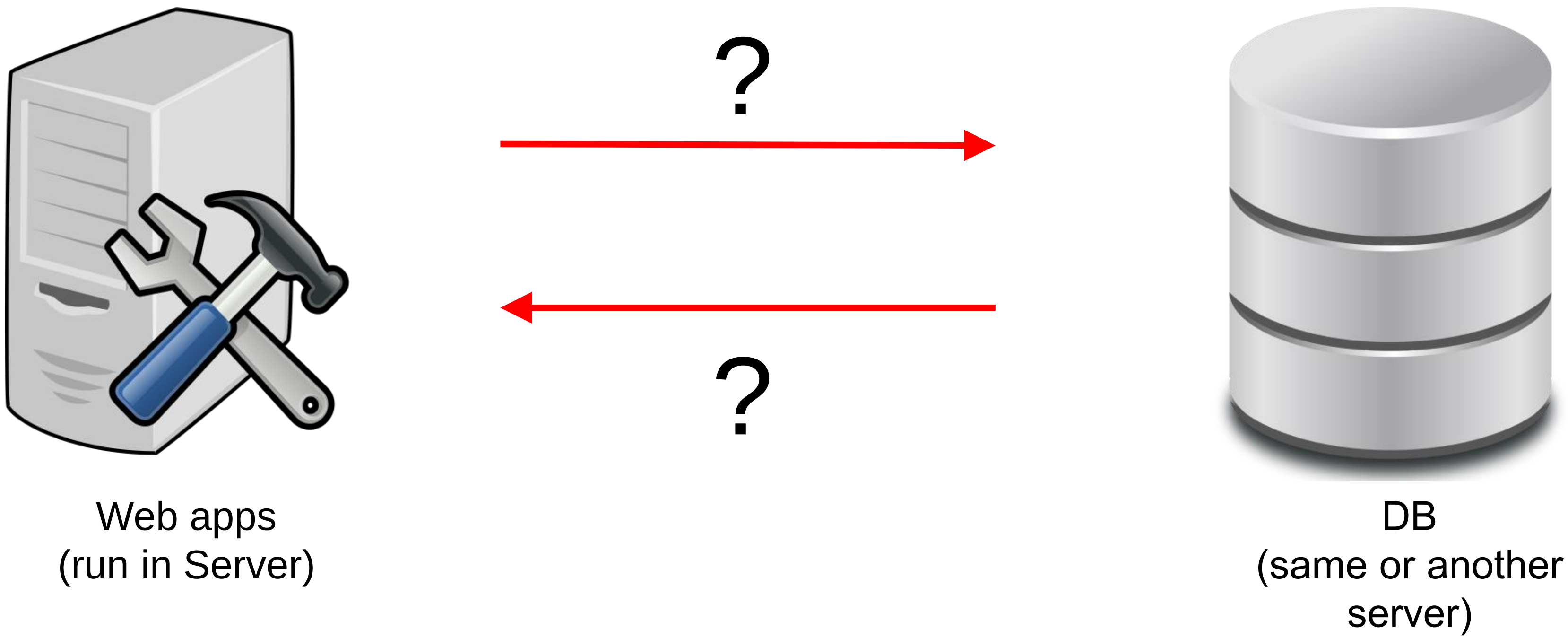
ASP.NET MVC

ADO.NET

issntt@nus.edu.sg

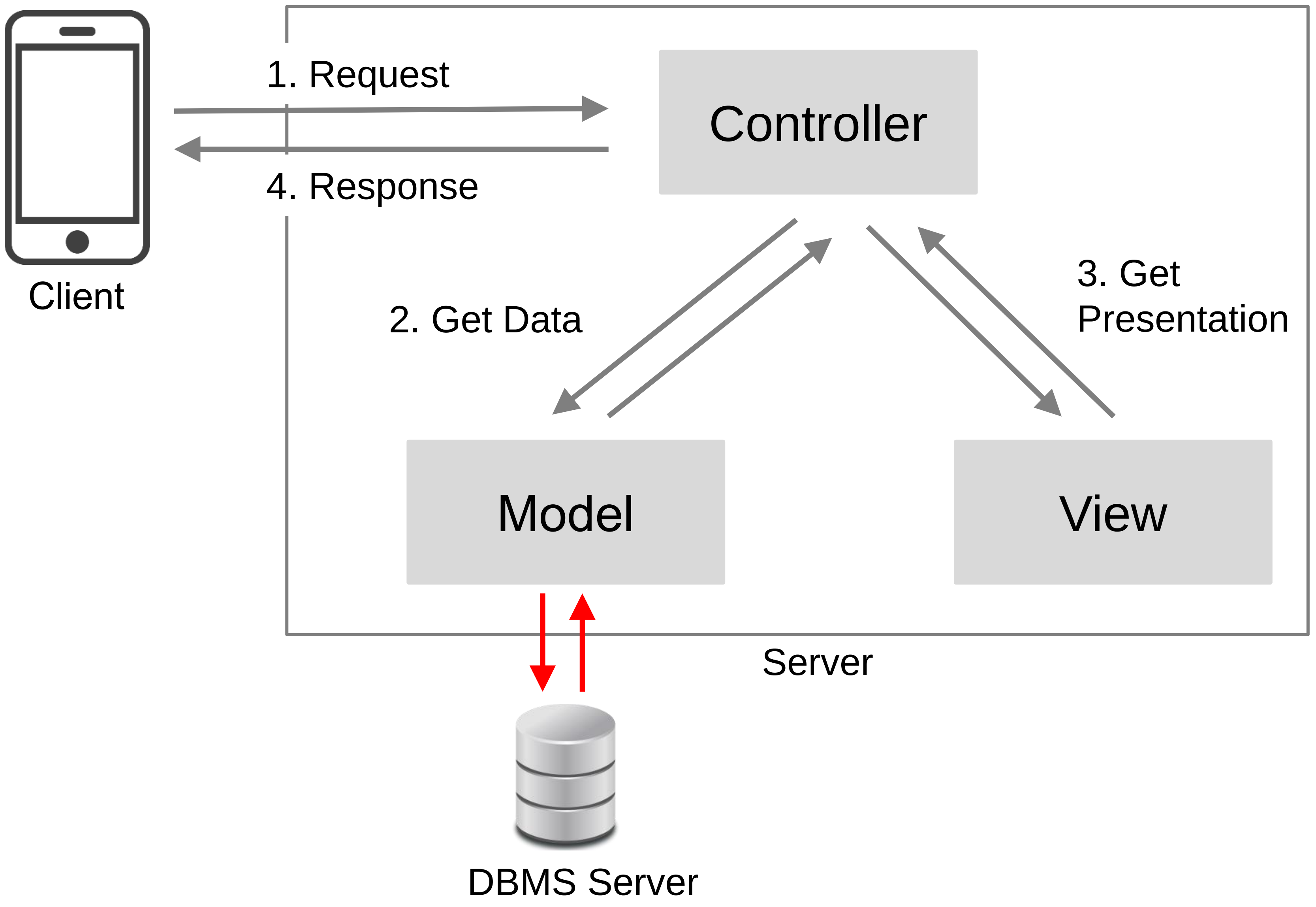
What concepts have we learnt in ASP.NET Core so far?

