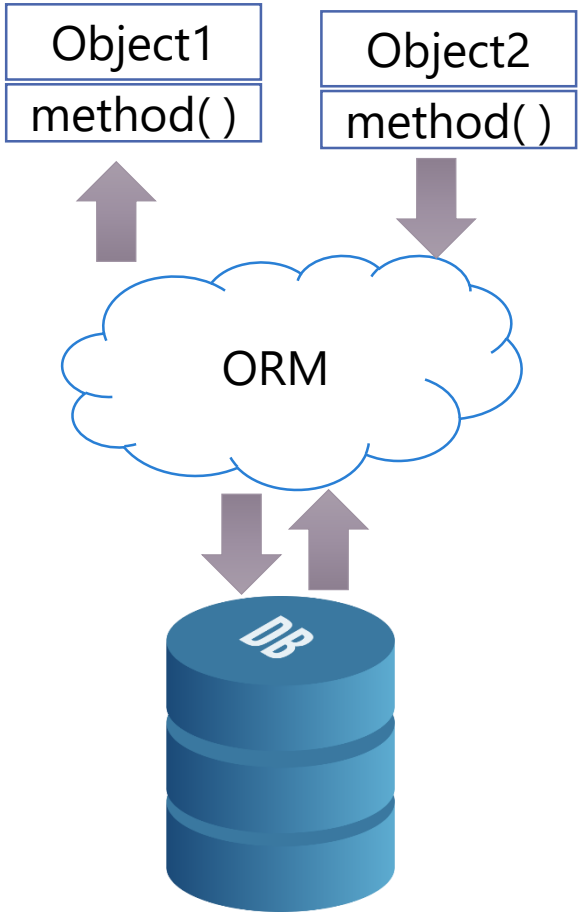
JOHN ABBOTT
CEGEP/COLLEGE

(420-PS4-AB)
Entity Framework Introduction

Summer 2018

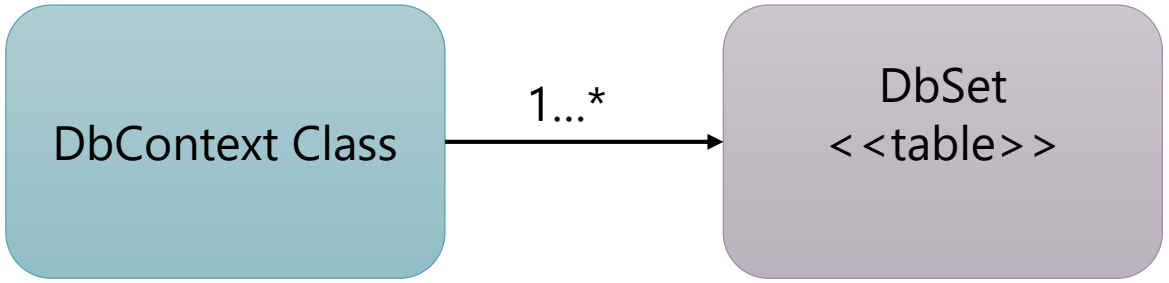
Entity Framework is an ORM

- The .NET Framework provides support for Object Relation Mapping (ORM)

