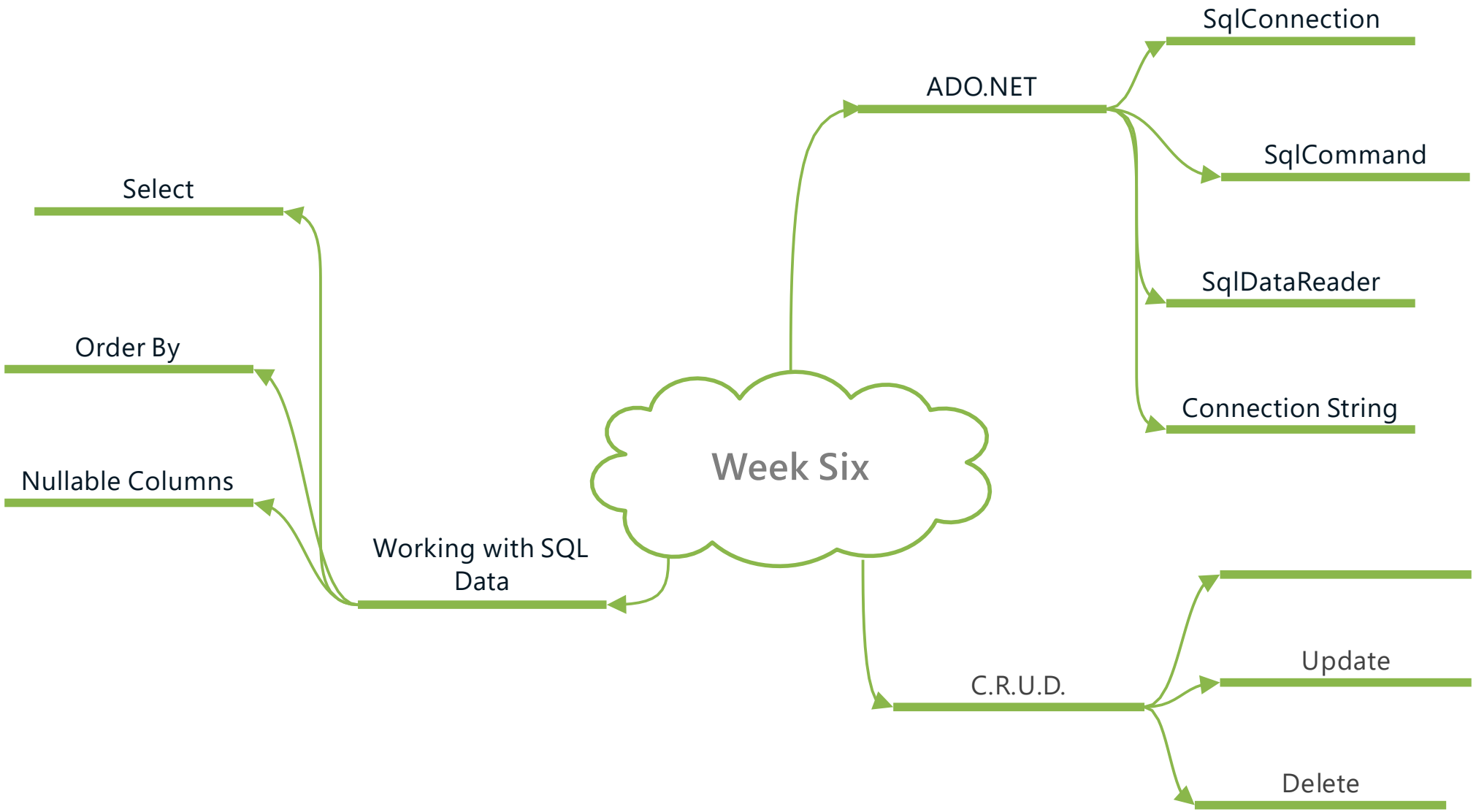
COMP 3602

C# Application Development

Week Six

Tonight's Learning Outcomes



Database Server Communication

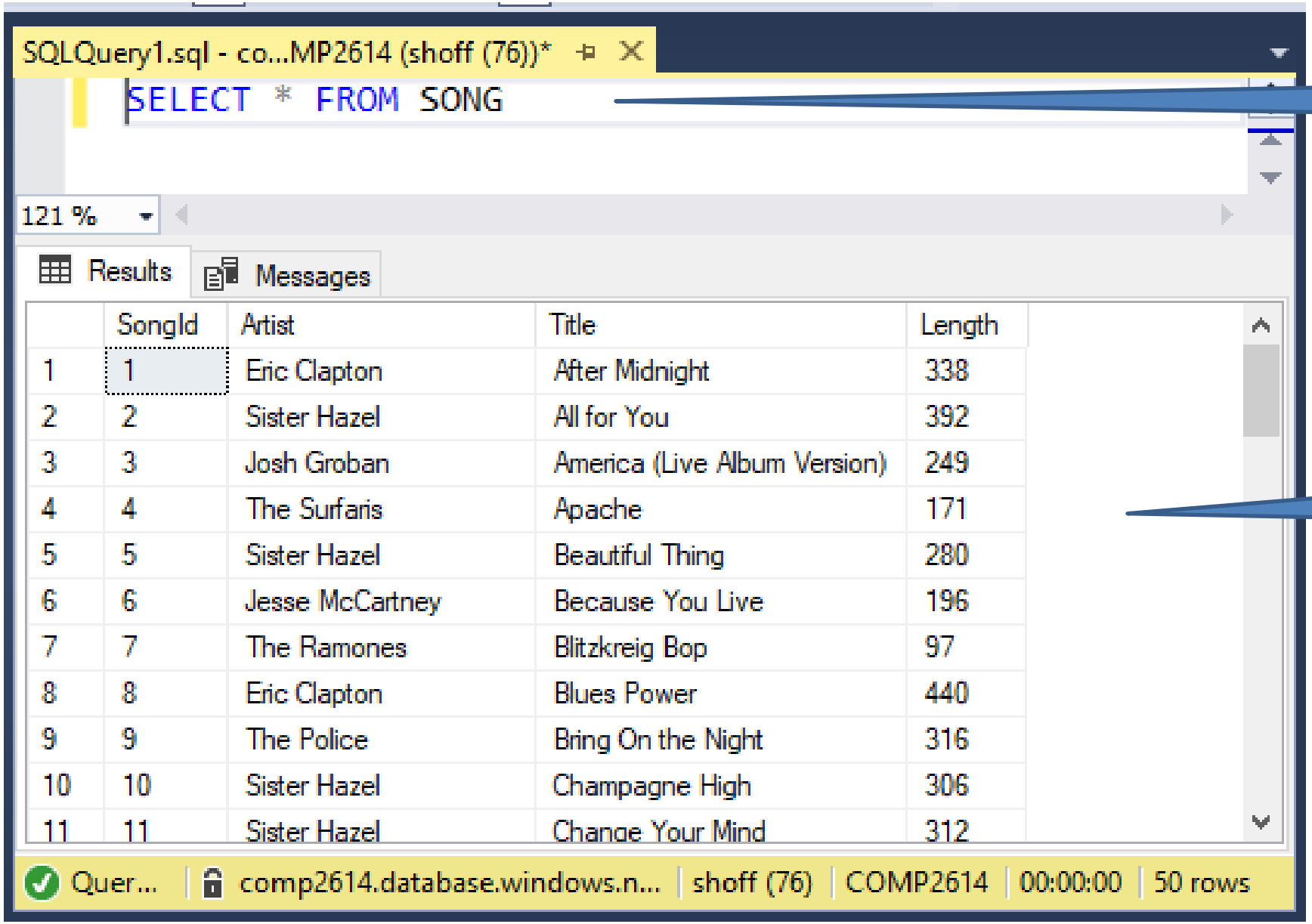
Client sends query to server

```
1  SELECT CustomerId, CompanyName, ContactName, ContactTitle
2   FROM Customers
3   WHERE Country = 'Canada'
4
```

Server sends result set to client

	CustomerId	CompanyName	ContactName	ContactTitle
1	BOTTM	Bottom-Dollar Markets	Janet Wilke	Marketing Manager
2	LAUGB	Laughing Bacchus Wine Cellars	Yoshi Tannamuri	Marketing Assistant
3	MEREP	Mère Paillarde	Jean Fresnière	Marketing Assistant

