ASP.NET Data Access Options

**.NET Framework 4.5**

[Other Versions](javascript:;)

25 out of 30 rated this helpful - [Rate this topic](http://msdn.microsoft.com/en-us/library/ms178359.aspx#feedback)

ASP.NET provides many options for storing, retrieving, and displaying data. For developers who are new to ASP.NET, the appropriate options are not always obvious. This topic provides recommendations and guidelines for choosing the options that best fit your scenario.

This topic focuses on relational databases. For many scenarios, a NoSQL solution might be a better choice. For more information, see [Data Storage Options (Building Real-World Cloud Apps with Windows Azure)](http://www.asp.net/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/data-storage-options). For additional resources related to data access in ASP.NET web applications, see also [ASP.NET Data Access – Recommended Resources](http://www.asp.net/whitepapers/aspnet-data-access-content-map) on the ASP.NET site.

The topic contains the following sections:

* [Database Management Systems (DBMS)](http://msdn.microsoft.com/en-us/library/ms178359.aspx#databases)
* [SQL Server Editions](http://msdn.microsoft.com/en-us/library/ms178359.aspx#sqlserver)
* [Object-Relational Mappers (ORM)](http://msdn.microsoft.com/en-us/library/ms178359.aspx#orm)
* [Entity Framework Development Workflows](http://msdn.microsoft.com/en-us/library/ms178359.aspx#dbfmfcf)
* [LINQ versus SQL](http://msdn.microsoft.com/en-us/library/ms178359.aspx#linq)
* [Accessing Data through a Web Service](http://msdn.microsoft.com/en-us/library/ms178359.aspx#webservice)
* [Working with Data in ASP.NET Web Forms Applications](http://msdn.microsoft.com/en-us/library/ms178359.aspx#webforms)
* [Working with Data in ASP.NET MVC applications](http://msdn.microsoft.com/en-us/library/ms178359.aspx#mvc)
* [Working with Data in ASP.NET Web Pages Applications](http://msdn.microsoft.com/en-us/library/ms178359.aspx#webpages)
* [Summary of Recommended Choices](http://msdn.microsoft.com/en-us/library/ms178359.aspx#summary)

For information about how to choose data access options for ASP.NET Web Forms, such as data-bound controls, data-binding methods, and Dynamic Data scaffolding, see [ASP.NET Web Forms Data Access Options](http://msdn.microsoft.com/en-us/library/jj822927.aspx).

[Database Management Systems (DBMS)](javascript:void(0))

The technology you use to connect to a relational database in ASP.NET code that runs on a server is [ADO.NET](http://msdn.microsoft.com/en-us/sqlserver/aa937722.aspx). ADO.NET communicates with a database management system (DBMS) such as SQL Server or Oracle by using *data provider* software. Microsoft data providers enable you to connect to the following databases:

* SQL Server, including SQL Server Express and LocalDB
* SQL Server Compact
* Any database that supports ODBC or OLEDB and is suitable for use in a web application

|  |
| --- |
| **NoteNote** |
| Microsoft Access is not recommended for web applications. For more information, see **Can my ASP.NET web application use a Microsoft Access database?** in [ASP.NET Data Access FAQ](http://msdn.microsoft.com/en-us/library/jj653753.aspx). |

You can also get data providers from third-party software vendors. Some popular relational databases that you can get data providers for include MySQL, SQLite, Oracle, and DB2. For information about data providers that are available, see [.NET Framework Data Providers](http://msdn.microsoft.com/en-us/library/a6cd7c08.aspx) and [ADO.NET Data Providers](http://msdn.microsoft.com/en-us/data/dd363565).

If you're choosing a relational database for an ASP.NET web application and you don't have special needs that dictate a different choice, choose SQL Server. Some reasons for choosing SQL Server include the following:

* SQL Server is supported by Microsoft.
* SQL Server is integrated with other Microsoft data access technologies such as the Entity Framework. See [Object-Relational Mappers](http://msdn.microsoft.com/en-us/library/ms178359.aspx#orm) later in this topic.
* Visual Studio provides built-in tools for working with SQL Server. [SQL Server Data Tools](http://msdn.microsoft.com/en-us/data/tools.aspx) (SSDT), enables you to create databases, manipulate schema and data, generate and run scripts, debug, and deploy databases and database updates.
* Visual Studio includes web deployment features that are designed to facilitate deploying SQL Server databases along with web projects. For more information, see **Configuring Database Deployment in Visual Studio** in [ASP.NET Web Site Project Deployment Overview](http://msdn.microsoft.com/en-us/library/377y0s6t.aspx).
* The Visual Studio web project templates use SQL Server for the ASP.NET membership database by default.
* Nearly all ASP.NET documentation on the MSDN web site, ASP.NET web sites, and in blog posts uses SQL Server. Relatively little documentation that would help you get started and resolve difficulties is available for other databases.