Problem



How can our web apps **store / receive data** to / from database?

Problem – More specifically



Options

	Option 1	Option 2
Data Access Technology	ADO.NET	Entity Framework
Data Query Language	SQL	LINQ

Objectives

At the end of this lesson, students will be able to

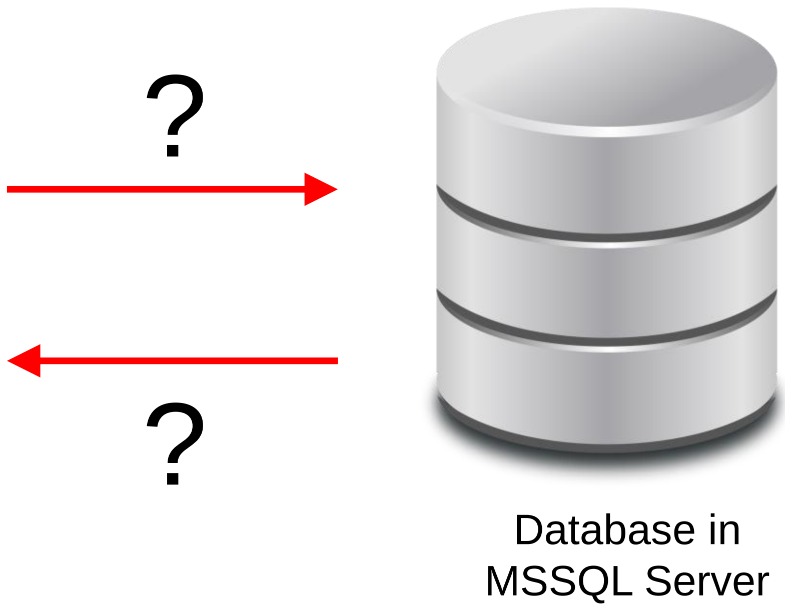
- Describe the common steps to query data from a database
- Describe the scenarios that we should use C# `using` statement
- Describe some scenarios that hackers can use SQL Injection to attack a web application, and apply appropriate mechanisms to prevent them
- Describe the transaction concept and how we can use them to ensure the atomicity in our web apps
- Design and implement the data access layer using ADO.NET key components

Topics

- **Querying Database**
- ADO.NET Overview
- SQLConnection
- SQLCommand
- SQLParameter
- SQLDataReader
- SQLTransaction