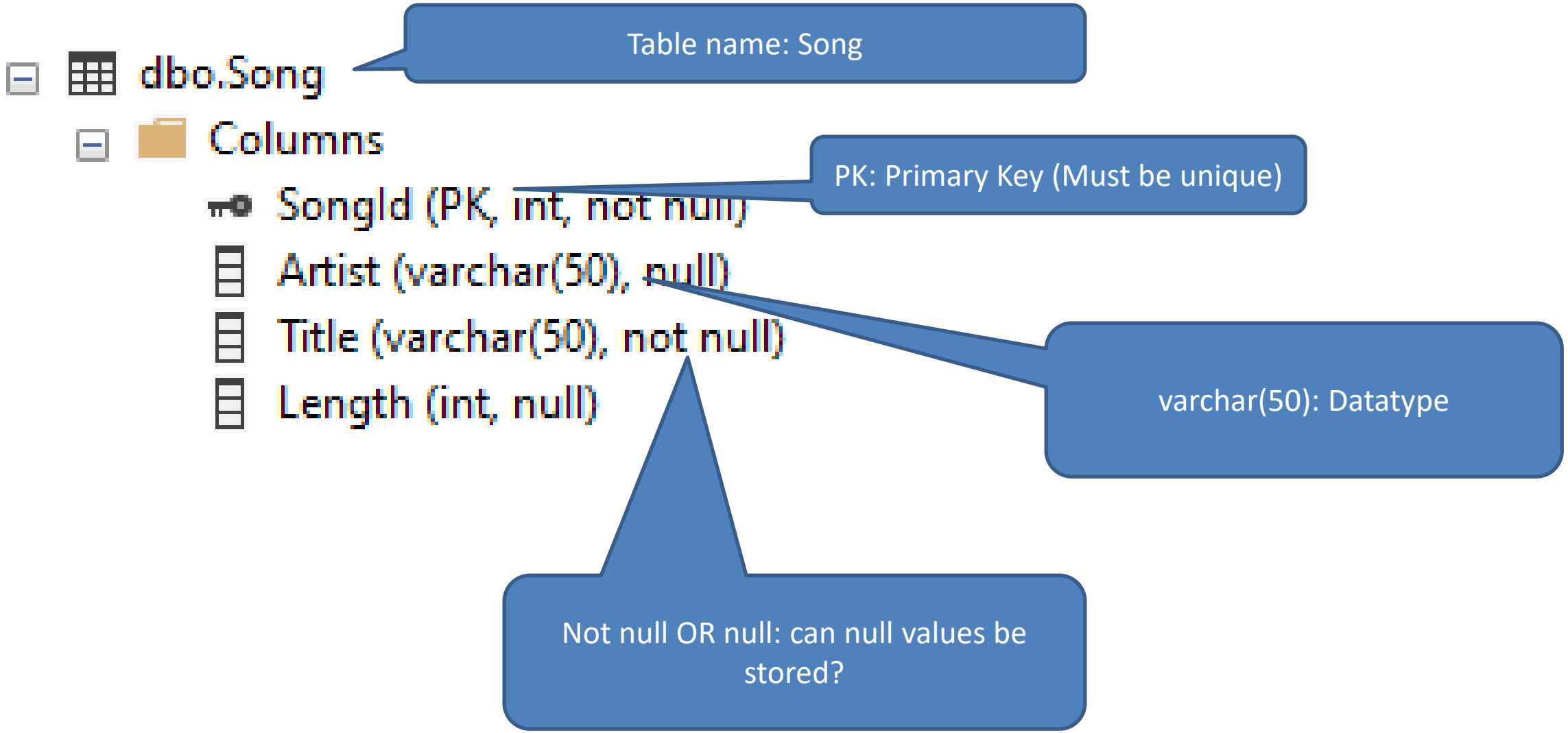
SQL Table (SSMS)



SQL Query

Result

SQL Table Definition



ADO.NET Classes

The System.Data.SqlClient namespace contains several classes designed to interact with databases including:

Class Name	Description
SqlConnection	Establishes a connection with the database server
SqlCommand	Creates the query to send to the database server
SqlDataReader	Reads and iterates through the resultset

Connection String

The Connection String provides the parameters required to connect to and log on the database server. It is a semi-colon delimited list of name/value pairs.

```
14
15      private static readonly string connString = @"Server=tcp:skeena.database.windows.net,1433;
16                                                    Initial Catalog=comp2614;
17                                                    User ID=student;
18                                                    Password=p8SmM5/mKZk=;
19                                                    Encrypt=True;
20                                                    TrustServerCertificate=False;
21                                                    Connection Timeout=30;";
22
```

Connecting to the Database Server

- The SqlConnection class is used to connect to the database. One of its constructor overloads takes a connection string as a parameter. This provides all the connection and authentication data.

- Declaring and instantiating the SqlConnection object in a “using” block will automatically close and dispose the object at the end of the block. This syntax can be used with any class that implements the IDisposable interface.

```
29
30      using (SqlConnection conn = new SqlConnection(connString))
31      {
32          // processing code
33      }
34
```

Creating the Query

Instantiate a SqlCommand object, set some properties and open the connection

```
29      // embedded SQL
30      string query = @"SELECT SongId, Artist, Title, Length
31              FROM Song
32              ORDER BY Artist, Title";
33
34      using (SqlCommand cmd = new SqlCommand())
35      {
36          cmd.CommandType = CommandType.Text;
37          cmd.CommandText = query;
38          cmd.Connection = conn;
39
40          conn.Open();
41
```

SQL Query

Specify the CommandType, CommandText (query), and the Connection.
Then Open the Connection

Connect → Send Command → Read Result

Processing the ResultSet

```
songs = new SongList();
using (SqlDataReader reader = cmd.ExecuteReader())
{
    string artist = null;
    string title;
    int length = 0;

    while (reader.Read())
    {
        //No need to null check strings, can use "as" keyword
        artist = reader[0] as string;

        //NOT NULL column in SQL anyhow, can't be null
        title = reader["Title"] as string;

        //May be null in DB, need to null check before casting
        if (!reader.IsDBNull(2))
        {
            length = (int)reader["Length"];
        }

        songs.Add(new Song(artist, title, length));

        artist = null;
        length = 0;
    }
}
return songs;
```

Executes Query

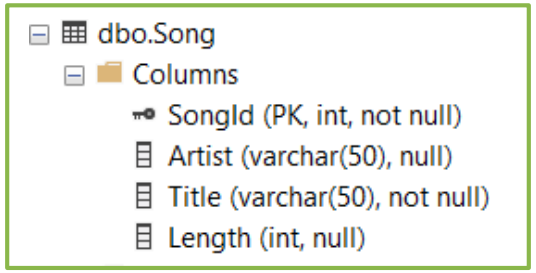
The Read method traverses the resultset in a forward direction only.

SqlDataReader resultset data are untyped (object) so casting is required.

Test for DbNull with nullable fields. Must refer to its position in the result set

Field data can be referred to by name [string] or ordinal position [int]

Table Column Specifications



SQL Server to C# Data Type Mappings

SQL Server Type	C# Type
varchar / nvarchar	string
char / nchar	string
text / ntext	string
bigint	long
int	int
smallint	short
tinyint	byte
float	double
real	float
Decimal(28,4)	decimal
money	decimal
bit	bool
datetime	DateTime

- The SQL Server datatypes are mostly ANSI standard compliant and do not match the C# type names.
- Here is a list of mappings of the most common types

Query Parameters

```
147     // embedded SQL
148     string query;
149
150     if (artistFilter.ToUpper() == "ALL")
151     {
152         query = @"SELECT SongId, Artist, Title, Length
153                   FROM Song
154                   ORDER BY Artist, Title";
155     }
156     else
157     {
158         query = @"SELECT SongId, Artist, Title, Length
159                   FROM Song
160                   WHERE Artist = @artist
161                   ORDER BY Artist, Title";
162     }
163
164     using (SqlCommand cmd = new SqlCommand())
165     {
166         cmd.CommandType = CommandType.Text;
167         cmd.CommandText = query;
168         cmd.Connection = conn;
169         cmd.Parameters.AddWithValue("@artist", artistFilter);
170
171         conn.Open();
172
```