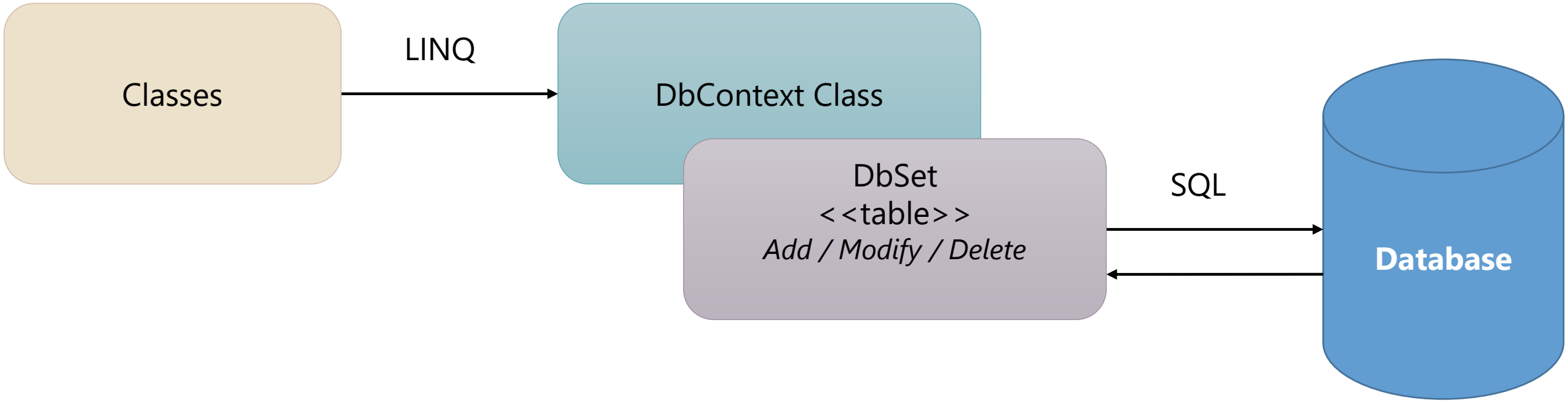


- Features
  - Automatically generate necessary SQL code
  - Map result sets to strongly typed objects
  - Persist object changes back to a database
  - Implicit support for transactions

