# Short Answer:

Answer the following questions with complete sentences in your own words. You are encouraged to conduct your own research online or through other methods before answering the questions. If you research online, please consult multiple sources before you write down your answers. You are expected to be able to explain your answers in detail (Provide examples for each question).

1. What is NuGet?

An essential tool for any modern development platform is a mechanism through which developers can create, share, and consume useful code.

Often such code is bundled into "packages" that contain compiled code (as DLLs) along with other content needed in the projects that consume these packages.

For .NET, the Microsoft-supported mechanism for sharing code is NuGet (Pronunciation is similar to “New Get”), which defines how packages for .NET are created, hosted, and consumed and provides the tools for each of those roles.

In short, NuGet is the package manager for .NET. A NuGet package is a single ZIP file with the .nupkg extension containing compiled code (DLLs), other files related to that code, and a descriptive manifest that includes information like the package's version number.

Diagram

Description automatically generated

NuGet is a package manager for the .NET platform. It makes it easier for developers to discover, install, and use packages (libraries and tools) in their .NET projects.

NuGet stores packages in a central repository, called the NuGet Gallery, which is hosted on the nuget.org website. Developers can search for packages in the NuGet Gallery using the NuGet command-line interface (CLI) or the NuGet Package Manager in Visual Studio. Once they find a package they want to use, they can use NuGet to download and install the package into their project.

NuGet also helps developers to manage the dependencies between packages and to update packages to newer versions. It can automatically resolve dependencies between packages and ensure that the correct versions of packages are installed and used in a project.

Overall, NuGet makes it easier for developers to use and manage packages in their .NET projects, helping them to save time and improve the quality of their code.

2. What is ADO.NET, and what are the major components it provides?

ADO.NET

● ADO.NET(ActiveX Data Object.NET) is a data access technology from the Microsoft .NET Framework that provides communication between our application and database.

● ADO.NET provides consistent access to data sources such as SQL Server and XML, and to data sources exposed through OLE DB and ODBC. Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, handle, and update the data they contain.

● Now we are only focusing on RDBMS (Most .NET positions are related to RDBMS, especially in MS SQL Server, non-relational DB is a plus)

ADO.NET Components The two main components of ADO.NET for accessing and manipulating data are

● .NET Framework data providers

○ The .NET Framework Data Providers are components that have been explicitly designed for data manipulation and fast, forward-only, read-only access to data.

○ Ex. Connection, Command, DataReader, DataAdapter

● DataSet

○ The ADO.NET DataSet is explicitly designed for data access independent of any data source. As a result, it can be used with multiple and differing data sources, used with XML data, or used to manage data local to the application.

Graphical user interface

Description automatically generated

ADO.NET (ActiveX Data Objects for .NET) is a data access technology that is part of the .NET framework. It provides a set of classes and interfaces that allow developers to access and manipulate data stored in a database from a .NET application.

The major components that ADO.NET provides are:

1. Connected and Disconnected Data Access: ADO.NET provides both connected and disconnected data access models. The connected model involves maintaining an open connection to the database throughout the lifetime of the data operation, while the disconnected model involves fetching data into a local data store (such as a DataSet object) and working with the data locally.
2. Data Providers: ADO.NET includes a set of data providers that are optimized for different types of databases. These data providers provide the connection, command, and data reader objects that are used to execute SQL queries and stored procedures and to retrieve data from a database.
3. DataSet and DataTable: ADO.NET includes the DataSet and DataTable objects, which are in-memory representations of data. These objects allow developers to work with data locally, without maintaining a connection to the database.
4. Data Adapters: ADO.NET includes data adapters, which are used to populate a DataSet or a DataTable with data from a database. Data adapters also provide the means to update the database with changes made to the data in a DataSet or DataTable.

Overall, ADO.NET provides a set of components that enable developers to access and manipulate data stored in a database from a .NET application, using either a connected or a disconnected data access model.

3. What is DataSet in ADO.NET?

● DataSet

○ The ADO.NET DataSet is explicitly designed for data access independent of any data source. As a result, it can be used with multiple and differing data sources, used with XML data, or used to manage data local to the application.

In ADO.NET, a DataSet is an in-memory representation of data. It is a disconnected, read-only copy of data that is retrieved from a database. The DataSet includes a collection of DataTable objects that contain the data, as well as a set of relationships between the tables.

