creating a new Connection
  14. If prompted by a **Choose data source** dialog, select **Microsoft SQL Server** and click **Continue**.
  15. Now, configure a connection to your SQL Azure database. In the **Add Connection**dialog, ensure your provider is **Microsoft SQL Server (SqlClient)** selecting **Microsoft SQL Server** inside **Data Source** list and **.NET Framework Data Provider for SQL Server** in the Data Provider combo. Then set the **Server name** to the name of the server for your SQL Azure subscription. Next, change the authentication type to **Use SQL Server Authentication** and type the credentials for your SQL Azure subscription. Finally, enter *HoLTestDB* in the database name drop down list.
      1. 
      2. Figure 41
      3. Configuring a connection to the HolTestDB database in SQL Azure
  16. Press **Test Connection**. If the connection information is correct,you should receive a dialog indicating success. Click **OK** to proceed.
      1. 
      2. Figure 42
      3. Confirmation of a successful connection
  17. Click **OK** to close the **Add Connection**dialog.
  18. Click **Next** to proceed with the **Data Source Configuration Wizard**.
  19. Ensure that the option labeled **Yes, save this connection as** ischecked, set the name of the connection to **AdventureWorksLTConnectionString**, and then click **Next**.
      1. 
      2. Figure 43
      3. Saving the connection string in the application configuration file
  20. Select the option labeled **Specify a custom SQL statement or stored procedure** and then click **Next**.
      1. 
      2. Figure 44
      3. Using a custom SQL statement to query the database
      4. **Note:** You cannot use the **Specify columns from a table or view** option because *AdventureWorks* uses a named Schema (SalesLT) that you need to explicitly reference.
  21. Paste the following statement into the **SQL Statement** box and click **Next**.
      1. T-SQL
      2. SELECT [FirstName], [LastName], [CompanyName], [EmailAddress] FROM [SalesLT].[Customer]
      3. 
      4. Figure 45
      5. Defining a custom SQL statement
  22. Press **Test Query** and you should see results returned.
      1. 
      2. Figure 46
      3. Testing the query against the database
  23. Click **Finish**.
  24. Press **F5** to run the application in the compute emulator.
  25. The application will execute and you will see the list of all customers in the browser:
      1. 
      2. Figure 47
      3. Retrieving a list of customers from the database
  26. Close the browser window.

Exercise 4: Connecting via Client Libraries

* 1. In this exercise, you will learn how to use ADO.NET, ODBC, OLEDB and LINQ to SQL technologies to connect to your SQL Azure database and perform some simple T-SQL operations. In addition, you will see how to connect to the database from other technologies like Java and PHP.
  2. Using Microsoft Technologies, you will see that the way in which you interact with your SQL Azure database from your applications is the same as a traditional SQL database. The main differences between the technologies lie in the type of connection and the connection strings used to connect to SQL Azure. After the connection is established, you can then use the appropriate inheritor of the ‘DbCommand’ to issue your commands to SQL Azure.

Task 1 – Opening the Begin Solution and Exploring the Common Functionalities

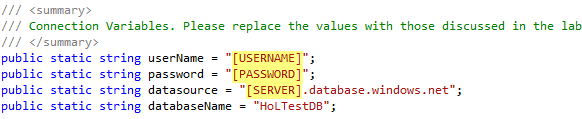
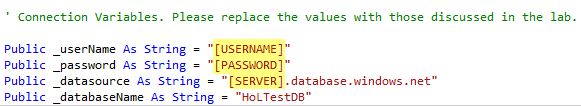
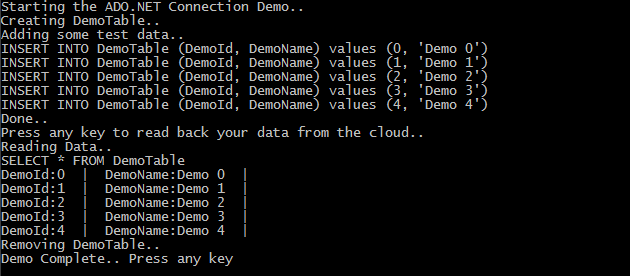
* 1. You will test the different Microsoft technologies connecting to SQL Azure and performing some tasks against a new table. To avoid spending time implementing logic that creates, inserts, queries and deletes a table, this exercise provides a begin solution that implements these common functionalities. This allows you to focus on learning how to connect to SQL Azure and explore the differences between the proposed technologies.
  2. In this task, you will open the **ConnectDemoApp** solution and explore the **SQLAzureConnectionDemo** class. During the exercise, you will inherit from this class for each different implementation of a data access technology.
  3. Open **Microsoft Visual Studio 2010** from **Start** | **All Programs** | **Microsoft Visual Studio 2010** | **Microsoft Visual Studio 2010**.
  4. Open the begin solution provided for this exercise. To do this, from the **File** menu, choose **Open Project**. In the **Open Project** dialog, navigate to **Ex4-ConnectingViaClientLibraries\begin** inside the **Source** folder of this lab. Select the folder for the language of your preference (C# or VB), and then open the solution **ConnectDemoApp.sln** inside the **ConnectDemoApp** folder. A solution with the following structure should open.
     1. 
     2. Figure 48
     3. Connect Demo App solution’s structure (C#)
     4. 
     5. Figure 49
     6. Connect Demo App solution’s structure (Visual Basic)
  5. As mentioned before, you will create a class per technology inheriting from the **SQLAzureConnectionDemo** abstract class. This class provides common functionality to perform basic operations against SQL Azure using the provider that you implement in the derived class. The table below explains each of the methods in this class to understand how it works and determine which methods you need to implement in the derived classes:

|  |  |  |
| --- | --- | --- |
| Method | Type | Description |
| Class Constructor |  | Sets the connection property based on the result of the CreateConnection abstract method that will be implemented on the derived class. |
| CreateConnection | Abstract | A derived class implements this method in order to create the connection according to the underlying technology. |
| CreateCommand | Abstract | A derived class implements this method to create a command according to the underlying technology. |
| GetServerName |  | Returns the server name from the data source. It is a common task required to create the connection to the database. |
| ConnectToSQLAzureDemo |  | Executes the demo flow against the SQL Azure Database. It gets a command from the derived class using the CreateCommand method and then executes the Execute\* methods to create, fill, query and delete a demo table. |
| ExecuteCreateDemoTableStatement |  | Executes a create table statement to create the “DemoTable” table. |
| ExecuteInsertTestDataStatement |  | Executes an insert statement against the “DemoTable” table. |
| ExecuteReadInsertedTestData |  | Executes a select statement trying to retrieve the data inserted by the previous method and calls the ReadData method to show it in the Console. |
| ReadData |  | Reads the data retrieved from the table and displays it in the Console. |
| ExecuteDropDemoTable |  | Executes a delete statement removing the “DemoTable” table from the SQL Azure database. |

* + 1. Notice that you will only have to override the **CreateConnection** and **CreateCommand** methods on the implementation of each technology to create a connection to SQL Azure successfully.

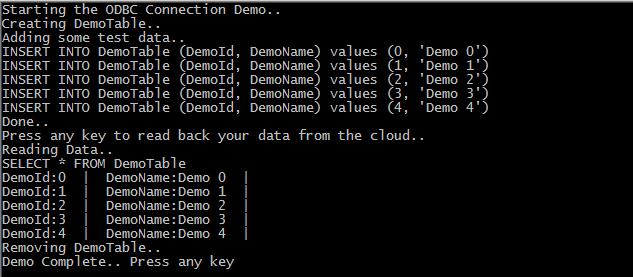
Task 2 – Connecting to SQL Azure Using ADO.NET

In this task, you will create a class that inherits from the **SQLAzureConnectionDemo** class and implements the methods to connect to SQL Azure using ADO.NET.