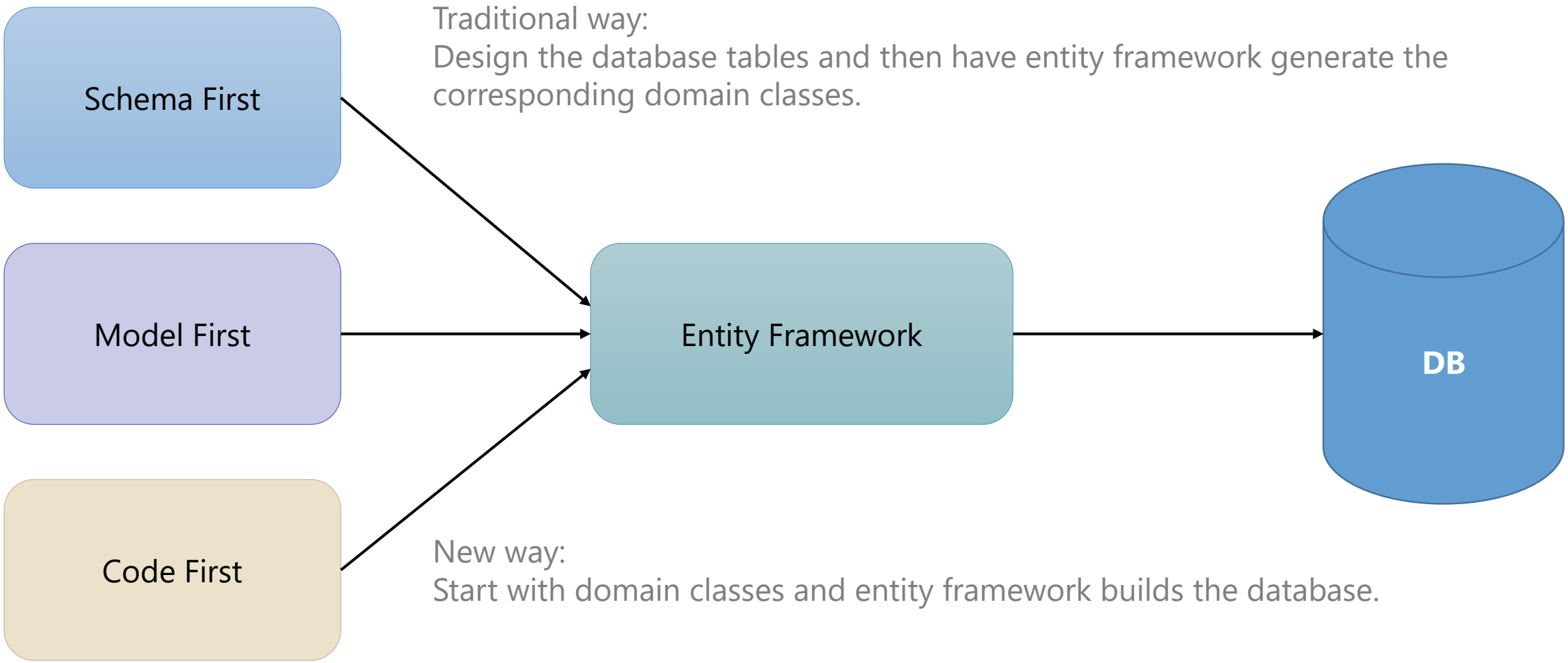
Entity Framework Basics



Entity Framework Flow



Entity Framework Options



Schema First

Schema First / Model First

1. Add an ADO .NET Data Model into your project
2. Select the database to query from the wizard
3. Select the tables, views and stored procedures
4. Write LINQ to Entities queries that use the ObjectContext