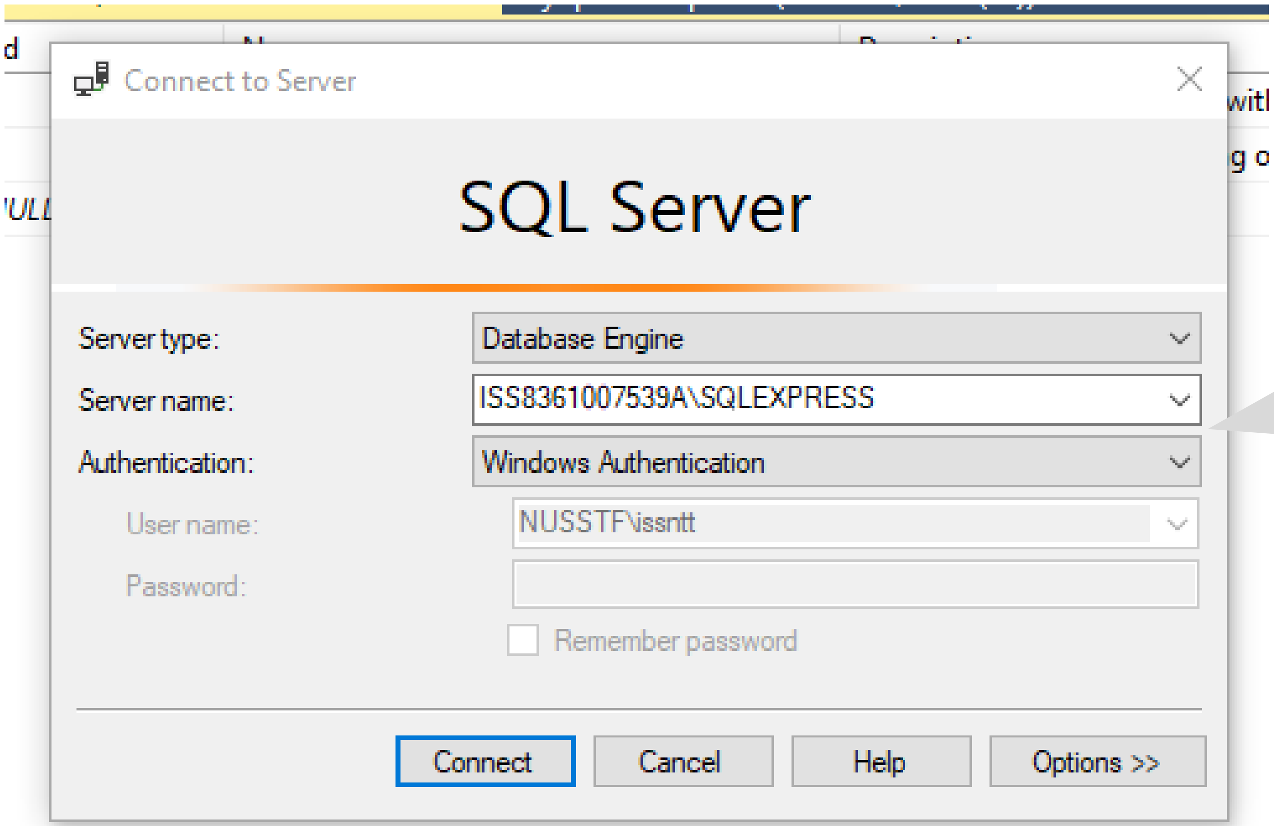
Review Question

In the SQL Programming course, how do we **query data** from a database in MS SQL Server?



