MONASH University

FIT5032 - Internet Applications Development

Introduction to Entity Framework

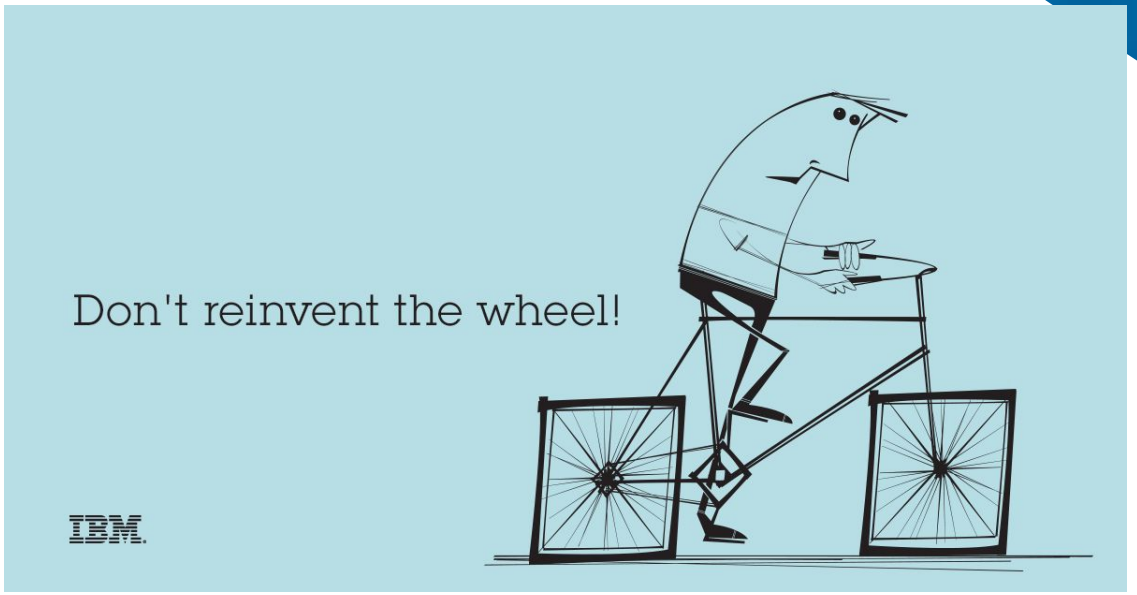
Prepared by - Jian Liew

Last Updated - 30th May 2018

Monash University (Caulfield Campus)

Amount of programming needed

- The amount of programming needed for this subject is actually **considerably smaller in comparison to other subjects.**
- In this subject, we will be using tools which are ready made for us to simplify the development process, but it is still important for us to understand how to properly utilise the tools.
- In other words, we will try “not reinvent the wheel”.
- However, at the end of the day everything depends on the context. This subject is designed so that we try to avoid “reinventing the wheel”. In real life, however often times there are times when reinventing the wheel is needed.
- That being said, it is important to understand the basics of what we will be using.



Outline

- Introduction to Entity Framework
- What is Object Relational Mapping?
- Benefits and Drawbacks of ORM
- How does Entity Framework work?
- Entity Framework Architecture
- Four Different Development Workflows for EF
- Model First with EF
- Code First
- Inheritance Strategy
- Advantages and Disadvantages of Code First

Introduction to Entity Framework (EF)

- The current version is 6.2 as of 18th May 2018
- The Entity Framework is a set of technologies **that support the development of data-oriented software applications.**
- The Entity Framework enables developers to work with data in the form of domain-specific objects and properties, such as customers and customer addresses, **without having to concern themselves with the underlying database tables and columns where this data is stored.**
- Entity Framework is an **object-relational mapper (ORM)** that **reduces the impedance mismatch between the object-oriented world of .NET Framework developers and the world of relational databases**
- In the Java world, this is accomplished by **JPA or Hibernate** just to name a few.
- There is also NHibernate for .NET applications however it will not be our focus in this subject.

What is an Object Relational Mapper (ORM)?