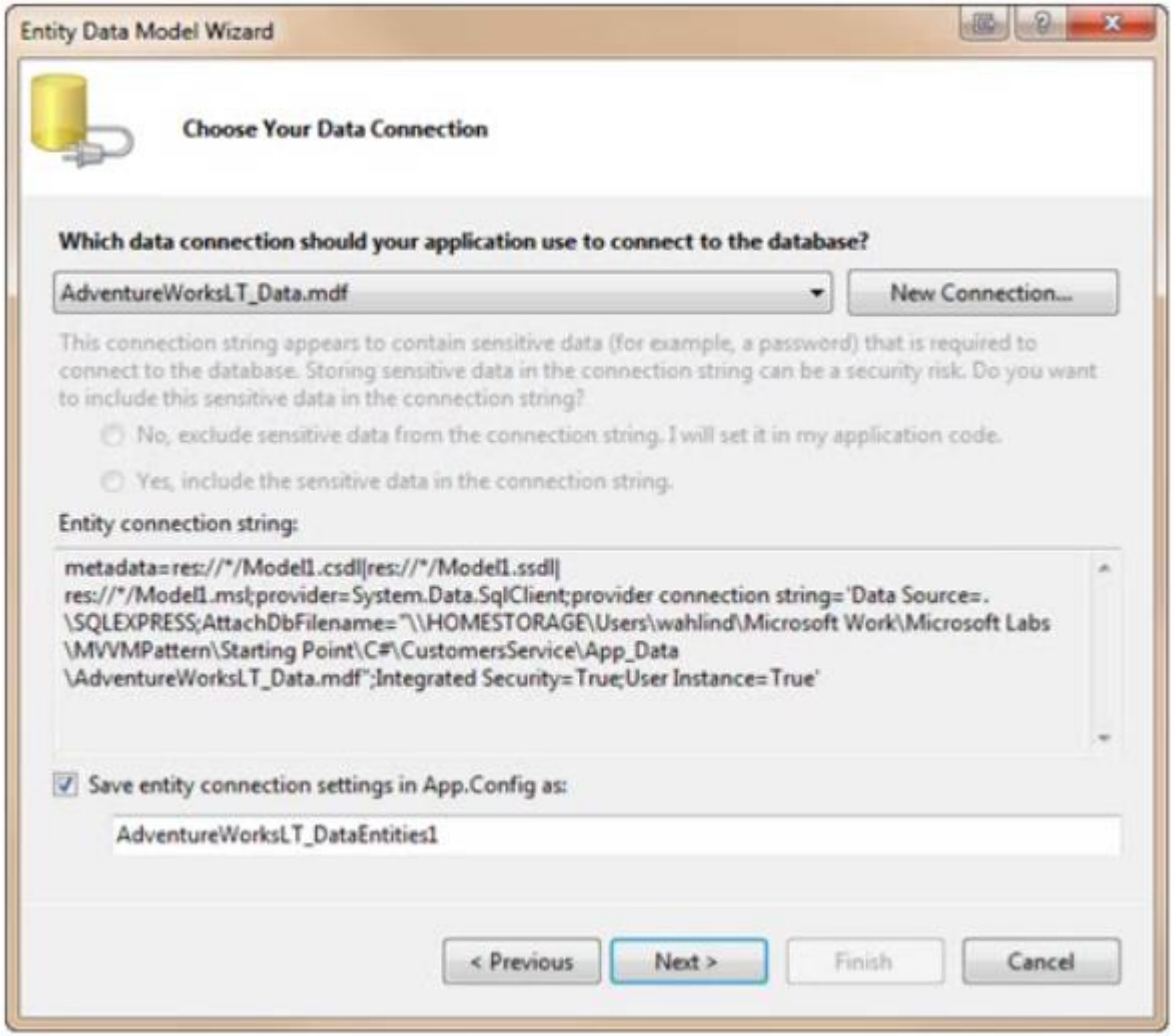
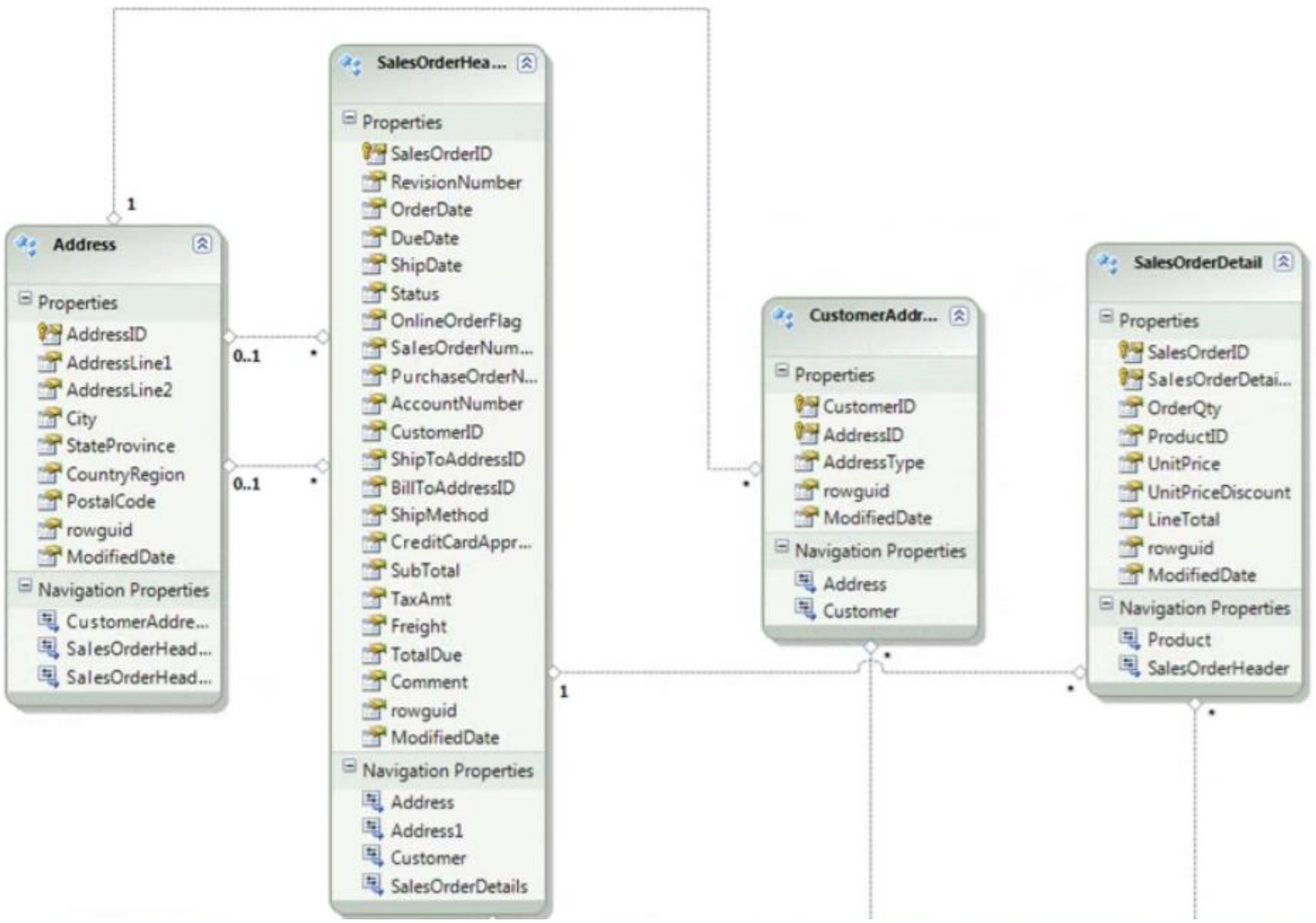