* 1. Add a new class to the project named **AdoConnectionDemo**. To do this, right-click the **ConnectDemoApp** project in **Solution Explorer** and select **Add** | **Class**. In the **Add New Item** dialog, make sure that you select the **Class** template and set the name to **AdoConnectionDemo.cs** or **AdoConnectionDemo.vb** depending on the language of your project**.**
  2. Make sure that you have the following namespace directives at the top of the file:
     1. C#
     2. **using System.Data.Common;**
     3. **using System.Data.SqlClient;**
     4. Visual Basic
     5. **Imports System.Data.Common**
     6. **Imports System.Data.SqlClient**
  3. Update the class definition to make it public and to inherit from **SQLAzureConnectionDemo**. The final implementation should look like the following:
     1. **Note:** In Visual Basic, the template for a new class already declares the class as Public.
     2. C#
     3. **public** class AdoConnectionDemo **:** SQLAzureConnectionDemo
     4. {
     5. }
     6. Visual Basic
     7. Public Class AdoConnectionDemo
     8. **Inherits** **SQLAzureConnectionDemo**
     9. End Class
  4. Implement the class constructor to retrieve the connection information and pass it as parameters to the base class constructor:
     1. (Code Snippet – *Intro to SQL Azure – Ex4 ADO constructor – C#*)
     2. C#
     3. **public AdoConnectionDemo(string userName, string password, string dataSource, string databaseName)**
     4. **: base(userName, password, dataSource, databaseName)**
     5. **{**
     6. **}**
     7. (Code Snippet – *Intro to SQL Azure – Ex4 ADO constructor – VB*)
     8. Visual Basic
     9. **Public Sub New(ByVal userName As String, ByVal password As String, ByVal dataSource As String, ByVal databaseName As String)**
     10. **MyBase.New(userName, password, dataSource, databaseName)**
     11. **End Sub**
  5. The **SQLAzureConnectionDemo** class delegates the connection construction to the derived class. Override the **CreateConnection** method to create a **SqlConnection** in your **AdoConnectionDemo** class:
     1. (Code Snippet – *Intro to SQL Azure – Ex4 ADO CreateConnection – C#*)
     2. C#
     3. **protected override DbConnection CreateConnection(string userName, string password, string dataSource, string databaseName)**
     4. **{**
     5. **return new SqlConnection(CreateAdoConnectionString(userName, password, dataSource, databaseName));**
     6. **}**
     7. (Code Snippet – *Intro to SQL Azure – Ex4 ADO CreateConnection – VB*)
     8. Visual Basic
     9. **Protected Overrides Function CreateConnection(ByVal userName As String, ByVal password As String, ByVal dataSource As String, ByVal databaseName As String) As DbConnection**
     10. **Return New SqlConnection(CreateAdoConnectionString(userName, password, dataSource, databaseName))**
     11. **End Function**
  6. Implement the **CreateAdoConnectionString** method used by the **CreateConnection** method. This method is responsible for building up the connection string for the ADO.NET Connection, which takes advantage of the **SqlConnectionStringBuilder** class in the underlying implementation.
     1. (Code Snippet – *Intro to SQL Azure – Ex4 ADO CreateAdoConnectionString method – C#*)
     2. C#
     3. **private string CreateAdoConnectionString(string userName, string password, string dataSource, string databaseName)**
     4. **{**
     5. **// create a new instance of the SQLConnectionStringBuilder**
     6. **SqlConnectionStringBuilder connectionStringBuilder = new SqlConnectionStringBuilder**
     7. **{**
     8. **DataSource = dataSource,**
     9. **InitialCatalog = databaseName,**
     10. **Encrypt = true,**
     11. **TrustServerCertificate = false,**
     12. **UserID = userName,**
     13. **Password = password,**
     14. **};**
     15. **return connectionStringBuilder.ToString();**
     16. **}**
     17. (Code Snippet – *Intro to SQL Azure – Ex4 ADO CreateAdoConnectionString method – VB*)
     18. Visual Basic
     19. **Private Function CreateAdoConnectionString(ByVal userName As String, ByVal password As String, ByVal dataSource As String, ByVal databaseName As String) As String**
     20. **' create a new instance of the SQLConnectionStringBuilder**
     21. **Dim connectionStringBuilder As SqlConnectionStringBuilder = New SqlConnectionStringBuilder With {.DataSource = dataSource, .InitialCatalog = databaseName, .Encrypt = True, .TrustServerCertificate = False, .UserID = userName, .Password = password}**
     22. **Return connectionStringBuilder.ToString()**
     23. **End Function**
  7. Override the **CreateCommand** method to create an ADO.NET command. Remember that this abstract method is called in the parent class to get the connection and execute the different SQL statement samples.
     1. (Code Snippet – *Intro to SQL Azure – Ex4 ADO CreateCommand method – C#*)
     2. C#
     3. **protected override DbCommand CreateCommand(DbConnection connection)**
     4. **{**
     5. **return new SqlCommand() { Connection = connection as SqlConnection };**
     6. **}**
     7. (Code Snippet – *Intro to SQL Azure – Ex4 ADO CreateCommand method – VB*)
     8. Visual Basic
     9. **Protected Overrides Function CreateCommand(ByVal connection As DbConnection) As DbCommand**
     10. **Return New SqlCommand() With {.Connection = TryCast(connection, SqlConnection)}**
     11. **End Function**
     12. That is all the code required to use an ADO.NET connection. Now you will include some code on the **Program.cs** or **Module1.vb** file (depending on the language of your project) to test the connection and see how the different operations work.
  8. Open the **Program.cs** (for Visual C# projects) or **Module1.vb** (for Visual Basic projects) file double-clicking it in the **Solution Explorer** inside the **ConnectDemoApp** project.
  9. Implement the logic to create an instance of the **AdoConnectionDemo** class and execute the demo against SQL Azure.
     1. (Code Snippet – *Intro to SQL Azure – Ex4 ADO demo implementation – C#*)
     2. C#
     3. static void Main(string[] args)
     4. {
     5. **//Invoke the ADO.NET connection demo**
     6. **Console.WriteLine("Starting the ADO.NET Connection Demo...");**
     7. **AdoConnectionDemo demo1 = new AdoConnectionDemo(userName, password, datasource, databaseName);**
     8. **demo1.ConnectToSQLAzureDemo();**
     9. **Console.WriteLine("Demo Complete... Press any key");**
     10. **Console.ReadKey();**
     11. }
     12. (Code Snippet – *Intro to SQL Azure – Ex4 ADO demo implementation – VB*)
     13. Visual Basic
     14. Sub Main()
     15. **' Invoke the ADO.NET connection demo**
     16. **Console.WriteLine("Starting the ADO.NET Connection Demo...")**
     17. **Dim demo1 = New AdoConnectionDemo(\_userName, \_password, \_datasource, \_databaseName)**
     18. **demo1.ConnectToSQLAzureDemo()**
     19. **Console.WriteLine("Demo Complete... Press any key")**
     20. **Console.ReadKey()**
     21. End Sub
  10. Locate the member variables declared immediately above method **Main** and update the placeholders with the connection information for your SQL Azure account.
      1. 
      2. Figure 50
      3. Configuring connection parameters for SQL Azure (C#)
      4. 
      5. Figure 51
      6. Configuring connection parameters for SQL Azure (Visual Basic)
  11. Run the application by pressing **F5**. You should see the following output in a console window.
      1. 
      2. Figure 52
      3. Expected output from the ADO.NET connection demo

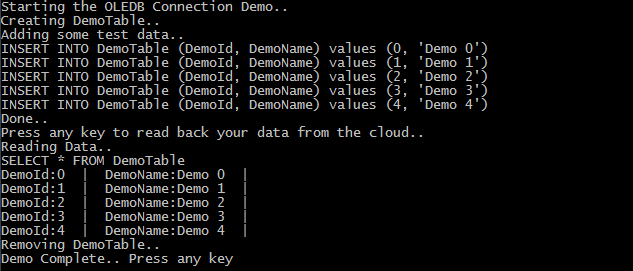
Task 3 – Connecting to SQL Azure Using ODBC

In this task, you will create a class that inherits from the **SQLAzureConnectionDemo** class and implement the methods for connecting to SQL Azure using ODBC.