A parameter is prefaced with an '@' in the query string

Call the AddWithValue method of the Parameters collection in the command object. Pass it the parameter name and value to use in the query

C.R.U.D – Create (Insert)

SQL Code

```
1  INSERT INTO Region
2    (RegionId, RegionDescription)
3    VALUES (5, 'Central')
4
```

- The Insert statement is used to add records to a data table.
- The Insert statement includes a comma separated field list followed by a comma separated value list.

Embedded SQL Code in Application

```
26
27      string query = @"INSERT INTO Product123456
28                        (Quantity, Sku, Description, Cost, Taxable)
29                         VALUES (@quantity, @sku, @description, @cost, @taxable)";
30
31
```

C.R.U.D. - Update

SQL Code

```
1  UPDATE Customers
2    SET ContactName = 'Janet Wilke'
3      , ContactTitle = 'Marketing Manager'
4    WHERE CustomerId = 'BOTTM'
5
```

Embedded SQL Code in Application

```
78
79      string query = $@"UPDATE Product123456
80                           SET Quantity = @quantity,
81                               Sku = @sku,
82                               Description = @description,
83                               Cost = @cost,
84                               Taxable = @taxable
85                           WHERE ProductId = @productId";
86
```

- You can modify values in a record using the Update statement.
- Use the Set keyword with a comma separated list of Field = Value.
- Use a Where clause to limit the number of records affected.

C.R.U.D. - Delete

SQL Code

```
1  DELETE Region
2   WHERE RegionID = 5
3
```

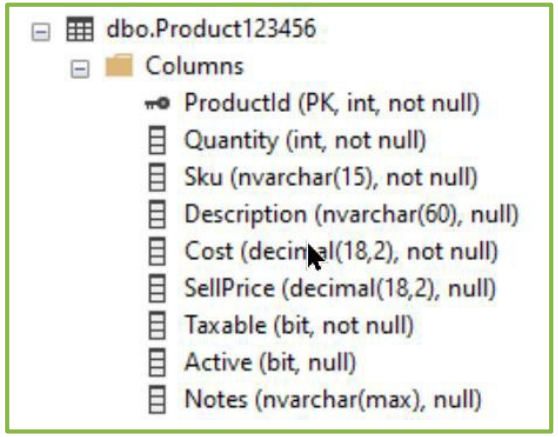
- The Delete statement is used to remove records from a data table.
- Use a Where clause to specify the record or records to be deleted.

Embedded SQL Code in Application

```
171
172         string query = @"DELETE Product123456
173                           WHERE ProductId = @productId";
174
```

Parameters for Inserts and Updates

- A parameter is required for each field in the database table
- Value types can be assigned directly to each parameter regardless of the nullability of the associated column
- Reference types (string) can be assigned directly to each parameter of non-nullable columns
- Reference types (string) can only be assigned directly to the parameter of nullable columns when the string is not null. Pass DBNull.Value to the parameter when the string is null




```
181
182     cmd.Parameters.AddWithValue("productId", product.ProductId);
183     cmd.Parameters.AddWithValue("quantity", product.Quantity);
184     cmd.Parameters.AddWithValue("sku", product.Sku);
185     cmd.Parameters.AddWithValue("description", (object)product.Description ?? DBNull.Value);
186     cmd.Parameters.AddWithValue("cost", product.Cost);
187     cmd.Parameters.AddWithValue("sellPrice", product.SellPrice);
188     cmd.Parameters.AddWithValue("taxable", product.Taxable);
189     cmd.Parameters.AddWithValue("active", product.Active);
190     cmd.Parameters.AddWithValue("notes", (object)product.Notes ?? DBNull.Value);
191
```

Null Conditional ? And Null Coalescing ?? Operators

Allows a cleaner, simpler syntax when working with nulls

```
16
17      string name = "Melissa";
18
19      int nameLength;
20
21      nameLength = name.Length; // could throw an exception
22
23      if (name != null) // null test before Length call
24      {
25          nameLength = name.Length;
26      }
27      else
28      {
29          nameLength = 0;
30      }
31
32      // with ? and ?? operators
33      nameLength = name?.Length ?? 0;
34
```