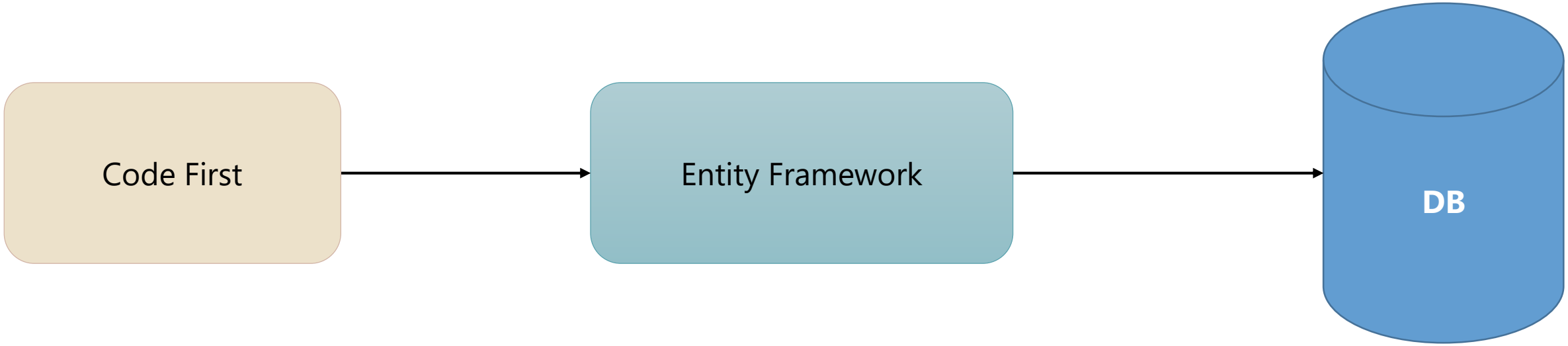