NoSQL databases are alternatives to relational databases. NoSQL databases can provide advantages over relational databases for applications that manage large volumes of unstructured data. The [Windows Azure Table storage service](http://www.windowsazure.com/en-us/develop/net/how-to-guides/table-services/) is an example of a NoSQL database management system. Providing guidance on choosing a NoSQL database is outside of the scope of this document.

[SQL Server Editions](javascript:void(0))

There are several SQL Server editions to choose from:

* [SQL Server Express](http://www.microsoft.com/sqlserver/en/us/editions/2012-editions/express.aspx). The entry-level, free edition of SQL Server. For more information about SQL Server Express, see [Working with SQL Server Express Databases](http://msdn.microsoft.com/en-us/library/6759sth4.aspx#sse).
* [SQL Server Express LocalDB](http://msdn.microsoft.com/en-us/library/hh510202(v=sql.110).aspx). A special execution mode of SQL Server Express that enables you to work with databases as .mdf files. Typically, LocalDB database files are kept in the App\_Data folder of a web project. (The *user instance* feature in SQL Server Express also enables you to work with .mdf files, but the user instance feature is deprecated; therefore, LocalDB is recommended for working with .mdf files.)

In Visual Studio 2010 and earlier versions, SQL Server Express is installed by default with Visual Studio; in Visual Studio 2012 LocalDB is installed by default with Visual Studio. LocalDB is recommended for web development but not for production web applications because it is not designed to work with IIS. For more information about LocalDB, see[Working with SQL Server Express LocalDB Databases](http://msdn.microsoft.com/en-us/library/6759sth4.aspx#sslocaldb).

* [SQL Server Developer Edition](http://msdn.microsoft.com/en-us/library/ms144275). Has all the features of Enterprise edition but is licensed only for development and testing, not for production.
* [SQL Server (full editions)](http://msdn.microsoft.com/en-us/library/ms144275)

Several full editions are available: Enterprise, Business Intelligence, Standard, and Web.

* [Windows Azure SQL Database](http://msdn.microsoft.com/en-us/library/windowsazure/ee336279). The cloud edition of SQL Server, formerly known as SQL Azure.
* [SQL Server Compact 4.0](http://www.microsoft.com/sqlserver/en/us/editions/2012-editions/compact.aspx). A free, lightweight database engine that you can use in ASP.NET web applications for both development and production.

SQL Server Compact uses a different database engine and .NET data provider than the other editions. SQL Server Management Studio (SSMS) and SQL Server Data Tools (SSDT) don’t work with SQL Server Compact 4.0. For more information, see [Differences Between SQL Server Compact and SQL Server](http://msdn.microsoft.com/en-us/library/bb896140.aspx) and [Using SQL Server Compact for ASP.NET Web Applications](http://msdn.microsoft.com/en-us/library/ms247257.aspx).

Guidance for choosing an edition for your production database is outside the scope of this topic. That choice is generally determined by your production environment, or you're working on an existing application that already has a database. If you need to choose a SQL Server edition for production, see the following resources:

* [ASP.NET Data Access Content Map](http://msdn.microsoft.com/en-us/library/6759sth4.aspx) (See **Choosing between SQL Server and Windows Azure SQL Database**.)
* [Features Supported by the Editions of SQL Server 2012](http://msdn.microsoft.com/en-us/library/cc645993)

You don't have to use the same edition of SQL Server in development that you use in production. The choice of SQL Server edition for development depends on your development environment and the edition you choose for your production database:

* If you're using Visual Studio 2012, and your production database is SQL Database or a full edition of SQL Server, the default choice for development is LocalDB. This is the default choice because LocalDB is installed with Visual Studio 2012 by default.
* If you're using Visual Studio 2010 or WebMatrix, and your production database is SQL Database or a full edition of SQL Server, the default choice for development is SQL Server Express. This is the default choice because SQL Server Express 2008 is installed with Visual Studio 2010 by default, and LocalDB doesn't work with WebMatrix.
* If your production database is SQL Server Compact, use SQL Server Compact for development.