A DataSet can be used to store and manipulate data in a disconnected manner, which means that it is not necessary to maintain an open connection to the database while working with the data. This can be useful in scenarios where the data needs to be accessed by multiple users or where the data needs to be cached for performance reasons.

The DataSet is also designed to work with XML data, which means that it can be used to serialize its contents to and from an XML document. This allows the data in a DataSet to be easily exchanged with other applications or systems.

Overall, the DataSet is a powerful and flexible object that is useful for working with data in a disconnected manner in ADO.NET applications.

Fill()

adapter.Fill();

● It is used to add rows in the DataSet to match those in the data source.

● no matter the status of connection, after this adapter.Fill(), the connection status will stay the same as before executing this.

● During adapter.Fill(), adapter is the one to control the connection

● Retrieve data and sink to memory at once Manually - faster: conn.Open() adapter.Fill() conn.Close() Automatically: adapter.Fill()

adapter.Fill(datatabe), adapter.Fill(dataset, datatable)

4. What are Data Providers in ADO.NET?

● .NET Framework data providers

○ The .NET Framework Data Providers are components that have been explicitly designed for data manipulation and fast, forward-only, read-only access to data.

○ Ex. Connection, Command, DataReader, DataAdapter

In ADO.NET, data providers are a set of classes and interfaces that provide the means to access and manipulate data stored in a database. Data providers are optimized for specific types of databases and include the connection, command, and data reader objects that are used to execute SQL queries and stored procedures and to retrieve data from a database.

Each data provider is designed to work with a specific type of database, and the classes and interfaces it provides are specific to that database. For example, the SQL Server data provider includes classes and interfaces that are optimized for accessing and manipulating data in a SQL Server database, such as the SqlConnection, SqlCommand, and SqlDataReader classes.

Overall, data providers are an important part of ADO.NET as they provide the means to access and manipulate data stored in a database from a .NET application.

5. What are the steps of database handling via ADO.NET?

Database Handling Via ADO.NET

1) Open Connection

2) Create Command

3) Execute command and obtain result

4) Iterate through the results (DataReader) / directly obtain results (DataAdapter)

5) Close connection

1) Establish Connection to SQL Server

Graphical user interface, text, application, chat or text message

Description automatically generated

You can also

 ● Connect to an OLE DB Data Source via OleDbConnection

● Connect to an ODBC Data Source via OdbcConnection

● Connect to an Oracle Data Source via OracleConnection

Connection String Sql Server Authentication Syntax:

"Persist Security Info=False;User ID=\*\*\*\*\*;Password=\*\*\*\*\*;Initial Catalog=DatabaseName;Server=ServerName”

\*Define within config file

\*We can use other syntax: <https://learn.microsoft.com/enus/dotnet/framework/data/adonet/connection-string-syntax>

Graphical user interface, text, application, email

Description automatically generated

2) Create a Command

SqlCommand with no parameter

Graphical user interface, text, application, email

Description automatically generated

Command

Some important Properties in SqlCommand class:

● Connection ○ Specify which connection to use

● CommandText ○ Sql that command processed \

● CommandType(enum)

○ CommandType.Text – meaning it’s going to execute a sql query

 ○ CommandType.StoredProcedure – meaning it’s going to execute a stored procedure

● Parameters ○ It is used to add the input parameter

● Transaction ○ Success all or fail all

Parameterized Command

Text

Description automatically generated

3)Execute Command

● ExecuteScalar

○ used to execute SQL Command or stored procedure, after executing, return a single value(object) from the database

○ returns the first column of the first row in the result set from a database, additional columns or rows are ignored.

● ExecuteNonQuery

○ used to execute SQL Command or the stored procedure performs INSERT, UPDATE, or DELETE operations.

○ it returns an integer specifying the number of rows inserted, updated or deleted.

○ doesn't return any data from the database

● ExecuteReader

 ○ used to execute a SQL Command or stored procedure returns a set of rows from the database and stores the results in DataReader.

SqlDataReader

● The DataReader provides an unbuffered stream of data that allows procedural logic to efficiently process results from a data source sequentially. The DataReader is a good choice when you're retrieving large amounts of data because the data is not cached in memory.

● The DataReader is read-only, it’s not possible to change the data using DataReader.

● Use the DataReader.Read() method to obtain a row from the query results.