Diagram Outcome



Code First

Code First

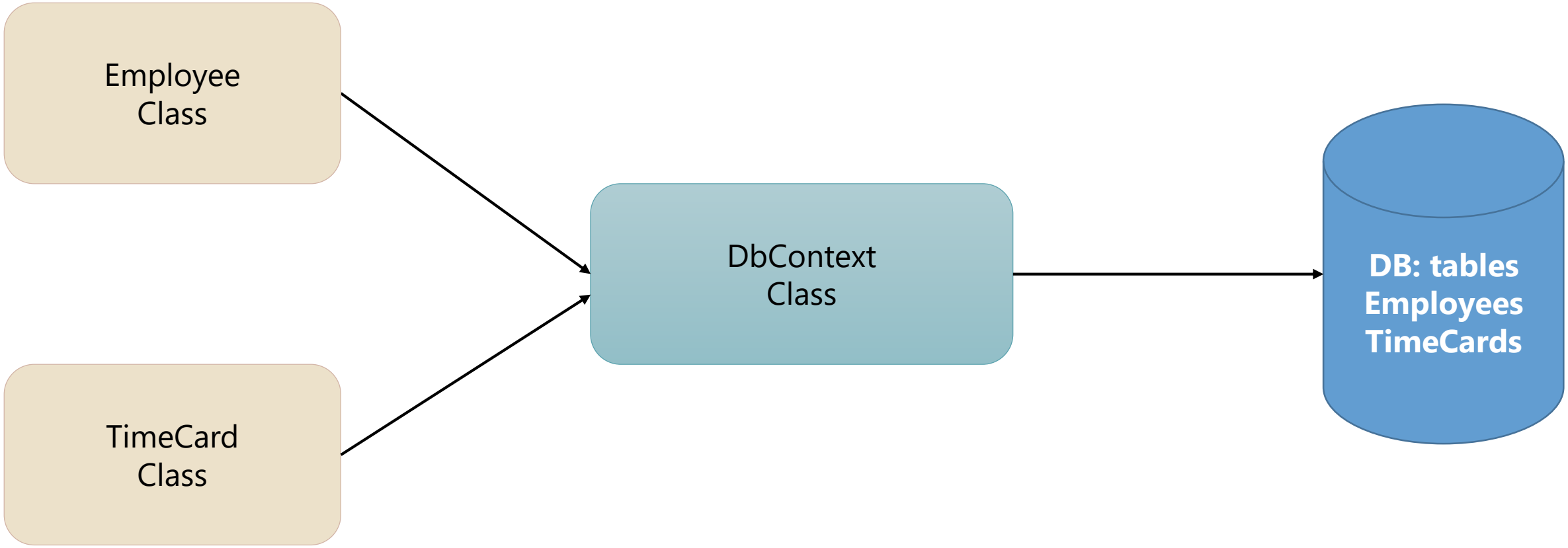


Code First Advantages

- Increase productivity.

- Database designing is a time consuming task.
  - Mapping all relation
  - Changes need scripts
  - Writing code is much faster.
- Full versioning of database
  - Using migration

Code First Example



Code First Migrations

- A migration technique is used to create and update the database every time a domain model changes.
  - By adding a class or by modifying an existing ones.
- Migrations provide a history of all changes done to the code (database) and allow us to revert back to a specific change made.
- Provide us with a Seeding method.
  - Actions to be taken upon database creation/change.

Steps for Code First Approach

- Install EntityFrame from Nuget Manager
- Design & write Classes
- Add Database Context Class
- Enable Migrations → Add first Migration
- Query Class (repository)

Demo: TimeTracker Display Employees (Web Forms)

- Write a web application that will use code first function to track time cards for employees in a company.
- For each employee we need to track: ID, First Name, Last Name, Department and all working time cards.
- For each time card we need to know the card: ID, submission data and the hours worked for each day of the week.

Step 1: Employee and TimeCard Classes

```
public class Employee
{
    public int ID { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Department { get; set; }
    public List<TimeCard> timeCards { get; set; }

}
```

```
public class TimeCard
{
    public int ID { get; set; }
    public DateTime submissionDate { get; set; }
    public int MondayHours { get; set; }
    public int TuesdayHours { get; set; }
    public int WednesdayHours { get; set; }
    public int ThurdaydayHours { get; set; }
    public int FridayHours { get; set; }
    public int SaturdayHours { get; set; }
    public int SundayHours { get; set; }
}
```