Exceptions that might dictate a different choice for development include the following:

* If you want to use IIS as your development web server in Visual Studio 2012, use SQL Server Express instead of LocalDB. You will have to install SQL Server Express, because it is not installed with Visual Studio 2012 by default. For information about why you might want to use IIS in development, and how to configure Visual Studio to use it, see [Test your ASP.NET MVC or Web Forms Application on IIS 7 in 30 seconds](http://blogs.msdn.com/b/rickandy/archive/2011/04/22/test-you-asp-net-mvc-or-webforms-application-on-iis-7-in-30-seconds.aspx) on Rick Anderson's blog.
* If there are features of the edition you use in production that aren't available in LocalDB or SQL Server Express, use Developer Edition or the same edition that you use in production.
* If you want to enable multiple developers to work on the same Visual Studio web project, with some of them using Visual Studio 2010 and some of them using Visual Studio 2012, see **How can Visual Studio 2010 and Visual Studio 2012 share a project that uses a database?** in [ASP.NET Data Access FAQ](http://msdn.microsoft.com/en-us/library/jj653753.aspx).

For help with configuring a connection string for the version of SQL Server that you choose, see [SQL Server Connection Strings for ASP.NET Web Applications](http://msdn.microsoft.com/en-us/library/jj653752.aspx).

When you use a different database for development than the one you use for production, you have to deploy the database when you deploy (publish) the web application. Deploying a database includes creating the production database, creating tables and other objects in the production database, and copying initial data to the database. Visual Studio includes web deployment features that facilitate deploying databases along with web projects. For more information, see **Configuring Database Deployment in Visual Studio** in [Web Deployment Overview for Visual Studio and ASP.NET](http://msdn.microsoft.com/en-us/library/dd394698.aspx), and the **Deploying a database** scenarios in [Web Deployment Content Map for Visual Studio and ASP.NET](http://go.microsoft.com/fwlink/p/?LinkId=282413).

[Object-Relational Mappers (ORM)](javascript:void(0))

To read or update data, you can use ADO.NET directly by using classes such as [SqlCommand](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlcommand.aspx), [SqlDataReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader.aspx), [SqlDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldataadapter.aspx), and [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx). The recommended alternative is to let an object-relational mapper (ORM) framework such as the [Entity Framework](http://msdn.microsoft.com/en-us/library/bb399572.aspx) handle the low-level code that interfaces with an ADO.NET data provider. If you use ADO.NET directly, you have to manually write and execute SQL queries. You also have to write code that converts data from the database's format into objects, properties, and collections that you can work with in code.

A quick way to see what an ORM does for you is to compare samples of code you would write for an ORM and code you would write to do the same task using ADO.NET directly. In the following example, a school database has an Instructor table, a Course table, and a CourseInstructor association table for the many-to-many relationship between them. To display a list of instructors and the courses they teach, you populate Instructor and Course objects similar to these:

C#

public class Instructor

{

public string LastName { get; set; }

public string FirstMidName { get; set; }

public List<Course> Courses { get; set; }

}

public class Course

{

public string Title { get; set; }

public int Credits { get; set; }

}

Here is what code to populate these classes looks like when you use the Entity Framework ORM:

C#

public IEnumerable<Instructor> GetInstructors()

{

List<Instructor> instructors;

using (var db = new SchoolContext())

{

instructors = db.Instructors.Include("Courses").ToList();

}

return instructors;

}

And here is code that accomplishes the same task by using ADO.NET directly:

C#

public IEnumerable<Instructor> GetInstructorsADONET()

{

var connString = ConfigurationManager.ConnectionStrings["SchoolContext"].ConnectionString;

List<Instructor> instructors = new List<Instructor>();

using (var conn = new SqlConnection(connString))

{

conn.Open();

var cmd = new SqlCommand("SELECT Person.PersonID, Person.LastName, Person.FirstName, Course.Title, Course.Credits " +

"FROM Course INNER JOIN " +

"CourseInstructor ON Course.CourseID = CourseInstructor.CourseID RIGHT OUTER JOIN " +

"Person ON CourseInstructor.PersonID = Person.PersonID " +

"WHERE (Person.Discriminator = 'Instructor') " +

"ORDER BY Person.PersonID", conn);

var reader = cmd.ExecuteReader();

int currentPersonID = 0;

Instructor currentInstructor = null;

while (reader.Read())

{

var personID = Convert.ToInt32(reader["PersonID"]);

if (personID != currentPersonID)

{

currentPersonID = personID;

if (currentInstructor != null)

{

instructors.Add(currentInstructor);

}

currentInstructor = new Instructor();

currentInstructor.LastName = reader["LastName"].ToString();

currentInstructor.FirstMidName = reader["FirstName"].ToString();

}

if (reader["Title"] != DBNull.Value)

{

var course = new Course();

course.Title = reader["Title"].ToString();

course.Credits = Convert.ToInt32(reader["Credits"]);

currentInstructor.Courses.Add(course);

}

}

if (currentInstructor != null)

{

instructors.Add(currentInstructor);

}

reader.Close();

cmd.Dispose();

}

return instructors;

}

Notice that not only does much more code have to be written and tested and debugged when you don't use an ORM, but also the code you write is database-specific. The Entity Framework code would not change if you decided to migrate the application from SQL Server to Windows Azure SQL Database.