● Always call the Close() method when you have finished using the DataReader object

● While a DataReader is open, the Connection is in use exclusively by that DataReader. You cannot execute any commands for the Connection, including creating another DataReader, until the original DataReader is closed.

4) Iterate through the results (DataReader)

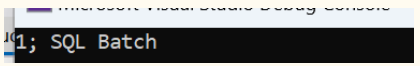
Graphical user interface

Description automatically generated

With Parameter

Text, application

Description automatically generated



5) Close Connection

Just like file I/O, we have to close the database connection after we finish all the jobs

● reader.Close() – if don’t use using()

● conn.Close() – if don’t use using()

The steps of database handling via ADO.NET typically involve the following:

1. Establishing a connection to the database: This involves creating a connection object (such as a SqlConnection in the case of a SQL Server database) and specifying the connection string that contains the information needed to connect to the database.
2. Creating a command object: This involves creating a command object (such as a SqlCommand in the case of a SQL Server database) and specifying the SQL query or stored procedure to be executed.
3. Executing the command: This involves calling the ExecuteReader method of the command object to execute the SQL query or stored procedure and retrieve the resulting data.
4. Processing the data: This involves iterating through the resulting data using a data reader object (such as a SqlDataReader in the case of a SQL Server database) and processing the data as needed.
5. Closing the connection: This involves calling the Close method of the connection object to close the connection to the database.

Overall, these are the basic steps involved in handling a database via ADO.NET. The specific details of each step will depend on the type of database being used and the requirements of the application.

6. What is connection pooling, and how can we enable or disable connection pooling in the application?

Connection Pooling

Connecting to a data source can be expensive and time consuming. To minimize the cost of opening connections, ADO.NET uses an optimization technique called connection pooling, which minimizes the cost of repeatedly opening and closing connections.

● SQL Server Connection Pooling (ADO.NET)

○ Default (whether to use connection pool)

○ Pooling=true

 ○ Max Pool Size

○ Min Pool Size

Graphical user interface, text, application, email

Description automatically generated

Connection pooling is a technique used to improve the performance of database applications by reusing established database connections instead of creating a new connection for each request. When a connection is returned to the pool, it is made available for reuse, which can reduce the overhead of creating new connections and improve the overall performance of the application.

To enable or disable connection pooling in an application, you can specify the "Pooling" attribute in the connection string. To enable connection pooling, set the "Pooling" attribute to "true". To disable connection pooling, set the "Pooling" attribute to "false".

For example, the following connection string enables connection pooling:

"Server=myServerAddress;Database=myDataBase;User Id=myUsername;Password=myPassword;Pooling=true;"

The following connection string disables connection pooling:

"Server=myServerAddress;Database=myDataBase;User Id=myUsername;Password=myPassword;Pooling=false;"

Overall, connection pooling is a useful technique for improving the performance of database applications by reusing established database connections. It can be enabled or disabled in the application by specifying the "Pooling" attribute in the connection string.

1. What’s the purpose of the using() statement?

The using statement is used to set one or more than one resource. These resources are executed and the resource is released. The statement is also used with database operations.

The main goal is to manage resources and release all the resources automatically.

[What is the use of ‘Using’ statement in C#? (tutorialspoint.com)](https://www.tutorialspoint.com/What-is-the-use-of-Using-statement-in-Chash#:~:text=The%20using%20statement%20is%20used%20to%20set%20one,manage%20resources%20and%20release%20all%20the%20resources%20automatically.)

The using statement provides a convenient syntax that ensures the correct use of IDisposable objects.

[using statement - C# Reference | Microsoft Learn](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/using-statement)

IDisposable is an interface that contains a single method, Dispose(), for releasing unmanaged resources, like files, streams, database connections and so on.

Using statement provides a convenient syntax that ensures the correct use of IDisposable objects. Beginning in C# 8.0, the using statement ensures the correct use of IAsyncDisposable objects.

● using statement and using directive - The using statement defines a scope at the end of which an object will be disposed. The using directive creates an alias for a namespace or imports types defined in other namespaces.

The purpose of the using statement in C# is to ensure that a resource is properly disposed when it is no longer needed.

The using statement declares a block of code that uses a resource, and it ensures that the resource is disposed when the block of code is exited. This is useful for resources that implement the IDisposable interface, such as database connections and streams, which need to be closed or released when they are no longer needed.