Review - Querying Database

We use Microsoft SQL Server Management Studio

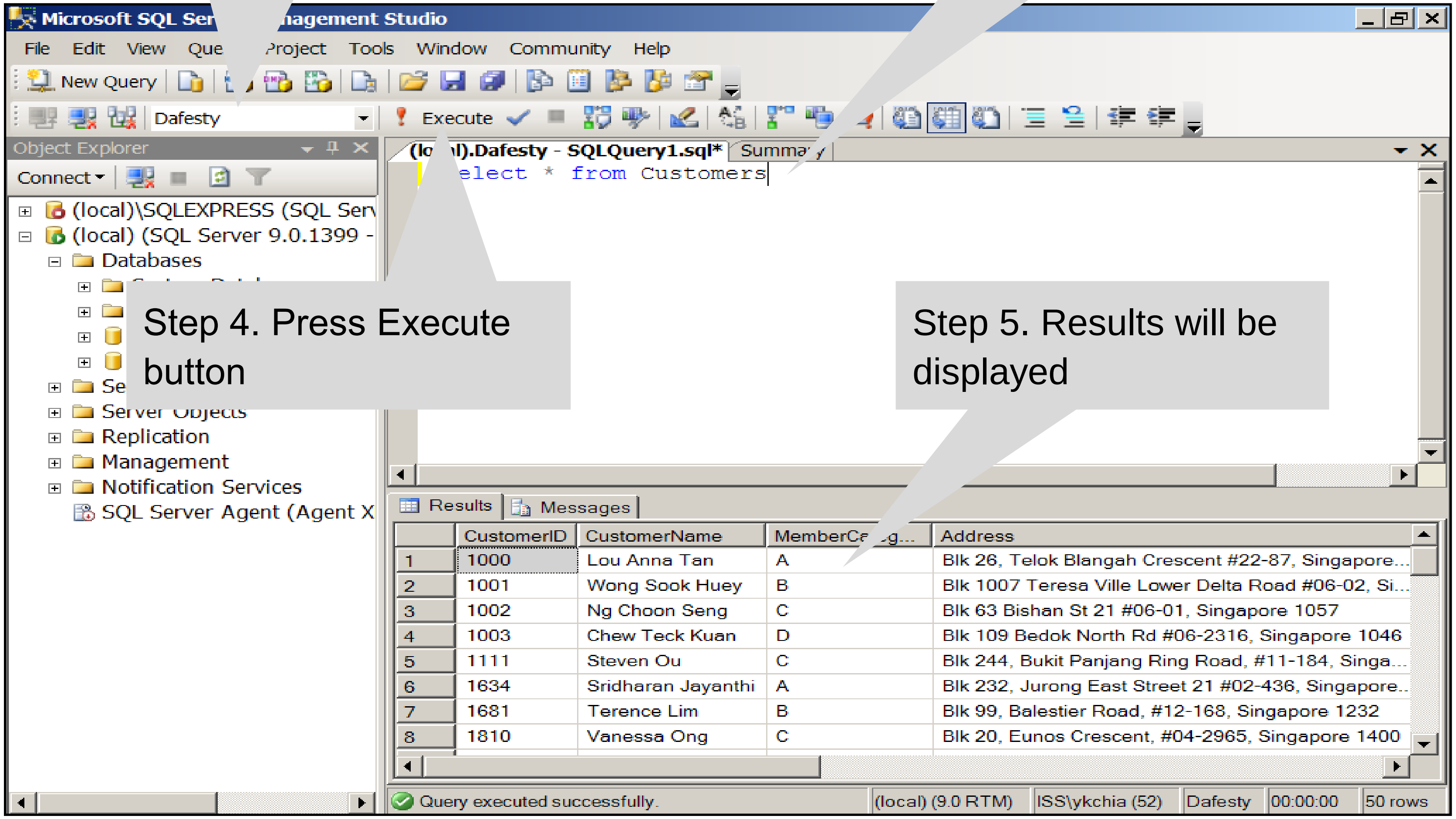

Connect to Server

SQL Server

Server type:	Database Engine
Server name:	ISS8361007539A\SQLEXPRESS
Authentication:	Windows Authentication
User name:	NUSSTF\issntt
Password:	
	Remember password

Step 1. Connect to SQL Server

Review - Querying Database

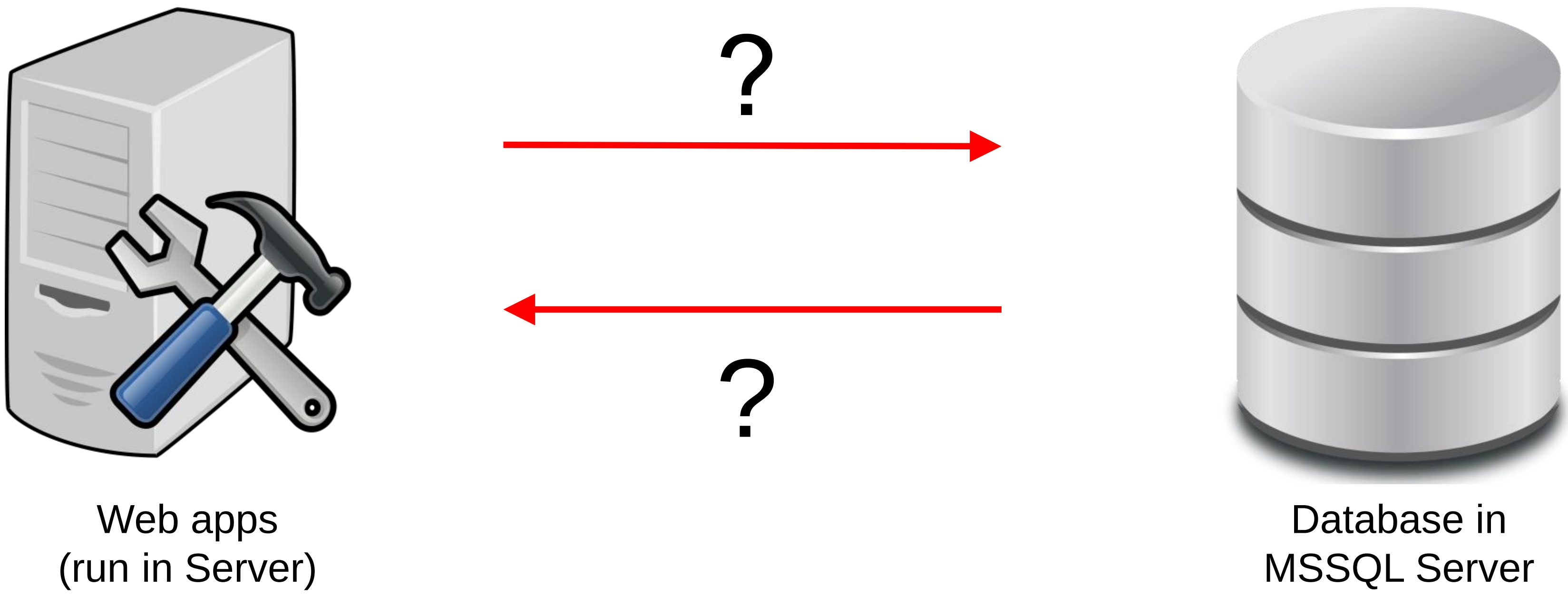

Step 2. Select database to be queried
Step 3. Input the query
Step 4. Press Execute button
Step 5. Results will be displayed
Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Community Help
New Query
Dafesty
Execute
Object Explorer
Connect
(local)\SQLEXPRESS (SQL Serv
(local) (SQL Server 9.0.1399 -
Databases
Server Objects
Replication
Management
Notification Services
SQL Server Agent (Agent X
(local).Dafesty - SQLQuery1.sql*
Summary
Results
Messages

```
select * from Customers
```

	CustomerID	CustomerName	MemberCategory	Address
1	1000	Lou Anna Tan	A	Blk 26, Telok Blangah Crescent #22-87, Singapore...
2	1001	Wong Sook Huey	B	Blk 1007 Teresa Ville Lower Delta Road #06-02, Si...
3	1002	Ng Choon Seng	C	Blk 63 Bishan St 21 #06-01, Singapore 1057
4	1003	Chew Teck Kuan	D	Blk 109 Bedok North Rd #06-2316, Singapore 1046
5	1111	Steven Ou	C	Blk 244, Bukit Panjang Ring Road, #11-184, Singa...
6	1634	Sridharan Jayanthi	A	Blk 232, Jurong East Street 21 #02-436, Singapore...
7	1681	Terence Lim	B	Blk 99, Balestier Road, #12-168, Singapore 1232
8	1810	Vanessa Ong	C	Blk 20, Eunos Crescent, #04-2965, Singapore 1400