This is a simplified example; in a real-world example with more tables, more columns, and more complex relationships, the differences would be much greater. As you can see, an ORM can make you more productive, and your application easier to maintain. For these reasons, in most scenarios you would want to use an ORM in an ASP.NET application that uses a relational database.

The most commonly used ORMs that work with ASP.NET are the following:

* The [ADO.NET Entity Framework](http://msdn.microsoft.com/en-us/library/bb399572.aspx) is the main ORM that Microsoft provides for the .NET Framework.
* [LINQ to SQL](http://msdn.microsoft.com/en-us/library/bb386976.aspx) is a legacy ORM that Microsoft provides. (Don't confuse LINQ to SQL with LINQ. For information about LINQ, see [LINQ versus SQL](http://msdn.microsoft.com/en-us/library/ms178359.aspx#linq) later in this topic.)
* [NHibernate](http://nhforge.org/Default.aspx) is an open source ORM for the .NET Framework.

Other data access frameworks that provide some of the benefits of an ORM but with less abstraction and potentially better performance include [micro ORMs](http://programmers.stackexchange.com/questions/120457/what-are-some-of-the-benefits-of-a-micro-orm) such as [Dapper](http://code.google.com/p/dapper-dot-net/) and [Massive](https://github.com/robconery/massive), and the [Enterprise Library Data Access Application Block](http://msdn.microsoft.com/en-us/library/ff664408).

For new development where you're not constrained by a legacy data access approach, Microsoft recommends the Entity Framework. (Except for ASP.NET Web Pages applications that you develop by using WebMatrix. For more information, see [Working with Data in ASP.NET Web Pages Applications](http://msdn.microsoft.com/en-us/library/ms178359.aspx#webpages) later in this topic.) Some factors that make the Entity Framework the best choice include the following:

* Microsoft supports and continues to enhance the Entity Framework. Starting with version 6.0, the Entity Framework is open source and is also being enhanced by an open source community. LINQ to SQL is supported but Microsoft is investing minimal efforts to enhance it. (See "Where does Microsoft stand on LINQ to SQL?" in [Data Developer Center - Top Questions and Answers on Data](http://msdn.microsoft.com/en-us/data/bb525059.aspx).) LINQ to SQL has a much more limited feature set to begin with, and choosing LINQ to SQL would limit your ability to take advantage of future advances in data access technology. NHibernate is not supported by Microsoft.
* The Entity Framework is the easiest ORM framework to learn for an ASP.NET developer. Because Microsoft continues to focus ORM development efforts on the Entity Framework, Microsoft and third parties continue to produce new tutorials, videos, and books. Most new tutorials use the Entity Framework even if they are focused on some other technology, such as the latest release of MVC or new model binding features for Web Forms in ASP.NET 4.5. LINQ to SQL is sometimes said to be easier to learn because it has a simpler feature set, but you can get started with the Entity Framework by using only the features you need. You can take advantage of its more advanced features later when you are ready for them. The learning curve for nHibernate is relatively steep.
* The community of Entity Framework users is larger than the LINQ to SQL and nHibernate communities. When you run into a problem that you need help with, you're more likely to get the answer you need when you post it to sites like [stackoverflow.com](http://stackoverflow.com/) or [the ASP.NET data access forum](http://forums.asp.net/1227.aspx/1?ADO%20NET%20Entity%20Framework%20LINQ%20to%20SQL%20NHibernate) if you're using Entity Framework than if you're using one of the other ORMs.
* The Entity Framework offers the most productivity gains because it is integrated with the .NET Framework and Visual Studio web development tools. Although this is also true to some degree of LINQ to SQL, some features have been developed exclusively for the Entity Framework. For example, Visual Studio automates deployment for Entity Framework Code First databases; see [Web Deployment Overview for Visual Studio and ASP.NET](http://msdn.microsoft.com/en-us/library/dd394698.aspx). Integrated development tooling for NHibernate is relatively limited.
* The Entity Framework gives you a choice of databases or data services that you can use. Microsoft provides support for using SQL Server, SQL Server Compact, and WCF Data Services with the Entity Framework. Third-party vendors provide support for other databases, such as Oracle, MySQL, and SQLite. LINQ to SQL is limited to SQL Server. NHibernate can also be used with multiple database engines, but support is potentially more problematic.