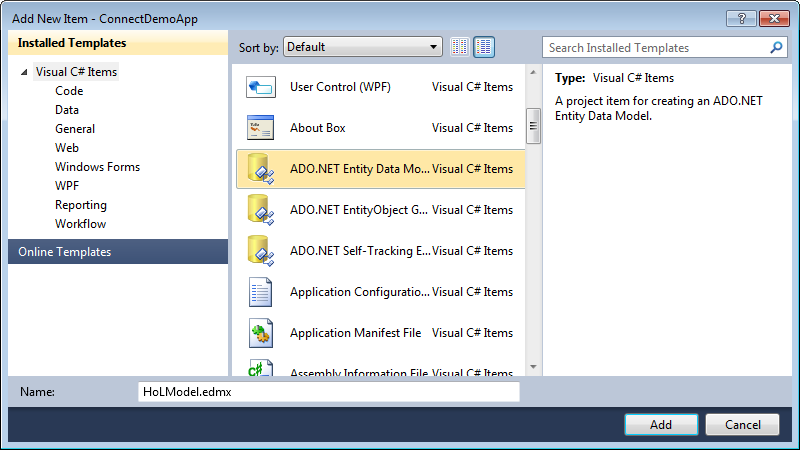
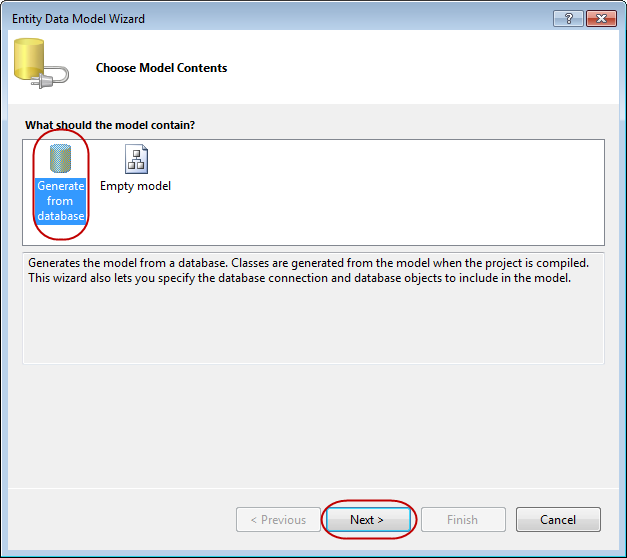
* 1. Add a new class to the project named **OdbcConnectionDemo**. To do this, right-click the **ConnectDemoApp** project in **Solution Explorer** and select **Add** | **Class**. In the **Add New Item** dialog, make sure that you select the **Class** template and set the name to **OdbcConnectionDemo.cs** or **OdbcConnectionDemo.vb** depending on the language of your project**.**
  2. Make sure that you have the following namespace directives at the top of the file:
     1. C#
     2. **using System.Data.Common;**
     3. **using System.Data.Odbc;**
     4. Visual Basic
     5. **Imports System.Data.Common**
     6. **Imports System.Data.Odbc**
  3. Update the class definition to make it public and to inherit from **SQLAzureConnectionDemo**. The final implementation should look like the following:
     1. **Note:** In Visual Basic, the template for a new class already declares the class as Public.
     2. C#
     3. **public** class OdbcConnectionDemo **:** **SQLAzureConnectionDemo**
     4. {
     5. }
     6. Visual Basic
     7. Public Class OdbcConnectionDemo
     8. **Inherits** **SQLAzureConnectionDemo**
     9. End Class
  4. Implement the class constructor to get the connection information and pass it as parameters to the base class constructor:
     1. (Code Snippet – *Intro to SQL Azure – Ex4 ODBC constructor – C#*)
     2. C#
     3. **public OdbcConnectionDemo(string userName, string password, string dataSource, string databaseName):**
     4. **base (userName, password, dataSource, databaseName)**
     5. **{**
     6. **}**
     7. (Code Snippet – *Intro to SQL Azure – Ex4 ODBC constructor – VB*)
     8. Visual Basic
     9. **Public Sub New(ByVal userName As String, ByVal password As String, ByVal dataSource As String, ByVal databaseName As String)**
     10. **MyBase.New(userName, password, dataSource, databaseName)**
     11. **End Sub**
  5. Override the **CreateConnection** method to create an **OdbcConnection** in your **OdbcConnectionDemo** class:
     1. (Code Snippet – *Intro to SQL Azure – Ex4 ODBC CreateConnection – C#*)
     2. C#
     3. **protected override DbConnection CreateConnection(string userName, string password, string dataSource, string databaseName)**
     4. **{**
     5. **return new OdbcConnection(CreateOdbcConnectionString(userName, password, dataSource, databaseName));**
     6. **}**
     7. (Code Snippet – *Intro to SQL Azure – Ex4 ODBC CreateConnection – VB*)
     8. Visual Basic
     9. **Protected Overrides Function CreateConnection(ByVal userName As String, ByVal password As String, ByVal dataSource As String, ByVal databaseName As String) As DbConnection**
     10. **Return New OdbcConnection(CreateOdbcConnectionString(userName, password, dataSource, databaseName))**
     11. **End Function**
  6. Implement the **CreateOdbcConnectionString** method used by the **CreateConnection** method. This method is responsible for building up the ODBC Drivers connection string. The proposed implementation is using **SQL Server Native Client 10.0** as its driver. You can specify any other ODBC driver of your preference here.
     1. (Code Snippet – *Intro to SQL Azure – Ex4 ODBC CreateOdbcConnectionString method – C#*)
     2. C#
     3. **private string CreateOdbcConnectionString(string userName, string password, string dataSource, string databaseName)**
     4. **{**
     5. **string serverName = GetServerName(dataSource);**
     6. **OdbcConnectionStringBuilder connectionStringBuilder = new OdbcConnectionStringBuilder**
     7. **{**
     8. **Driver = "SQL Server Native Client 10.0",**
     9. **};**
     10. **connectionStringBuilder["Server"] = "tcp:" + dataSource;**
     11. **connectionStringBuilder["Database"] = databaseName;**
     12. **connectionStringBuilder["Uid"] = userName + "@" + serverName;**
     13. **connectionStringBuilder["Pwd"] = password;**
     14. **return connectionStringBuilder.ConnectionString;**
     15. **}**
     16. (Code Snippet – *Intro to SQL Azure – Ex4 ODBC CreateOdbcConnectionString method – VB*)
     17. Visual Basic
     18. **Private Function CreateOdbcConnectionString(ByVal userName As String, ByVal password As String, ByVal dataSource As String, ByVal databaseName As String) As String**
     19. **Dim serverName As String = GetServerName(dataSource)**
     20. **Dim connectionStringBuilder As OdbcConnectionStringBuilder = New OdbcConnectionStringBuilder With {.Driver = "SQL Server Native Client 10.0"}**
     21. **connectionStringBuilder("Server") = "tcp:" & dataSource**
     22. **connectionStringBuilder("Database") = databaseName**
     23. **connectionStringBuilder("Uid") = userName & "@" & serverName**
     24. **connectionStringBuilder("Pwd") = password**
     25. **Return connectionStringBuilder.ConnectionString**
     26. **End Function**
  7. Override the **CreateCommand** method to create an **OdbcCommand**. Remember that this abstract method is called in the parent class to get the connection and execute the different SQL statement samples.
     1. (Code Snippet – *Intro to SQL Azure – Ex4 ODBC CreateCommand method – C#*)
     2. C#
     3. **protected override DbCommand CreateCommand(DbConnection connection)**
     4. **{**
     5. **return new OdbcCommand() { Connection = connection as OdbcConnection };**
     6. **}**
     7. (Code Snippet – *Intro to SQL Azure – Ex4 ODBC CreateCommand method – VB*)
     8. Visual Basic
     9. **Protected Overrides Function CreateCommand(ByVal connection As DbConnection) As DbCommand**
     10. **Return New OdbcCommand() With {.Connection = TryCast(connection, OdbcConnection)}**
     11. **End Function**
     12. That is the specific code required to use an ODBC connection. Now you will include some code in the **Program.cs** or **Module1.vb** file (depending on the language of your project) to test the connection and see how the different operations work.
  8. Open the **Program.cs** (for Visual C# projects) or **Module1.vb** (for Visual Basic projects) file double-clicking it in the **Solution** **Explorer** inside the **ConnectDemoApp** project.
  9. In method **Main**, implement the logic to create a new instance of the **OdbcConnectionDemo** class and execute the demo against SQL Azure. You can add or replace the code from the previous tasks based on whether you want to test all the technologies at once or only this one.
     1. (Code Snippet – *Intro to SQL Azure – Ex4 ODBC demo implementation – C#*)
     2. C#
     3. static void Main(string[] args)
     4. {
     5. //...
     6. **//Invoke the ODBC connection demo**
     7. **Console.WriteLine("Starting the ODBC Connection Demo...");**
     8. **OdbcConnectionDemo demo2 = new OdbcConnectionDemo(userName, password, datasource, databaseName);**
     9. **demo2.ConnectToSQLAzureDemo();**
     10. **Console.WriteLine("Demo Complete... Press any key");**
     11. **Console.ReadKey();**
     12. }
     13. (Code Snippet – *Intro to SQL Azure – Ex4 ODBC demo implementation – VB*)
     14. Visual Basic
     15. Sub Main()
     16. ' ...
     17. **' Invoke the ODBC connection demo**
     18. **Console.WriteLine("Starting the ODBC Connection Demo...")**
     19. **Dim demo2 = New OdbcConnectionDemo(\_userName, \_password, \_datasource, \_databaseName)**
     20. **demo2.ConnectToSQLAzureDemo()**
     21. **Console.WriteLine("Demo Complete... Press any key")**
     22. **Console.ReadKey()**
     23. End Sub
  10. If you have not done so before, update the value of the member variables located immediately above method **Main** by replacing the placeholders with the connection information for your SQL Azure account.
  11. Run the application by pressing **F5**. You should see the following output in a console window.
      1. 
      2. Figure 53
      3. Expected output from the ODBC connection demo