The using statement has the following syntax:

using (resource)

{

// Code that uses the resource

}

The resource specified in the using statement must be an object that implements the IDisposable interface, and it is typically created in the using statement. When the block of code inside the using statement is exited, the Dispose method of the resource is automatically called to release the resource.

For example, the following code uses a using statement to open a file and read its contents:

using (StreamReader reader = new StreamReader("file.txt"))

{

string content = reader.ReadToEnd();

}

In this example, the StreamReader object is created inside the using statement, and it is automatically disposed when the block of code is exited, ensuring that the file is properly closed.

Overall, the using statement is a useful construct in C# that helps to ensure that resources are properly disposed when they are no longer needed.

8. What are the different execute() methods available in ADO.NET?

There are several execute methods available in ADO.NET that can be used to execute SQL queries and stored procedures:

1. ExecuteReader: This method is used to execute a SQL query or stored procedure and retrieve the resulting data. It returns a data reader object that can be used to iterate through the resulting data.
2. ExecuteScalar: This method is used to execute a SQL query or stored procedure and return a single value. It is useful for queries that return a single value, such as a count or sum.
3. ExecuteNonQuery: This method is used to execute a SQL query or stored procedure that does not return any data, such as an INSERT, UPDATE, or DELETE statement. It returns the number of rows affected by the query.
4. ExecuteXmlReader: This method is used to execute a SQL query or stored procedure and return the resulting data as an XML document. It is useful for queries that return data that needs to be processed as XML.

Overall, these are the main execute methods available in ADO.NET that can be used to execute SQL queries and stored procedures and process the resulting data. The specific execute method to use depends on the requirements of the application and the type of data being returned by the query or stored procedure.

● ExecuteScalar

○ used to execute SQL Command or stored procedure, after executing, return a single value(object) from the database

○ returns the first column of the first row in the result set from a database, additional columns or rows are ignored.

● ExecuteNonQuery

○ used to execute SQL Command or the stored procedure performs INSERT, UPDATE, or DELETE operations.

○ it returns an integer specifying the number of rows inserted, updated or deleted.

○ doesn't return any data from the database

● ExecuteReader

 ○ used to execute a SQL Command or stored procedure returns a set of rows from the database and stores the results in DataReader.

9. What are the differences between DataReader and DataAdapter. Which one would you prefer to use?

DataReader and DataAdapter are two different components in ADO.NET that are used to retrieve data from a database.

The main differences between DataReader and DataAdapter are:

1. Connection model: DataReader uses a connected data access model, which means that it maintains an open connection to the database throughout the lifetime of the data operation. DataAdapter, on the other hand, uses a disconnected data access model, which means that it retrieves data into a local data store (such as a DataSet object) and works with the data locally.
2. Data retrieval: DataReader retrieves data in a forward-only, read-only manner, which means that it can only read the data sequentially and cannot update the data. DataAdapter, on the other hand, can retrieve data in a bidirectional manner, and it can update the data in the database using the data in the local data store.
3. Performance: DataReader is generally faster than DataAdapter because it does not have to populate a local data store with data and it does not maintain a connection to the database for as long.

Which one to use depends on the requirements of the application. If you need to retrieve data in a fast and forward-only manner and do not need to update the data, you can use DataReader. If you need to work with the data in a disconnected manner and update the data, you can use DataAdapter.

Overall, DataReader and DataAdapter are two different components in ADO.NET that are used to retrieve data from a database, and the choice between them depends on the specific requirements of the application.

Table

Description automatically generated

DataAdapter

A DataAdapter is used to retrieve data from a data source and populate tables within a DataSet. The DataAdapter also resolves changes made to the DataSet back to the data source. The DataAdapter uses the Connection object of the .NET Framework data provider to connect to a data source, and it uses Command objects to retrieve data from and resolve changes to the data source.

Graphical user interface, text, application, email

Description automatically generated

SqlDataReader

● The DataReader provides an unbuffered stream of data that allows procedural logic to efficiently process results from a data source sequentially. The DataReader is a good choice when you're retrieving large amounts of data because the data is not cached in memory.

● The DataReader is read-only, it’s not possible to change the data using DataReader.

● Use the DataReader.Read() method to obtain a row from the query results.