Developers who are considering whether to adopt the Entity Framework frequently express concern about its impact on application performance. Any ORM, including the Entity Framework, will sometimes perform inefficiently compared to what you could accomplish by writing SQL and code manually. In most scenarios, the loss in performance is not likely to be noticeable to end users. In other scenarios a loss in performance may be an acceptable trade-off for the improvements in application maintainability and reliability that you get from using the Entity Framework. For scenarios where Entity Framework performance is not acceptable, you can take one of these approaches to resolve the issue:

* Write and execute your own SQL statements or stored procedures for scenarios that are not handled efficiently by the Entity Framework.

For example, suppose you want to change a date field by setting it to the current date in a table that has millions of rows. Using the Entity Framework to do that one row at a time would be very inefficient, but you could do it quickly by running a SQL update query.

* Change the way you configure or use the Entity Framework in problematic scenarios.

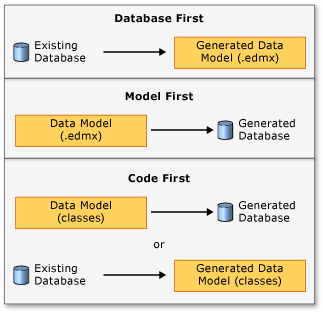
For example, the Entity Framework determines how an entity has changed (and therefore which updates need to be sent to the database) by comparing the current values of an entity with the original values. If you're tracking a large number of entities and you trigger automatic change tracking many times in a loop, the performance cost might be significant. In these scenarios you can improve performance by disabling change tracking until the loop is completed.

For more information about executing custom SQL statements while using the Entity Framework or configuring the Entity Framework to handle special situations, see [Advanced Entity Framework Scenarios for an MVC Web Application](http://www.asp.net/mvc/tutorials/getting-started-with-ef-using-mvc/advanced-entity-framework-scenarios-for-an-mvc-web-application) on the ASP.NET site.

For more information about the Entity Framework, see [ASP.NET Data Access Content Map](http://go.microsoft.com/fwlink/p/?LinkId=282414).

[Entity Framework Development Approaches](javascript:void(0))

As shown in the following diagram and in [this video clip](http://msdn.microsoft.com/en-us/data/jj590134), there are three ways you can work with data models and databases in the Entity Framework: *Database First*, *Model First*, and*Code First*.



* Database First

If you already have a database, the Entity Framework designer built into Visual Studio can automatically generate a data model that consists of classes and properties that correspond to existing database objects such as tables and columns. The information about your database structure (*store schema*), your data model (*conceptual model*), and the mapping between them is stored in XML in an *.edmx* file. The Entity Framework designer provides a graphical UI that you can use to display and edit the .edmx file.

* Model First

If you don't have a database yet, you can begin by creating a model in an .edmx file by using the Entity Framework graphical designer in Visual Studio. When the model is finished, the Entity Framework designer can generate DDL (data definition language) statements to create the database. As in Database First, the .edmx file stores model and mapping information.

* Code First

Whether you have an existing database or not, you can use the Entity Framework without using the designer or an .edmx file. If you don't have a database, you can code your own classes and properties that correspond to tables and columns. If you do have a database, Entity Framework tools can generate the classes and properties that correspond to existing tables and columns. The mapping between the store schema and the conceptual model represented by your code is handled by convention and by a special mapping API. If you let Code First create the database, you can use Code First Migrations to automate the process of deploying the database to production. Migrations can also automate the deployment of database schema changes to production when your data model changes.

Choose Code First for new development unless you want to use a graphical designer to model database objects and relationships. The Entity Framework designer only works with Database First and Model First. Before you choose Database First or Model First, however, consider how you want to handle updates to the data model after you create the database, and how you want to deploy the database and deploy updates to it. Code First Migrations automates the process of implementing and deploying database schema changes that result from data model changes. The advantages of Code First Migrations might outweigh the advantages of the Entity Framework designer.

For more information, see [ASP.NET Data Access Content Map](http://go.microsoft.com/fwlink/p/?LinkId=282414).