Query executed successfully. | (local) (9.0 RTM) | ISS\ykchia (52) | Dafesty | 00:00:00 | 50 rows

Problem



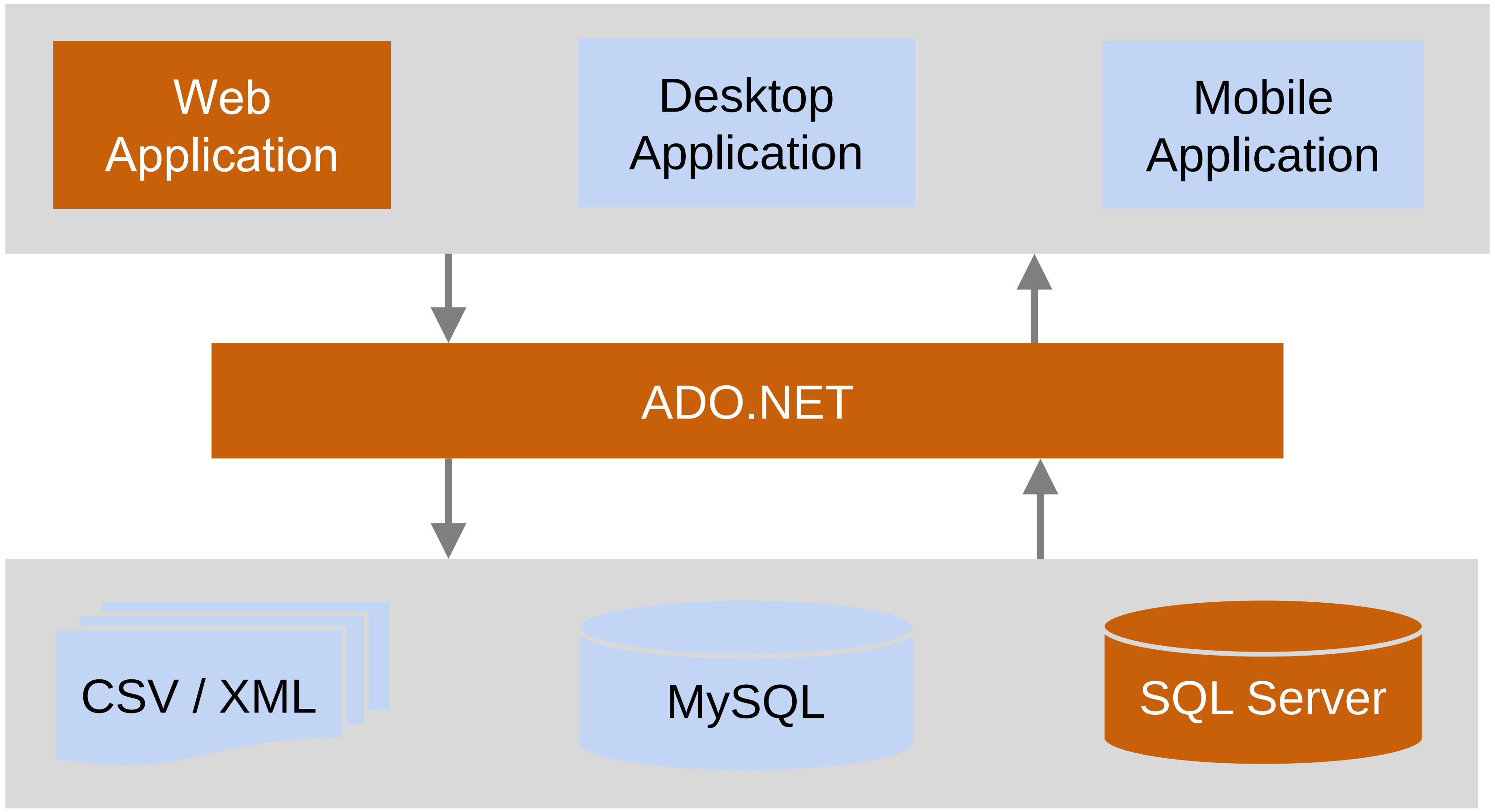
Those steps are for **human**. How **similar** and **different** do you think they are for .NET (code)?

Topics

- Querying Database
- **ADO.NET Overview**
- SQLConnection
- SQLCommand
- SQLParameter
- SQLDataReader
- SQLTransaction

ADO.NET

ADO.NET is a set of classes that expose **data access service** for .NET developers



An example

Following is a sample usage of ADO.NET

```
string connectionString = 2
    @"Server=ISS8361007539A\SQLEXPRESS;Database=adoLecture; Integrated Security = true";
public Employee GetEmployee(int? id) {
   Employee emp = null;
   using (SqlConnection conn = new SqlConnection(connectionString)) {
   1  conn.Open();
      string sql = @"SELECT * FROM tblEmployee WHERE ID = " + id;
   3  SqlCommand cmd = new SqlCommand(sql, conn);
      SqlDataReader reader = 4 cmd.ExecuteReader();
      while (reader.Read()) {
         emp = new Employee() {
            ID = (int)reader["ID"],
            Name = (string)reader["Name"],                      5
            Gender = (string)reader["Gender"],
            Department = (string)reader["Department"],
            City = (string)reader["City"]
         };
      }
      conn.Close();
      return emp;
   }
}
```