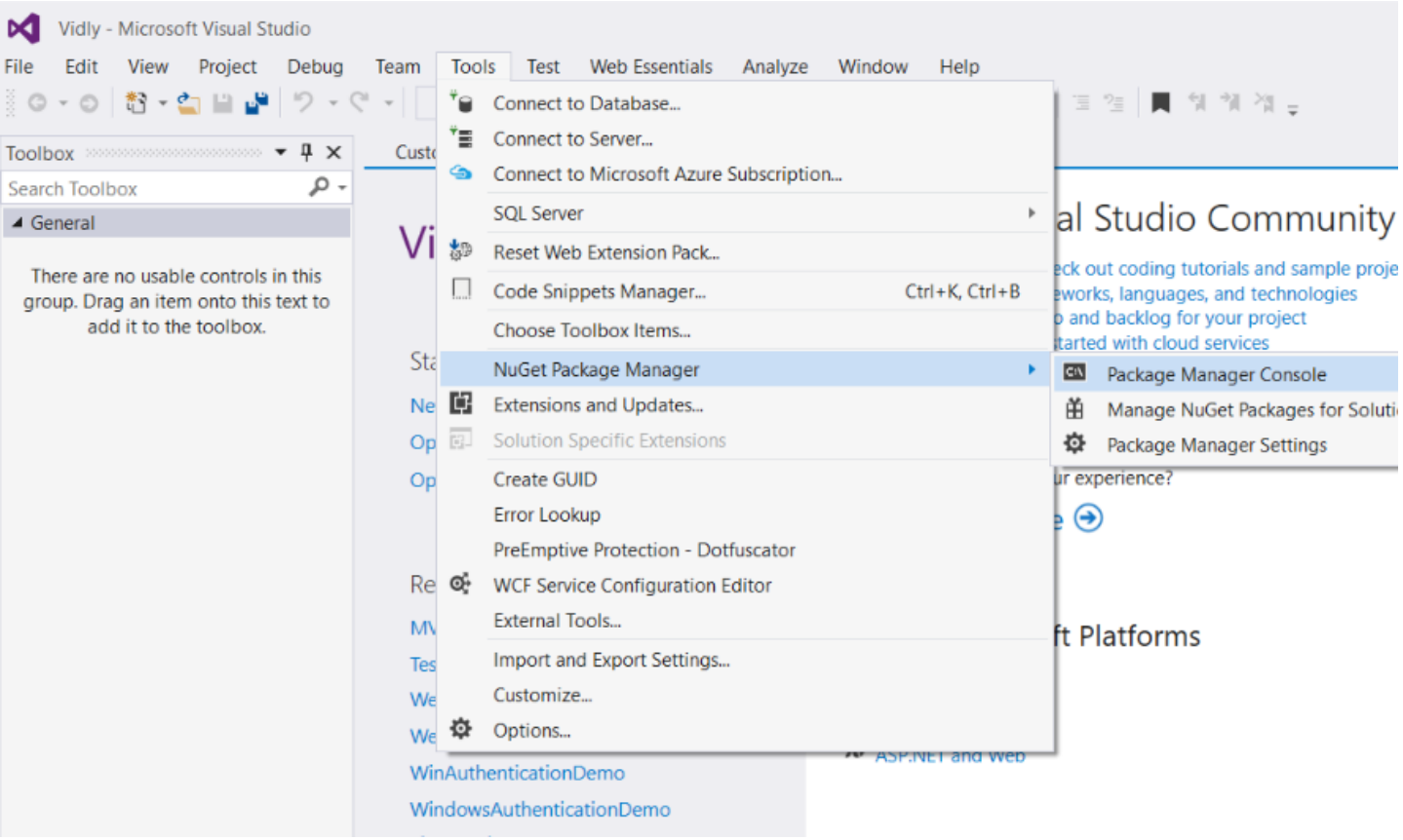
Step 2: DbContext Class

```
public class TimeTrackerDbContext : DbContext
{
    public DbSet<Employee> Employees { get; set; }
    public DbSet<TimeCard> TimeCards { get; set; }
}
```

Note: do not forget to add the needed name spaces to your created classes.

Step 3: Code First Migration

- Go to Package Manager Console



Step 3: Code First Migration

- Enable migration: First time only
  - Type: *enable-migrations*.....**(3.1)**
- Check solution explorer for "Migration" folder
  - Stores all migration history
  - Contains a configuration file to ***Seed*** the database.....**(3.2)**
- Creating first migration
  - Console: *add-migration UniqueName*.....**(3.3)**
  - Use a useful name, for example: *InitialMigration*
  - Examine Migration folder content again.
  - Consider a migration as a restore point that you can come back to, so choose the name wisely.
- Commit the Migration
  - *Update-database*.....**(3.4)**

Step 4: Running Engine – The Repository Class

In this class you write all methods that will retrieve data from you from the DB.

```
public class TimeTrackeRepository
{
    TimeTrackerDbContext _context = new TimeTrackerDbContext();

    /*
     * This methods gets the records of all employees
     * */
    public List<Employee> getAllEmployees()
    {
        List<Employee> allEmps =
            (from data in _context.Employees
             select data).ToList();

        return allEmps;
    }

    public List<TimeCard> getemployeeTimeCard(int empID)
    {
        List<TimeCard> mytimeCards =
            (from data in _context.Employees
```

Note: do not forget to add the needed name spaces to your created classes.