[LINQ versus SQL](javascript:void(0))

LINQ is a feature of the C# and Visual Basic languages that lets you query data by writing code. You can write LINQ expressions in two ways: as queries and as fluent API.

LINQ query syntax is similar to SQL except that the FROM clause comes first in LINQ so that Visual Studio can provide IntelliSense for the remainder of the statement. The following example of a LINQ query retrieves Department entities for departments that have at least one course assigned to them, and it sorts the resulting list by last name:

C#

var departments = from i in db.Departments

where i.Courses.Count > 0

orderby i.Name

select i;

Fluent API refers to the practice of chaining method calls in a single statement, as shown in the following example which does exactly the same thing as the LINQ query shown previously:

C#

var departments = db.Departments.Where(i => i.Courses.Count > 0).OrderBy(i => i.Name);

The choice between queries and fluent API depends on which syntax you're more comfortable with or want to learn, and you can mix and match based on the needs of each scenario. If you decide to use fluent API, you must learn or be familiar with [lambda expressions](http://msdn.microsoft.com/en-us/magazine/cc163362.aspx).

LINQ uses the .NET provider model to access data, which means that the same code can access different data stores depending on which provider is used. The .NET Framework includes the following LINQ providers:

* [LINQ to Objects](http://msdn.microsoft.com/en-us/library/bb397919.aspx)

Query any in-memory collection that implements the IEnumerable interface.

* [LINQ to DataSet](http://msdn.microsoft.com/en-us/library/bb386977.aspx)

Query DataTable and DataRow objects in a DataSet object.

* [LINQ to XML](http://msdn.microsoft.com/en-us/library/bb387098.aspx)

Query in-memory XML data.

* [LINQ to SQL](http://msdn.microsoft.com/en-us/library/bb386976.aspx)

Query a LINQ to SQL data model. For information about LINQ to SQL, see [Object-Relational Mappers (ORM)](http://msdn.microsoft.com/en-us/library/ms178359.aspx#orm) earlier in this topic.

* [LINQ to Entities](http://msdn.microsoft.com/en-us/library/bb386964)

Query an Entity Framework data model.

For an introduction to LINQ, see [LINQ (Language-Integrated Query)](http://msdn.microsoft.com/en-us/library/bb397926.aspx).

There is also a version of LINQ that is designed to facilitate constructing queries at run time out of information that is not known until run time. For example, you might want to specify a different **OrderBy** clause depending on which column a user clicks in a grid. In these scenarios you can use *dynamic LINQ*. For more information, see [Dynamic LINQ (Part 1: Using the LINQ Dynamic Query Library)](http://weblogs.asp.net/scottgu/archive/2008/01/07/dynamic-linq-part-1-using-the-linq-dynamic-query-library.aspx) on ScottGu's blog.

For data access scenarios in which LINQ is an alternative to SQL, LINQ is the recommended way to query data for the following reasons:

* LINQ provides compile-time validation of query syntax. With SQL you don't find out about syntax errors until run time.
* LINQ provides IntelliSense, which makes the process of writing queries quicker, easier, and less error-prone.
* LINQ provides a level of abstraction between your queries and the data store that they run against. You can change the data store without changing the code. For example, you might have written the queries in the previous example to run against an [IQueryable](http://msdn.microsoft.com/en-us/library/system.linq.iqueryable.aspx) object that would result in a database query. If you later decide to change the data source to an[IEnumerable<T>](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) collection, you can do that without changing the LINQ code. LINQ will automatically use the correct provider. (However, there may be slight changes between provider implementations, so whenever you make changes of this nature, test to verify that the result is the same.)
* Using LINQ automatically protects you from SQL injection attacks. If you're using SQL statements, you have to make a conscious effort to protect them from SQL injection attacks.

Exceptional scenarios in which you might want to use SQL instead of LINQ include the following:

* When you want to perform an operation more efficiently than would be possible by using the Entity Framework. For example, a single SQL UPDATE statement that affects many rows is more efficient than retrieving the rows with a LINQ statement, updating the entities in code, and then saving the changes.
* When you want to use an advanced SQL command such as the MERGE command.
* When you want to construct queries at run time, and dynamic LINQ does not have the degree of flexibility that you need.