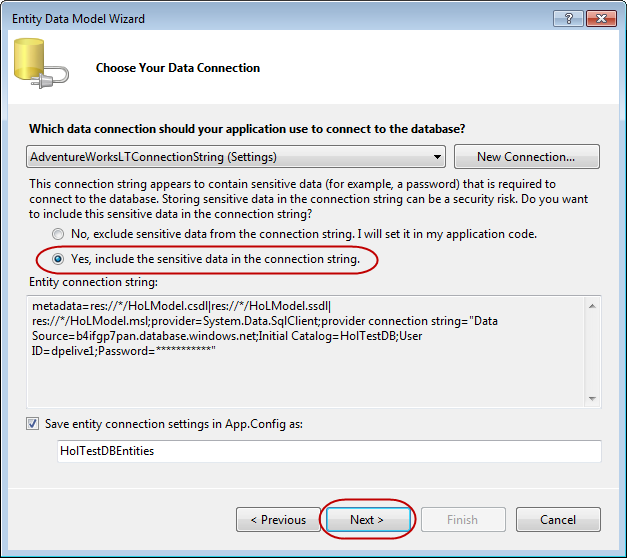
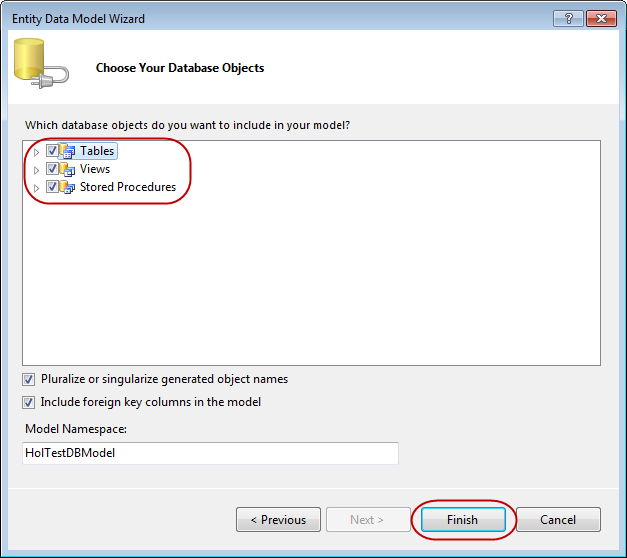
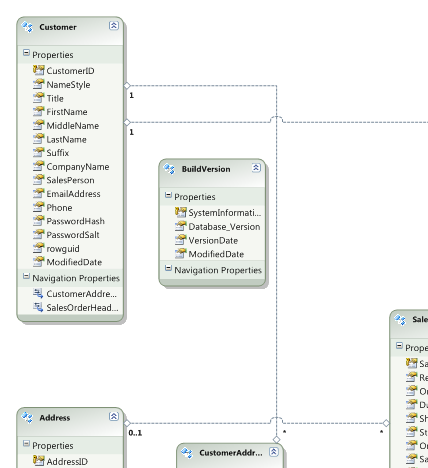
Task 4 – Connecting to SQL Azure Using OLEDB

In this task, you will create a class that inherits from the **SQLAzureConnectionDemo** class and implements the methods for connecting to SQL Azure using OLEDB.