ADO.NET Key Components

Purpose	Components
1. To connect to MS SQL Server	**SqlConnection**
2. To select a database	**SqlConnection**
3. To input a query	**SqlCommand** **SqlParameter**
4. To execute the query	**SqlCommand**
5. To retrieve the result data	**SqlDataReader**
6. Others	**SqlTransaction** (to ensure atomicity)

Namespace: Microsoft.Data.SqlClient

ADO.NET Key Components

- **SqlConnection**
  - **Connects** to the SQL Server
  - Creates a Transaction object
- **SqlCommand**
  - Placeholder for SQL statements
  - Binds SQL statements to a Transaction object
  - **Executes** SQL statements
- **SqlParameter**
  - **Binds parameters** to SqlCommand
  - Use it to prevent **SQL Injection**

ADO.NET Key Components

- **SqlDataReader**
  - Holds on to an **opened connection**
  - **Read** the returned **results**
- **SqlTransaction**
  - Represents a SQL transaction
  - Created from SqlConnection’s BeginTransaction()
  - Attach it to a SqlCommand for execution

Topics

- Querying Database
- ADO.NET Overview
- **SQLConnection**
    - **C# *using* statement**
- SQLCommand
- SQLParameter
- SQLDataReader
- SQLTransaction

SQLConnection

This class **connects** to a SQL Server database. Its object represents a **unique session** to the DB server

```
using (SqlConnection conn = new
                              SqlConnection(connectionString)) {
   conn.Open();
   // Code is omitted for brevity
   conn.Close();
}
```

Connection String

`SqlConnection` string requires **Server Name** (as in step 1), **Database Name** (step 2) and **Authentication**

```
string connectionString =
      1 "Server=ISS8361007539A\\SQLEXPRESS;" +
      2 "Database=adoExample; " +
      3 "Integrated Security=true";

using (SqlConnection conn
           = new SqlConnection(connectionString))
{
   conn.Open();
   // Code is omitted for brevity
   conn.Close();
}
```

Integrated Security means using **Windows Account** credentials, i.e., our **current** login credentials

Next

```
using (SqlConnection conn = new
                    SqlConnection(connectionString))
{
   conn.Open();
   // Code is omitted for brevity
   conn.Close();
}
```

What is `using` statement? Why are we using it?

Do you remember `finally` block

When **using** a **resources** such as a network connection, we need to **release/dispose** it eventually

```
SqlConnection conn = null;
try {
   SqlConnection conn = new SqlConnection(connectionString);
   conn.Open();
   // using the resource conn
} finally
{
   if (conn != null)
      conn.Dispose();
}
```

No matter if try block throws any **exception**, we need to **dispose** the **resource**

Do you remember `finally` block

That code pattern is **repeated** every time when any type of resource is used

```
StreamReader myResource = null;
try {
   myResource = new StreamReader("File1.txt");
   // Using the resource
} finally
{
   if (myResource != null)
      myResource.Dispose();
}
```

Is there any way to write **less code**?

Keyword `using`

Ensure that the resource object is **automatically disposed** as soon as it goes out of scope

```
using (SqlConnection conn = new
                    SqlConnection(connectionString))
{
   conn.Open();
   // Code is omitted for brevity
   conn.Close(); // Code is optional
}
```

Some conditions when we use `using() {}` statement

- The `Dispose()` method for objects initiated in () will be automatically called
- Only objects that inherit from `IDisposable` can be declared inside (), because it has `Dispose()` method
- Due to scope, only code within {} can access objects created within ()

Topics

- Querying Database
- ADO.NET Overview
- SQLConnection
- **SQLCommand**
- SQLParameter
- SQLDataReader
- SQLTransaction

SQLCommand

SQLCommand **encapsulates** SQL statement(s)

```
using (SqlConnection conn = new SqlConnection(connectionString))
{
   conn.Open();

   string sql = @"SELECT * FROM tblEmployee";
   SqlCommand cmd = new SqlCommand(sql, conn);
   SqlDataReader reader = cmd.ExecuteReader();

   while (reader.Read()) {
       // retrieving results
   }
   conn.Close();
}
```

SQLCommand

Provide methods to **return results** for **different types** of SQL statements. *`ExecuteNonQuery()`* is for **non-query**

```
string sql = @"DELETE FROM Course WHERE CourseId = " + id;
SqlCommand cmd = new SqlCommand(sql, conn);

int noAffectedRows = cmd.ExecuteNonQuery();
```

SQLCommand

ExecuteScalar() is for queries that return only **one value** such as *COUNT*, *AVG*, *SUM*

```
string sql = @"SELECT COUNT(*) FROM Course";
SqlCommand cmd = new SqlCommand(sql, conn);

int noCourses = (int) cmd.ExecuteScalar();
```

SQLCommand

ExecuteReader() for **common queries**, returning an *SQLDataReader* object to retrieve data

```
string sql = @"SELECT ID, Code, Name FROM Course";
SqlCommand cmd = new SqlCommand(sql, conn);

SqlDataReader reader = cmd.ExecuteReader();
```

Topics

- Querying Database
- ADO.NET Overview
- SQLConnection
- SQLCommand
- **SQLParameter**
  - **SQL Injection**
- SQLDataReader
- SQLTransaction