There are two ways to execute SQL statements in the Entity Framework: [Entity SQL](http://msdn.microsoft.com/en-us/library/bb399560) and SQL that is native to the underlying database. Entity SQL is an older feature that is no longer the recommended method of executing SQL in the Entity Framework except for a few scenarios that require it. (One example is the [EntityDataSource](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.entitydatasource.aspx) control, which uses Entity SQL.) To execute SQL in the Entity Framework, use the [DbSet.SqlQuery Method](http://msdn.microsoft.com/en-us/library/system.data.entity.dbset.sqlquery.aspx), the [Database.SqlQuery Method](http://msdn.microsoft.com/en-us/library/gg679164), or the [Database.ExecuteSqlCommand Method](http://msdn.microsoft.com/en-us/library/system.data.entity.database.executesqlcommand). For more information, see[Advanced Entity Framework Scenarios for an MVC Web Application](http://www.asp.net/mvc/tutorials/getting-started-with-ef-using-mvc/advanced-entity-framework-scenarios-for-an-mvc-web-application) on the ASP.NET site.

[Accessing Data through a Web Service](javascript:void(0))

The following technologies can be used to make data available to clients over a web service:

* [WCF Data Services](http://msdn.microsoft.com/en-us/library/cc668792.aspx)

With WCF Data Services, you have to explicitly configure security restrictions: by default everything is made available and you write code to place restrictions on whatever you do not want to be available.

* [Web API](http://www.asp.net/web-api)

With Web API, by default nothing is made available, and you write code to explicitly make available only the data that you want to be available.

Choose WCF Data Services if you want to quickly expose a database over HTTP for use inside a firewall on an internal company network. In this scenario the consequences of accidentally making sensitive data available over the web service are less than if you were making the service available over the Internet. Conversely, choose Web API for Internet-based services because the whitelist approach is more secure than the blacklist approach. Web API is the preferred technology for flexible service development with REST approaches, OData, or JSON requirements.

[Working with Data in ASP.NET Web Forms Applications](javascript:void(0))

For information about data access options in ASP.NET Web forms, see [ASP.NET Web Forms Data Access Options](http://msdn.microsoft.com/en-us/library/jj822927.aspx).

[Working with Data in ASP.NET MVC Applications](javascript:void(0))

The recommended and default view engine for MVC is Razor. You can use *helpers* in MVC Razor views to automate the presentation or input of data in UI elements such as tables and text boxes and drop-down lists. Helpers correspond to data-bound controls in ASP.NET Web Forms in the sense that a small amount of code can generate all of the HTML and JavaScript needed for a UI element. (But they do not raise events that you can handle in server code, as is true of Web Forms controls.) Some of the helpers can recognize DataAnnotationsattributes on properties of the model that the view is displaying and format or validate the data accordingly. For example, the following markup is all you need to write in order to generate a drop-down list that includes all of the valid options and displays an error message if the user does not select one of them:

Xml

<div class="editor-label">

@Html.LabelFor(model => model.DepartmentID, "Department")

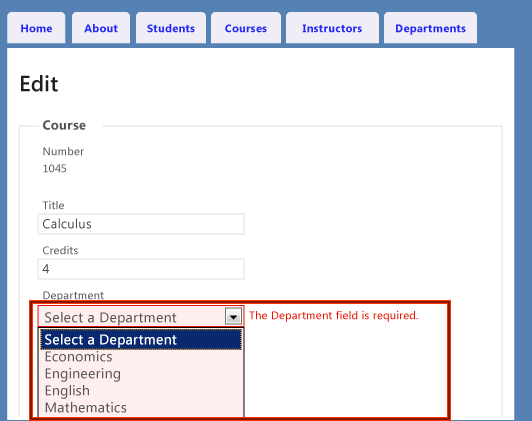
</div>

<div class="editor-field">

@Html.DropDownList("DepartmentID", "Select a Department")

@Html.ValidationMessageFor(model => model.DepartmentID)

</div>



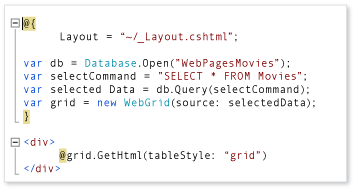
ASP.NET Web Pages and ASP.NET MVC Razor views share the same syntax and much of the same API, so many of the data helpers are the same or similar.

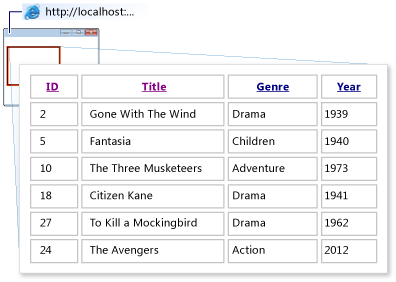
For more information, see the following resources:

* [Intro to ASP.NET MVC 4](http://www.asp.net/mvc/tutorials/mvc-4/getting-started-with-aspnet-mvc4/intro-to-aspnet-mvc-4) (ASP.NET site)
* [Getting Started with EF using MVC](http://www.asp.net/mvc/tutorials/getting-started-with-ef-using-mvc) (ASP.NET site.)
* [ASP.NET MVC 4 Content Map](http://msdn.microsoft.com/en-us/library/gg416514.aspx) (See especially **Working with data and the Entity Framework**.)