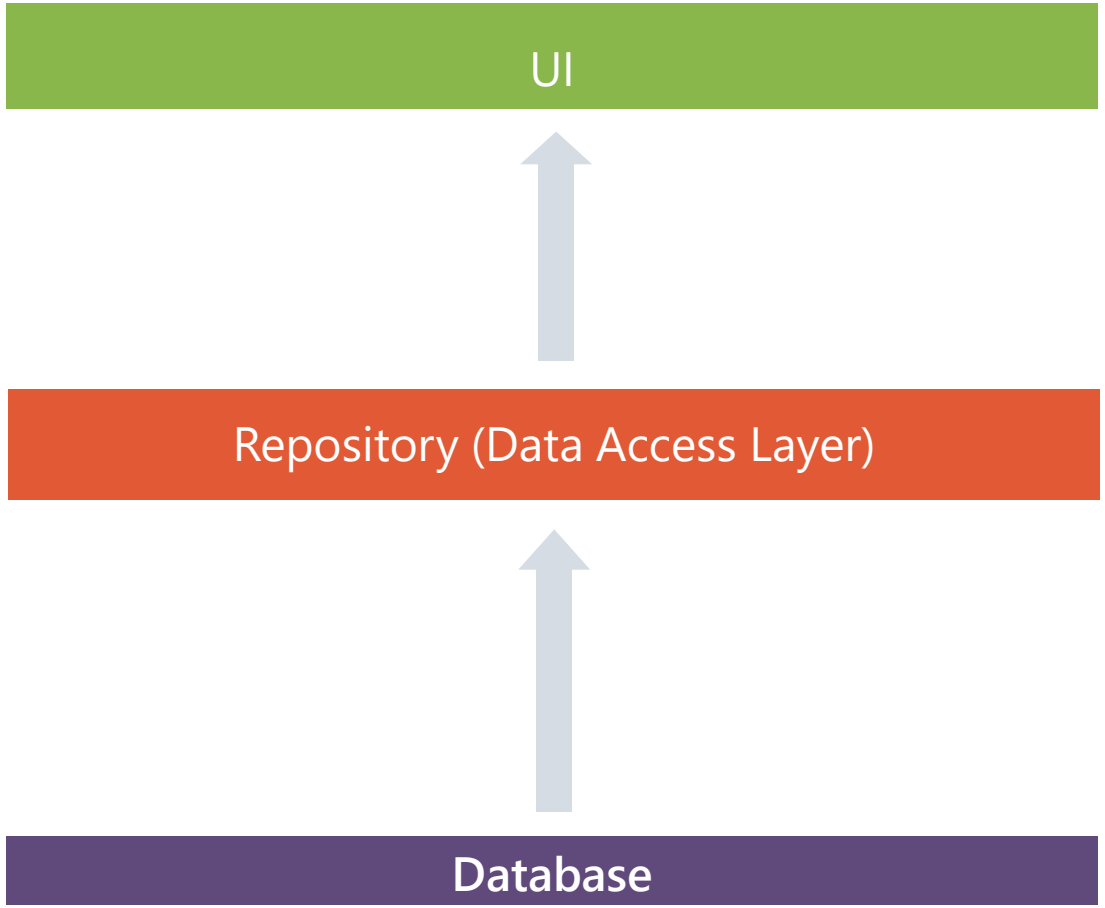
Nullable Types

Any value type can be made nullable by adding a ? to the end of the type name

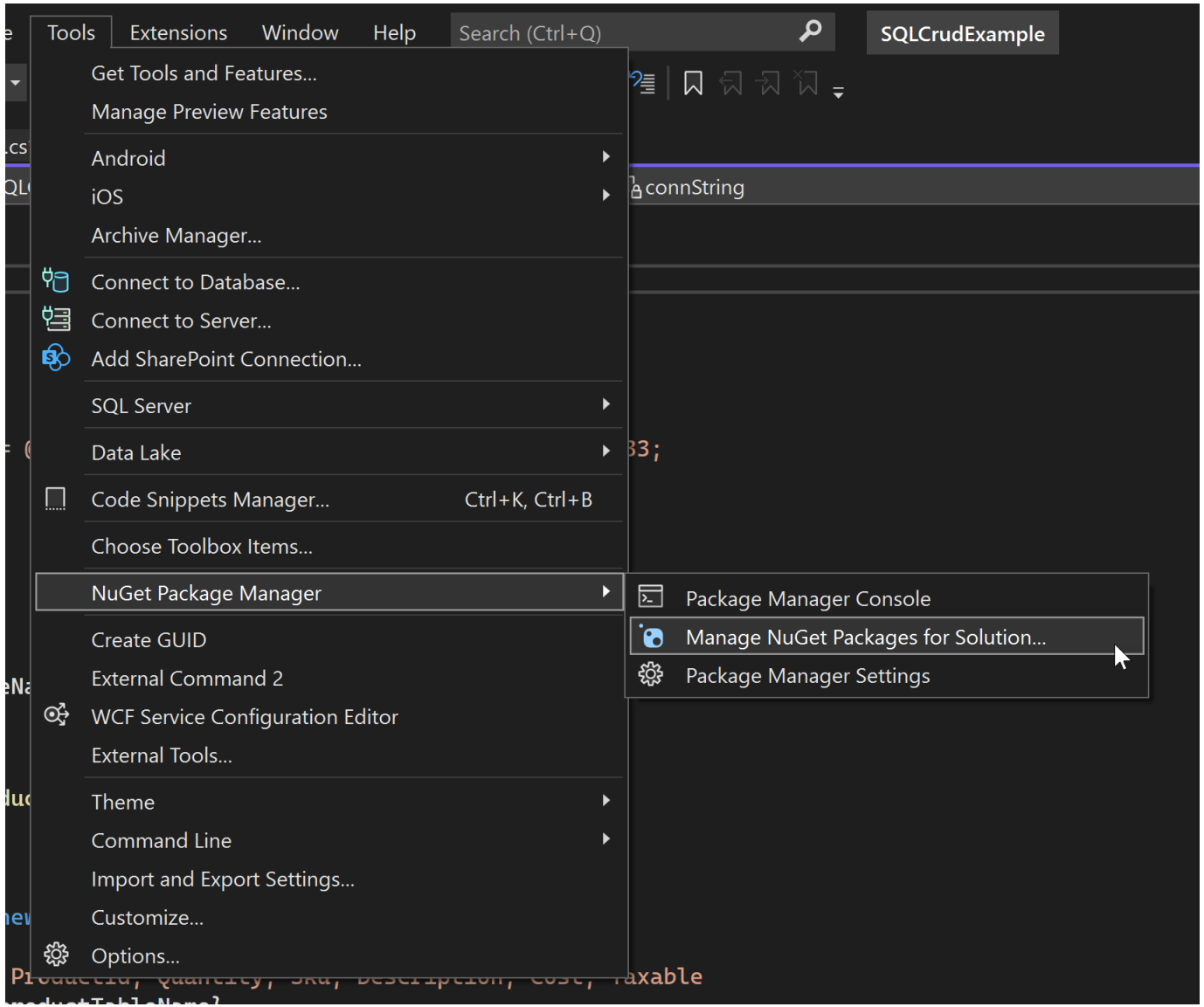
```
            sellPrice = reader["SellPrice"] as decimal?;

            cost = (decimal)reader["Cost"];
            taxable = (bool)reader["Taxable"];

            products.Add(new Product
            {
                ProductId = productId,
                Quantity = quantity,
                Sku = sku,
                Description = description,
                Cost = cost,
                Taxable = taxable,
                SellPrice = sellPrice ?? 0.0m
            });
        }
    }
```