SQL Injection

https://www.youtube.com/watch?v=G6t1HxgTyfg

SQLParameter

SqlParameter is to **prevent** SQL Injection**.** Using queries with *SqlParameter* is a **3-step process**

1

Construct the SQL string with parameters

2

Declare a *SqlParameter* object, assign values as appropriate

3

Add the *SqlParameter* object to the *SqlCommand*

SQLParameter

```
string sql = @"
      INSERT INTO tblEmployee (Name, Gender, Department)
      VALUES (1 @Name, @Gender, @Department)";

SqlParameter param1 = 2 new SqlParameter {
   ParameterName = "@Name",
   Value = emp.Name
};
SqlParameter param2 = new SqlParameter {
   ParameterName = "@Gender",
   Value = emp.Gender
};
SqlParameter param3 = new SqlParameter {
   ParameterName = "@Department",
   Value = emp.Department
};

SqlCommand cmd = new SqlCommand(sql, conn);
cmd.Parameters.3 Add(param1);
cmd.Parameters.Add(param2);
cmd.Parameters.Add(param3);

cmd.ExecuteNonQuery();
```

This *emp.Name* value is provided by the **client**

SQLParameter

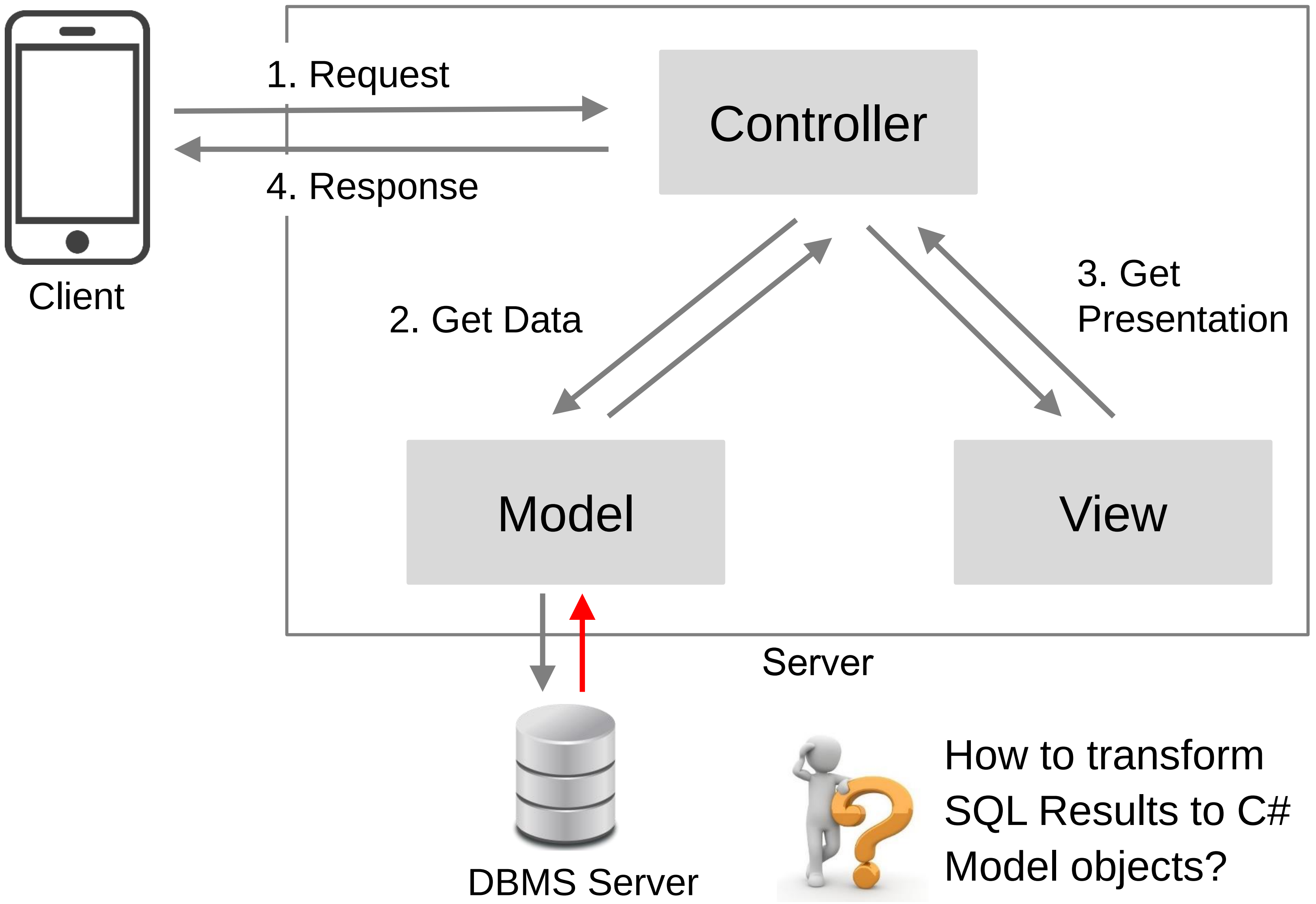
Alternatively, we can **combine** step 2 and step 3 for writing **less** code

```
string sql = @"
   INSERT INTO tblEmployee (Name, Gender, Department, City)
   VALUES (① @Name, @Gender, @Department, @City)";

SqlCommand cmd = new SqlCommand(sql, conn);
cmd.Parameters.③AddWithValue(② "@Name", emp.Name);
cmd.Parameters.AddWithValue("@Gender", emp.Gender);
cmd.Parameters.AddWithValue("@Department", emp.Department);
cmd.Parameters.AddWithValue("@City", emp.City);

cmd.ExecuteNonQuery();
```

Next



How to transform SQL Results to C# Model objects?