[Working with Data in ASP.NET Web Pages Applications](javascript:void(0))

In ASP.NET Web Pages applications, the Database helper provides a quick and easy interface with ADO.NET that enables you to connect to a database and execute SQL queries that you write yourself. For example, the following code uses the **Database** helper to open a SQL Server Compact database and run a query, and then it displays the results in a table by using the**WebGrid** helper:





When you use the **Database** helper you're using ADO.NET directly, without an Object-Relational Mapper. (For information about ORMs, see [Object-Relational Mappers (ORM)](http://msdn.microsoft.com/en-us/library/ms178359.aspx#orm) earlier in this topic.) The ASP.NET Web Pages framework is designed to provide an easy way for people who are new to web programming to get started and build applications that have relatively simple data access requirements. In these scenarios, using ADO.NET directly is a good choice because your data handling needs are not complex enough for the ORM's advantages to outweigh its learning curve.

For more information about the **Database** helper, see the following resources:

* [Introducing ASP.NET Web Pages 2 - Displaying Data](http://www.asp.net/web-pages/tutorials/introducing-aspnet-web-pages-2/displaying-data) (ASP.NET site)
* [Database Class](http://msdn.microsoft.com/en-us/library/webmatrix.data.database(v=vs.99)) (API reference)

If you're a more experienced developer and are building a relatively complex application with complex data access requirements, consider using the Entity Framework. If you do that, use Visual Studio instead of WebMatrix as your development environment. WebMatrix does not include some of the tools that are essential for working with the Entity Framework, such as the Entity Framework designer and the NuGet Package Manager Console.

[Summary of Recommended Choices](javascript:void(0))

|  |  |  |
| --- | --- | --- |
| **Options** | **Recommended Choice(s)** | **More Information** |
| Database Management Systems (DBMS)   * SQL Server * Third-party databases such as Oracle, MySQL, SQLite   (Microsoft Access is not recommended for web applications.) | * SQL Server | [Database Management Systems (DBMS)](http://msdn.microsoft.com/en-us/library/ms178359.aspx#databases) |
| SQL Server Editions   * LocalDB * SQL Server Express * SQL Server Developer Edition * SQL Server (full editions) * Windows Azure SQL Database * SQL Server Compact | Default choices for development:   * For Visual Studio 2012, and SQL Server or SQL Database in production: use LocalDB in development. * For Visual Studio 2010 or WebMatrix, and SQL Server or SQL Database in production: use SQL Server Express in development. * For SQL Server Compact in production: use SQL Server Compact in development. | [SQL Server Editions](http://msdn.microsoft.com/en-us/library/ms178359.aspx#sqlserver) |
| Object-Relational Mappers (ORM)   * ADO.NET Entity Framework * LINQ to SQL * nHibernate | * For ASP.NET Web Forms and MVC: Entity Framework * For ASP.NET Web Pages with WebMatrix: ADO.NET directly (the **Database** helper) | [Object-Relational Mappers (ORM)](http://msdn.microsoft.com/en-us/library/ms178359.aspx#orm)  [Working with Data in ASP.NET Web Pages Applications](http://msdn.microsoft.com/en-us/library/ms178359.aspx#webpages) |
| Entity Framework development workflows | * If you don't need a graphical designer, or the benefits of Code First Migrations outweigh the desire for a designer: Code First * For an existing database when you want to use a graphical designer: Database First * For a new database when you want to use a graphical designer: Model First | [Entity Framework Development Workflows](http://msdn.microsoft.com/en-us/library/ms178359.aspx#dbfmfcf) |
| LINQ versus SQL | Use LINQ whenever possible. | [LINQ versus SQL](http://msdn.microsoft.com/en-us/library/ms178359.aspx#linq) |
| Accessing data through a web service | * On an internal company network: WCF Data Services * Over the Internet: Web API | [Accessing Data Through a Web Service](http://msdn.microsoft.com/en-us/library/ms178359.aspx#webservice) |

[See Also](javascript:void(0))

Other Resources

[ASP.NET Data Access Content Map](http://www.asp.net/whitepapers/aspnet-data-access-content-map)