● Always call the Close() method when you have finished using the DataReader object

● While a DataReader is open, the Connection is in use exclusively by that DataReader. You cannot execute any commands for the Connection, including creating another DataReader, until the original DataReader is closed.

4) Iterate through the results (DataReader)

Graphical user interface

Description automatically generated

With Parameter

Text, application

Description automatically generated

Shape, rectangle

Description automatically generated

5) Close Connection

Just like file I/O, we have to close the database connection after we finish all the jobs

● reader.Close() – if don’t use using()

● conn.Close() – if don’t use using()

10. How to invoke stored procedures via ADO.NET?

Text

Description automatically generated

[ADO.NET Using Stored Procedure in C# - Dot Net Tutorials](https://dotnettutorials.net/lesson/ado-net-using-stored-procedure/)

To invoke a stored procedure via ADO.NET, you can use the following steps:

1. Establish a connection to the database: This involves creating a connection object (such as a SqlConnection in the case of a SQL Server database) and specifying the connection string that contains the information needed to connect to the database.
2. Create a command object: This involves creating a command object (such as a SqlCommand in the case of a SQL Server database) and specifying the stored procedure to be executed. You can set the CommandType property of the command object to "StoredProcedure" to indicate that the command is a stored procedure.
3. Specify the parameters for the stored procedure: If the stored procedure has any input or output parameters, you can use the Parameters collection of the command object to specify the values for these parameters.
4. Execute the stored procedure: You can use the ExecuteReader, ExecuteScalar, or ExecuteNonQuery method of the command object to execute the stored procedure and retrieve the resulting data.
5. Process the data: Depending on the method used to execute the stored procedure, you can use a data reader object, a scalar value, or the number of rows affected to process the data returned by the stored procedure.
6. Close the connection: When you are finished working with the database, you can call the Close method of the connection object to close the connection.

Here is an example of how to invoke a stored procedure called "GetEmployee" via ADO.NET:

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

using (SqlCommand command = new SqlCommand("GetEmployee", connection))

{

    command.CommandType = CommandType.StoredProcedure;

    // Set the input parameter for the stored procedure

    command.Parameters.AddWithValue("@EmpID", empID);

    // Execute the stored procedure and retrieve the data

    using (SqlDataReader reader = command.ExecuteReader())

    {

        // Process the data

        while (reader.Read())

        {

            Console.WriteLine("Employee ID: {0}", reader["EmpID"]);

            Console.WriteLine("Name: {0}", reader["Name"]);

            Console.WriteLine("Department: {0}", reader["Department"]);

        }

    }

}

}

In this example, the stored procedure "GetEmployee" is invoked with an input parameter of "empID", and the resulting data is retrieved using a data reader and processed. The connection is then closed when the using block is exited.

● CommandType(enum)

○ CommandType.Text – meaning it’s going to execute a sql query

○ CommandType.StoredProcedure – meaning it’s going to execute a stored procedure

3)Execute Command

● ExecuteScalar

○ used to execute SQL Command or stored procedure, after executing, return a single value(object) from the database

○ returns the first column of the first row in the result set from a database, additional columns or rows are ignored.

● ExecuteNonQuery

○ used to execute SQL Command or the stored procedure performs INSERT, UPDATE, or DELETE operations.

○ it returns an integer specifying the number of rows inserted, updated or deleted.

○ doesn't return any data from the database

● ExecuteReader

 ○ used to execute a SQL Command or stored procedure returns a set of rows from the database and stores the results in DataReader.

11. What is a transaction in ADO.NET?

Transaction in ADO.NET Transaction represents a single unit of work.

Transactions in ADO.NET are used when you want to bind multiple tasks together so that they execute as a single unit of work.

Transaction control is performed by the Connection object,

You can initiate a local transaction with the BeginTransaction method.

Once you have begun a transaction, you can enlist a command in that transaction with the Transaction property of a Command object.

You can then commit or roll back modifications made at the data source based on the success or failure of the components of the transaction

Text

Description automatically generated

Diagram

Description automatically generated

In ADO.NET, a transaction is a group of database operations that are treated as a unit and either completed in their entirety or rolled back if an error occurs. Transactions allow you to ensure the consistency and integrity of the data in a database by ensuring that all the operations in the transaction either succeed or fail together.