Topics

- Querying Database
- ADO.NET Overview
- SQLConnection
- SQLCommand
- SQLParameter
- **SQLDataReader**
- SQLTransaction

SQLDataReader

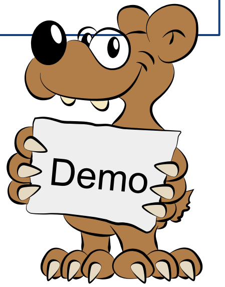
`SQLDataReader` provides a **forward-only access** that enables developers to **iterate** the query result

```
string sql = @"SELECT ID, Code, Name AS CourseName FROM Course";
SqlCommand cmd = new SqlCommand(sql, conn);

List<Course> courseList = new List<Course>();
SqlDataReader reader = cmd.ExecuteReader();
while (reader.Read()) {
   Course myCourse = new Course() {
      ID = (string)reader["ID"],
      Code = (string)reader["Code"],
      Name = (string)reader["CourseName"]
   };
   courseList.Add(myCourse);
}
```

ID	Code	CourseName
01	FOPCS	Fundamental of Programming using C#
02	OOPCS	Object Oriented Programming in C#

Query result



Topics

- Querying Database
- ADO.NET Overview
- SQLConnection
- SQLCommand
- SQLParameter
- SQLDataReader
- **SQLTransaction**
  - **Transactions**
  - **Commit and Rollback**

Problem

When Alice transfers $50 to Bob

Step 1. Reduce $50 from Alice’s account

Step 2. Increase $50 to Bob’s account

What may be **issues** of this scenario?

Transactions

- A transaction is a **logical unit of work**
  - even though **multiple steps** are required to accomplish it
- Transactions are mainly to achieve:
  - **Atomicity (all** or **nothing)**
  - **Consistent state** in database

Image by mohamed Hassan from Pixabay

How to achieve them?

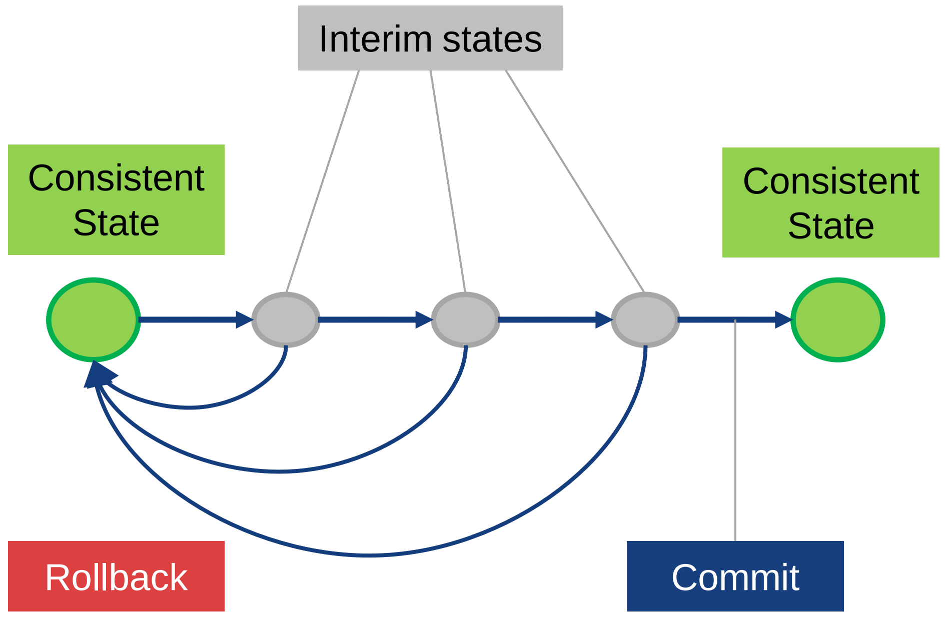
Mechanisms of Transactions

Commit

- **All** the necessary **steps** have been done **successfully**, so
- **All** the **changes** are **saved**

Rollback

- **At least one** of the necessary steps **cannot** be done successfully, so
- **All** the **changes** need to be **discarded**



SQLTransaction

We can execute **multiple queries** in a *SQLTransaction*

```
using (SqlConnection conn = new SqlConnection(connectionString)) {
   conn.Open();
   SqlTransaction trans = conn.BeginTransaction();
   SqlCommand cmd = new SqlCommand("", conn, trans);

   try {
      cmd.CommandText = @"UPDATE Accounts SET Balance = Balance - 50
                                             WHERE Holder = 'Alice'";
      cmd.ExecuteNonQuery();

      cmd.CommandText = @"UPDATE Accounts SET Balance = Balance + 50
                                             WHERE Holder = 'Bob'";
      cmd.ExecuteNonQuery();

      trans.Commit();
   } catch (Exception ex) {
      Debug.WriteLine("Some error with DB: " + ex.Message);
      trans.Rollback();
   }
}
```

This simplified solution is for demo purpose only

Readings

- ASP.NET Core 2.0: CRUD Operation With ADO.NET https://social.technet.microsoft.com/wiki/contents/articles/51324.asp-net-core-2-0-crud-operation-with-ado-net.aspx
- SQL Injection https://www.w3schools.com/sql/sql_injection.asp