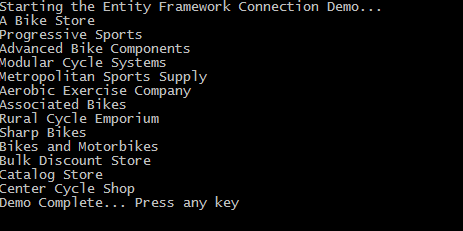
* 1. Add a new class to the project named **OleDbConnectionDemo**. To do this, right-click the **ConnectDemoApp** project in **Solution** **Explorer** and select **Add** | **Class**. In the **Add** **New** **Item** dialog, make sure to select the **Class** template and set the name to **OleDbConnectionDemo.cs** or **OleDbConnectionDemo.vb** depending on the language of your project**.**
  2. Make sure that you have the following namespace directives at the top of the file:
     1. C#
     2. **using System.Data.Common;**
     3. **using System.Data.OleDb;**
     4. Visual Basic
     5. **Imports System.Data.Common**
     6. **Imports System.Data.OleDb**
  3. Update the class definition to make it public and to inherit from **SQLAzureConnectionDemo**. It should look like the following:
     1. **Note:** In Visual Basic, the template for a new class already declares the class as Public.
     2. C#
     3. **public** class OleDbConnectionDemo **: SQLAzureConnectionDemo**
     4. {
     5. }
     6. Visual Basic
     7. Public Class OleDbConnectionDemo
     8. **Inherits** **SQLAzureConnectionDemo**
     9. End Class
  4. Implement the class constructor to get the connection information and pass it as parameters to the base class constructor:
     1. (Code Snippet – *Intro to SQL Azure – Ex4 OLEDB constructor – C#*)
     2. C#
     3. **public OleDbConnectionDemo(string userName, string password, string dataSource, string databaseName)**
     4. **: base(userName, password, dataSource, databaseName)**
     5. **{**
     6. **}**
     7. (Code Snippet – *Intro to SQL Azure – Ex4 OLEDB constructor – VB*)
     8. Visual Basic
     9. **Public Sub New(ByVal userName As String, ByVal password As String, ByVal dataSource As String, ByVal databaseName As String)**
     10. **MyBase.New(userName, password, dataSource, databaseName)**
     11. **End Sub**
  5. Override the **CreateConnection** method to create an **OleDbConnection** in your **OleDbConnectionDemo** class:
     1. (Code Snippet – *Intro to SQL Azure – Ex4 OLEDB CreateConnection – C#*)
     2. C#
     3. **protected override DbConnection CreateConnection(string userName, string password, string dataSource, string databaseName)**
     4. **{**
     5. **return new OleDbConnection(CreateOleDBConnectionString(userName, password, dataSource, databaseName));**
     6. **}**
     7. (Code Snippet – *Intro to SQL Azure – Ex4 OLEDB CreateConnection – VB*)
     8. Visual Basic
     9. **Protected Overrides Function CreateConnection(ByVal userName As String, ByVal password As String, ByVal dataSource As String, ByVal databaseName As String) As DbConnection**
     10. **Return New OleDbConnection(CreateOleDBConnectionString(userName, password, dataSource, databaseName))**
     11. **End Function**
  6. Implement the **CreateOleDbConnectionString** method used by the **CreateConnection** method. This method is responsible for building up the connection string used to create the connection to SQL Azure using OLEDB.
     1. (Code Snippet – *Intro to SQL Azure – Ex4 OLEDB CreateOleDbConnectionString method – C#*)
     2. C#
     3. **private string CreateOleDBConnectionString(string userName, string password, string dataSource, string databaseName)**
     4. **{**
     5. **string serverName = GetServerName(dataSource);**
     6. **OleDbConnectionStringBuilder connectionStringBuilder = new OleDbConnectionStringBuilder**
     7. **{**
     8. **Provider = "SQLOLEDB",**
     9. **DataSource = dataSource,**
     10. **};**
     11. **connectionStringBuilder["Initial Catalog"] = databaseName;**
     12. **connectionStringBuilder["UId"] = userName + "@" + serverName;**
     13. **connectionStringBuilder["Pwd"] = password;**
     14. **return connectionStringBuilder.ConnectionString;**
     15. **}**
     16. (Code Snippet – *Intro to SQL Azure – Ex4 OLEDB CreateOleDbConnectionString method – VB*)
     17. Visual Basic
     18. **Private Function CreateOleDBConnectionString(ByVal userName As String, ByVal password As String, ByVal dataSource As String, ByVal databaseName As String) As String**
     19. **Dim serverName As String = GetServerName(dataSource)**
     20. **Dim connectionStringBuilder As OleDbConnectionStringBuilder = New OleDbConnectionStringBuilder With {.Provider = "SQLOLEDB", .DataSource = dataSource}**
     21. **connectionStringBuilder("Initial Catalog") = databaseName**
     22. **connectionStringBuilder("UId") = userName & "@" & serverName**
     23. **connectionStringBuilder("Pwd") = password**
     24. **Return connectionStringBuilder.ConnectionString**
     25. **End Function**
  7. Override the **CreateCommand** method to create an **OleDbCommand**. Remember that this abstract method is called in the parent class to get the connection and execute the different SQL statement samples.
     1. (Code Snippet – *Intro to SQL Azure – Ex4 OLEDB CreateCommand method – C#*)
     2. C#
     3. **protected override DbCommand CreateCommand(DbConnection connection)**
     4. **{**
     5. **return new OleDbCommand() { Connection = connection as OleDbConnection };**
     6. **}**
     7. (Code Snippet – *Intro to SQL Azure – Ex4 OLEDB CreateCommand method – VB*)
     8. Visual Basic
     9. **Protected Overrides Function CreateCommand(ByVal connection As DbConnection) As DbCommand**
     10. **Return New OleDbCommand() With {.Connection = TryCast(connection, OleDbConnection)}**
     11. **End Function**
     12. That is the specific code required to use an OLEDB connection. Now you will include some code in the **Program.cs** or **Module1.vb** file (depending on the language of your project) to test the connection and see how the different operations work.
  8. Open the **Program.cs** (for Visual C# projects) or **Module1.vb** (for Visual Basic projects) file double-clicking it in the **Solution** **Explorer** inside the **ConnectDemoApp** project.
  9. In method **Main**, implement the logic to create an instance of the **OleDbConnectionDemo** class and execute the demo against SQL Azure. You can add or replace the code from the previous tasks based on whether you want to test all the technologies at once or only this one.
     1. (Code Snippet – *Intro to SQL Azure – Ex4 OLEDB demo implementation – C#*)
     2. C#
     3. static void Main(string[] args)
     4. {
     5. //...
     6. **//Invoke the OleDB connection demo**
     7. **Console.WriteLine("Starting the OLEDB Connection Demo…");**
     8. **OleDbConnectionDemo demo3 = new OleDbConnectionDemo(userName, password, datasource, databaseName);**
     9. **demo3.ConnectToSQLAzureDemo();**
     10. **Console.WriteLine("Demo Complete... Press any key");**
     11. **Console.ReadKey();**
     12. }
     13. (Code Snippet – *Intro to SQL Azure – Ex4 OLEDB demo implementation – VB*)
     14. Visual Basic
     15. Sub Main()
     16. ' ...
     17. **' Invoke the OleDB connection demo**
     18. **Console.WriteLine("Starting the OLEDB Connection Demo...")**
     19. **Dim demo3 = New OleDbConnectionDemo(\_userName, \_password, \_datasource, \_databaseName)**
     20. **demo3.ConnectToSQLAzureDemo()**
     21. **Console.WriteLine("Demo Complete... Press any key")**
     22. **Console.ReadKey()**
     23. End Sub
  10. If you have not done so before, update the value of the member variables located immediately above method **Main** by replacing the placeholders with the connection information for your SQL Azure account.
  11. Run the application by pressing **F5**. You should see the following output in a console window.
      1. 
      2. Figure 54
      3. Expected output from the OLEDB connection demo