Step 5: Design your Web Application

localhost:60598/Default.aspx

Welcome to my TimeTracker Application

	ID	First Name	Last Name	Department
Select	1	Barry	Allen	IT
Select	2	Tom	Alex	HR
Select	3	Tim	Horton	Sales
Select	4	Alice	Wonder	IT
Select	5	Aref	Mour	IT
Select	6	Thomas	Adison	Engineering

Use ObjectDataSource Control to query your created database

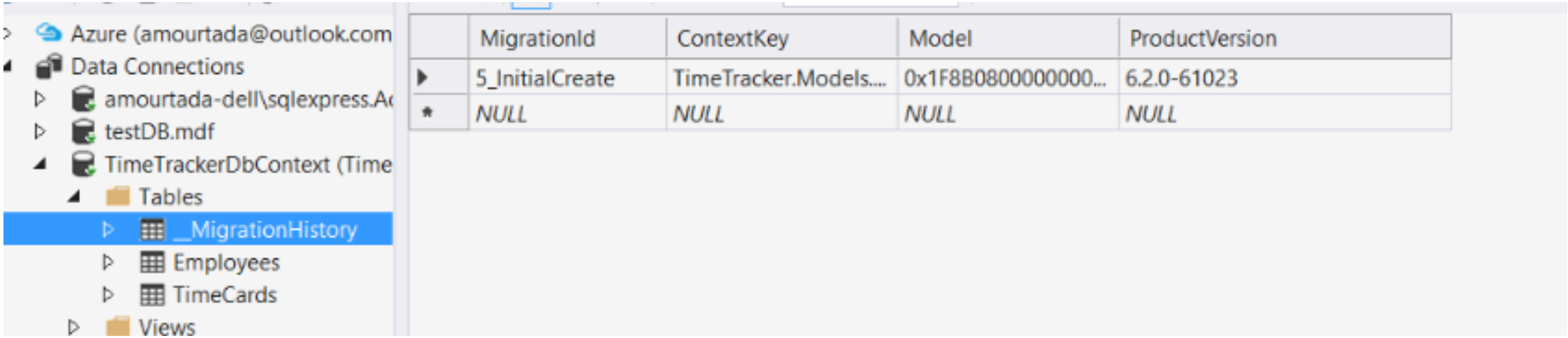
Exercise: TimeTracker Display Time Cards

- Implement the following using the web forms created in the Demo.

- Enable selection in the Employees GridView.
- Upon selection get all time cards for that employee from the database and display results in a GridView

Tracking Changes in Class Design

- _MigrationHistory table is used to track any change in the entity framework model.



Changes in Class

- Need to change to classes
- Classes are connected to the database table

- How can we handle changes?
  - Make changes to classes.
  - Add-Migration *NewUsefullName*
  - Update-database
- Do not forget to update ***Seed*** method.

To Make Changed in Domain Classes

1. Update C# Classes
2. Update Seed Method (if needed)
3. Add-migration *ProperName*
4. Update-database

Do not make new changes to C# classes before update-database

Demo: Handling Changes

- Add a new property to Employee.
  - Role (string)
  - Date of Hire (date)
- Update Seed method.
- Add a migration.

- Try it on you own that you feel is important.

Overriding Conventions (1)

- There several conventions that are built into entity framework that are used to determine the schema of the database.
- Example:
  - “FirstName” property in the Employee class in of String Value
  - In C#, string can have a null value with no limit of number of characters.
  - Entity frame work will define the column in the table as it can be null with max size. (type nvarchar(max))

Overriding Conventions (2)

- To override entity framework default convention, use ***"Data Annotation"***
- Add namespace:
  `System.ComponentModel.DataAnnotations;`
- Above each property, apply a data "Annotation"
- Annotations are placed within square brackets.
- A single property can have more than one annotation.
  - Examples
    - [Required] : cannot be null
    - [StringLength(100)]

Demo\Exercise: Data Annotation

- Add needed data annotations to the Employee class properties.
- Create a new migration named "ApplyAnnotationsToEmployeeClass"
- Update the database

- Try to add annotations to other variables.

Q & A

JOHN ABBOTT
CEGEP/COLLEGE