To use transactions in ADO.NET, you can use the following steps:

1. Establish a connection to the database: This involves creating a connection object (such as a SqlConnection in the case of a SQL Server database) and specifying the connection string that contains the information needed to connect to the database.
2. Begin the transaction: This involves calling the BeginTransaction method of the connection object to start a new transaction. This returns a Transaction object that represents the transaction.
3. Execute the database operations: You can use the usual ADO.NET techniques, such as creating command objects and executing SQL queries or stored procedures, to perform the database operations as part of the transaction.
4. Commit or roll back the transaction: After all the database operations have been completed, you can call the Commit method of the Transaction object to commit the transaction and make the changes permanent, or you can call the Rollback method to roll back the transaction and undo the changes.

Overall, transactions are a useful feature of ADO.NET that allow you to ensure the consistency and integrity of the data in a database by grouping database operations into a unit that either succeeds or fails together.

12. Explain ACID Properties.

 The ACID properties describes the transaction management well.

• Atomicity: all queries in a transaction must succeed. If one fails, all should rollback.

• Consistency: the database must be consistent before and after the transaction.

• Isolation: multiple Transactions occur independently without interference.

• Durability: committed transaction must be persisted in a durable storage(database).

ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties that are used to describe the behavior of a database transaction. These properties ensure that database transactions are reliable and predictable, and they are designed to protect the integrity of the data in the database.

The ACID properties are as follows:

1. Atomicity: Atomicity refers to the "all-or-nothing" nature of a transaction. This means that either all the operations in a transaction are completed successfully, or none of them are completed. There is no intermediate state.
2. Consistency: Consistency refers to the requirement that a transaction must leave the database in a consistent state. This means that if a transaction starts in a valid state, it must end in a valid state, even if it is rolled back.
3. Isolation: Isolation refers to the requirement that the execution of a transaction must be isolated from the execution of other transactions. This means that the effects of a transaction are not visible to other transactions until the transaction is committed.
4. Durability: Durability refers to the requirement that the changes made by a committed transaction must be permanent, even in the event of a system failure.

Overall, the ACID properties are an important concept in database systems, as they ensure the reliability and predictability of transactions and protect the integrity of the data in the database.

13. What is Connected Architecture, and what is Disconnected Architecture?

Diagram

Description automatically generated

Graphical user interface

Description automatically generated

Using DataReader is connected architecture.

Using Data Adapter along with DataTable and DataSet for separation/ independent database with application is disconnected architecture.

Connected architecture and disconnected architecture are two different approaches to accessing and manipulating data in a database.

Connected architecture is an approach in which the application maintains an open connection to the database throughout the lifetime of the data operation. This means that the application can communicate with the database and retrieve data from it as needed.

Connected architecture has the following characteristics:

* It requires an open connection to the database at all times.
* It allows for real-time data access and manipulation.
* It is generally faster than disconnected architecture because it does not have to transfer data between the application and the database.

Disconnected architecture, on the other hand, is an approach in which the application retrieves data from the database and stores it in a local data store (such as a DataSet object). The application can then work with the data locally and update the database later.

Disconnected architecture has the following characteristics:

* It does not require an open connection to the database at all times.
* It allows for offline data access and manipulation.
* It is generally slower than connected architecture because it involves transferring data between the application and the database.

Overall, connected architecture and disconnected architecture are two different approaches to accessing and manipulating data in a database, and the choice between them depends on the specific requirements of the application.

# Coding Questions:

Write code in c# to solve the following problems. Please write your own answers. You are highly encouraged to present more than one way to answer the questions. Please follow best practices when you write the code so that it is easily readable, maintainable, and efficient. Clearly state your assumptions if you have any. You may discuss with others on the questions, but please write your own code.

There is a pizza shop in East Windsor that requires some CRUD operations to be implemented using ADO.NET(try to use both DataReader and DataAdapter to implement the functions below);

* 1. Create a Pizza on the menu
  2. Read all Pizzas from the shop
  3. Read a Pizza given its unique Id
  4. Update a Pizza given its unique Id
  5. Delete a Pizza by its unique Id
  6. Create an UD-Stored Procedure called “The Signiture Menu” to get TOP3 expensive Pizzas and execute the SP using your .NET application

The domain of pizza has the following properties:

int Id {get; set;} string Name {get; set;} float Price {get; set;}

A SQL script has been provided for this assignment. Please run the script and base your implementation on it.