Task 5 – Connecting to SQL Azure Using Entity Framework

* 1. You have connected in three different ways to the database on SQL Azure. The last technology that you are going to try will be **Entity Framework**. You will notice that the class implementation for this demo will not inherit from the **SQLAzureConnectionDemo** class because when using **Entity Framework**, you do not have to manage Connections and Commands; those are administered by the underlying technology and you do not have to worry about them.
  2. Open the **App.config** file and change the connection string to point to your SQL Azure Database, and to connect using the test user created earlier in this lab.
     1. **Note:** This step is required because **Entity Framework** gets the connection settings from the configuration file.
  3. Add a new ADO.NET Entity Data Model to the project named **HoLModel**. To do this, right-click on the **ConnectDemoApp** project in the **Solution Explorer.** Select **Add | New Item**. In the **Add New Item** dialog, make sure you select **ADO.NET Entity Data Model** template and then set the name to **HoLModel.edmx.**
     1. 
     2. Figure 55
     3. Adding EF model
  4. In the **Entity Data Model Wizard** select **Generate from database** and click **Next**.
     1. 
     2. Figure 56
     3. Choosing model contents
  5. In the **Choose Your Data Connection** step, select **Yes, include sensitive data in the connection string** and leave **AdventureWorksLTConnectionString** as data connection and **HolTestDBEntities** as entity connection settings name.

* + 1. 
    2. Figure 57
    3. Choosing model contents
  1. In the **Choose Your Database Objects** step, select all database objects and click **Finish** .
     1. 
     2. Figure 58
     3. Choosing database objects
  2. Once created the Model is shown.
     1. 
     2. Figure 59
     3. EF model created
  3. Add a new class to theproject named **EFConnectionDemo**. To do this, right-click the **ConnectDemoApp** project in **Solution** **Explorer** and select **Add** | **Class**. In the **Add** **New** **Item** dialog, make sure you select the **Class** template and then set the name to **EFConnectionDemo.cs** or **EFConnectionDemo.vb** based on the language of your preference**.**
  4. Update the class definition to make it public. It should look like the following:
     1. C#
     2. **public** class EFConnectionDemo
     3. {
     4. }
     5. Visual Basic
     6. Public Class EFConnectionDemo
     7. End Class
  5. Only for C#, make sure that you have the following namespace directives at the top of the class:
     1. C#
     2. **using System;**
     3. **using System.Linq;**
  6. Add the following method to the **EFConnectionDemo** class. This retrieves from the database all the company names and prints them to the console. To do that, it takes advantage of the EF **HolTestDBEntities** class.
     1. (Code Snippet – *Intro to SQL Azure – Ex4 EF ConnectToSQLAzure method – C#*)
     2. C#
     3. **/// <summary>**
     4. **/// HolTestDBEntities takes care of handling your transactions for you**
     5. **/// leaving you free use Linq to extract information stores up in the cloud**
     6. **/// </summary>**
     7. **public void ConnectToSQLAzureDemo()**
     8. **{**
     9. **HolTestDBEntities context = new HolTestDBEntities();**
     10. **IQueryable<string> companyNames = from customer in context.Customers**
     11. **where customer.CustomerID < 20**
     12. **select customer.CompanyName;**
     14. **foreach (var company in companyNames)**
     15. **{**
     16. **Console.WriteLine(company);**
     17. **}**
     18. **}**
     20. (Code Snippet – *Intro to SQL Azure – Ex4 EF ConnectToSQLAzure method – VB*)