- Nullable types are reference types
- Value type is boxed

The start of a layered architecture

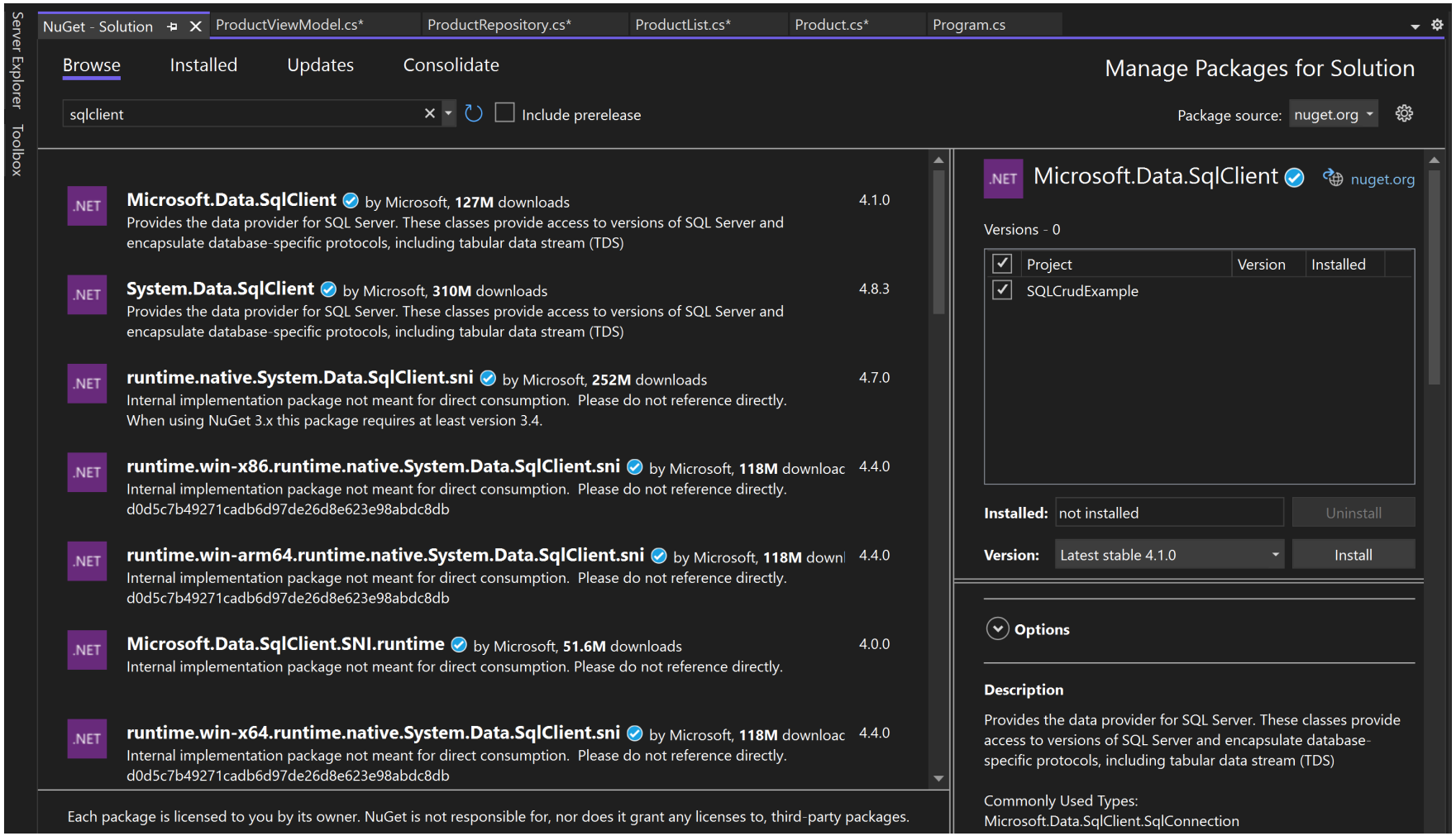


.Net Core or .Net 5/6 – Microsoft.Data.SQLClient



- Need to get package and add to solution
- Tools → NuGet Package Manager → Manage NuGet Packages for Solution

.Net Core or .Net 5/6 – Microsoft.Data.SQLClient



- Search for and install Microsoft.Data.SqlClient
- Need to select the project with a checkbox

.Net Core or .Net 5/6 – Microsoft.Data.SQLClient