Visual Basic

* + 1. **''' <summary>**
    2. **''' HolTestDBEntities takes care of handling your transactions for you**
    3. **''' leaving you free you use Linq to extraxt information stored up in the cloud.**
    4. **''' </summary>**
    5. **Public Sub ConnectToSQLAzureDemo()**
    6. **Dim context As New HolTestDBEntities()**
    7. **' get all company names**
    8. **Dim companyNames As IQueryable(Of String) = From customer In context.Customers \_**
    9. **Where customer.CustomerID < 20 \_**
    10. **Select customer.CompanyName**
    11. **' display these all on the console**
    12. **For Each company As String In companyNames**
    13. **Console.WriteLine(company)**
    14. **Next company**
    15. **End Sub**
  1. Add the following code to invoke the LINQ to SQL demo in method **Main** of the **Program.cs** file (for Visual C# projects) or **Module1.vb** (for Visual Basic projects) file. You can add or replace the code from the previous tasks depending on whether you want to test all the technologies at once or only this one.
     1. (Code Snippet – *Intro to SQL Azure – Ex4 EF demo implementation – C#*)
     2. C#
     3. static void Main(string[] args)
     4. {
     5. //...
     6. **//Invoke the Entity Framework connection demo**
     7. **Console.WriteLine("Starting the Entity Framework Connection Demo...");**
     8. **EFConnectionDemo demo4 = new EFConnectionDemo();**
     9. **demo4.ConnectToSQLAzureDemo();**
     10. **Console.WriteLine("Demo Complete... Press any key");**
     11. **Console.ReadKey();**
     12. }
     13. (Code Snippet – *Intro to SQL Azure – Ex4 EF demo implementation – VB*)
     14. Visual Basic
     15. Sub Main()
     16. ' ...
     17. **' Invoke the Entity Framework connection demo**
     18. **Console.WriteLine("Starting the Entity Framework Connection Demo...")**
     19. **Dim demo4 = New EFConnectionDemo()**
     20. **demo4.ConnectToSQLAzureDemo()**
     21. **Console.WriteLine("Demo Complete... Press any key")**
     22. **Console.ReadKey()**
     23. End Sub
  2. Press **F5** to run your application. You should see a long list of company names. These are retrieved from your database on the SQL Azure Server using LINQ to SQL.
     1. 
     2. Figure 60
     3. Expected output from the EF connection demo

Task 6 – Connecting to SQL Azure via Non-Microsoft Technologies

* 1. It is trivial to connect to SQL Azure using non-windows technologies.
  2. The following PHP version takes on a pattern that you should be familiar with from the previous task. It uses the SQL Server Native Client ODBC driver to establish a connection.
  3. PHP
  4. **<?php**
  5. **$host = "server.database.windows.net";**
  6. **$dbname = "database";**
  7. **$dbuser = "user@server";**
  8. **$dbpwd = "password";**
  9. **$driver = "{SQL Server Native Client 10.0}";**
  10. **// Build connection string**
  11. **$dsn="Driver=$driver;Server=$host;Database=$dbname;Encrypt=true;TrustServerCertificate=true";**
  12. **if (!($conn = @odbc\_connect($dsn, $dbuser, $dbpwd))) {**
  13. **die("Connection error: " . odbc\_errormsg());**
  14. **}**
  16. **// Got a connection, do what you will**
  18. **// Free the connection**
  19. **@odbc\_close($conn);**
  20. **?>**
  21. Connecting to SQL Azure using JDBC is also trivial. Refer to the following code.
  22. JAVA
  23. **// Build a connection string**
  24. **String connectionUrl= "jdbc:sqlserver://server.database.windows.net;" +**
  25. **"database=mydatabase;encrypt=true;user=user@server;password=\*\*\*\*\*";**
  26. **// Next, make the sure the SQL Server Driver is loaded.**
  27. **Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");**
  29. **// Then attempt to get a connection. This will null or throw if we can't get a connection.**
  30. **Connection sqlConn = DriverManager.getConnection(connectionUrl);**
  31. **if (sqlConn == null)**
  32. **{**
  33. **System.out.println("Unable to obtain connection. exiting");**
  34. **System.exit(1);**
  35. **}**
  36. **// Got a connection, do what you will**
  37. **// Free the connection**
  38. **sqlConn.close();**

Summary

* 1. In this lab, you have looked at the basics of working with SQL Azure. If you have any SQL Server experience, you may have found the lab familiar and that is, indeed, the point. Working with SQL Azure should be very familiar to anyone who has worked with SQL Server.
  2. You learned to create new databases, logins and users for those databases. You saw that for the most part, you could simply create objects in SQL Azure as you would with an on-premise SQL Server.
  3. In addition, you created a simple Windows Azure application that is able to consume a SQL Azure database.

Finally, you saw that creating connections to SQL Azure using Microsoft technologies is the same as creating connections to any normal on-premise database.