- Object-Relational Mapping (ORM) is a technique that lets you **query and manipulate data from a database using an object-oriented paradigm.**
- An ORM library is a completely ordinary library written in your language of choice that encapsulates the code needed to manipulate the data, so you don't use SQL anymore; you interact directly with an object in the same language you're using.
- We will be using Entity Framework 6 as our ORM of choice.
- **There are pro and cons of using an ORM.**

Benefits of using an ORM

Using ORM saves a lot of time because:

- **DRY (Don't Repeat Yourself)**: You write your data model in only one place, and it's easier to update, maintain, and reuse the code.
- You don't have to **write poorly-formed SQL.** So, you do not need to know how to write SQL.
- Sanitizing; using prepared statements or transactions are as easy as calling a method. (Input sanitization)

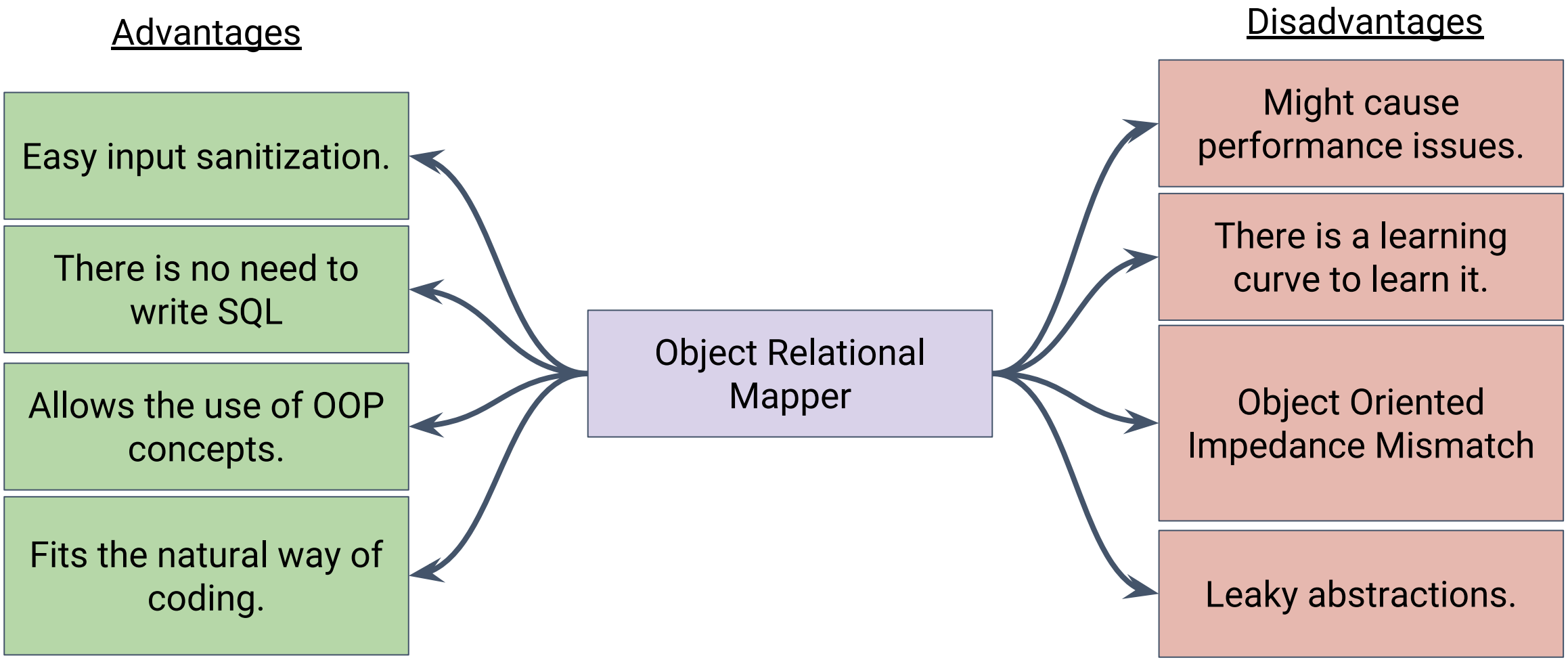
Using an ORM library is more flexible because:

- **It fits in your natural way of coding** (You can write queries in the language of your development, in this case C#)
- It abstracts the DB system, so you can change it whenever you want.
- The model is weakly bound to the rest of the application, so you can change it or use it anywhere else.
- **It allows you to use Object Oriented Programming concepts like inheritance.**

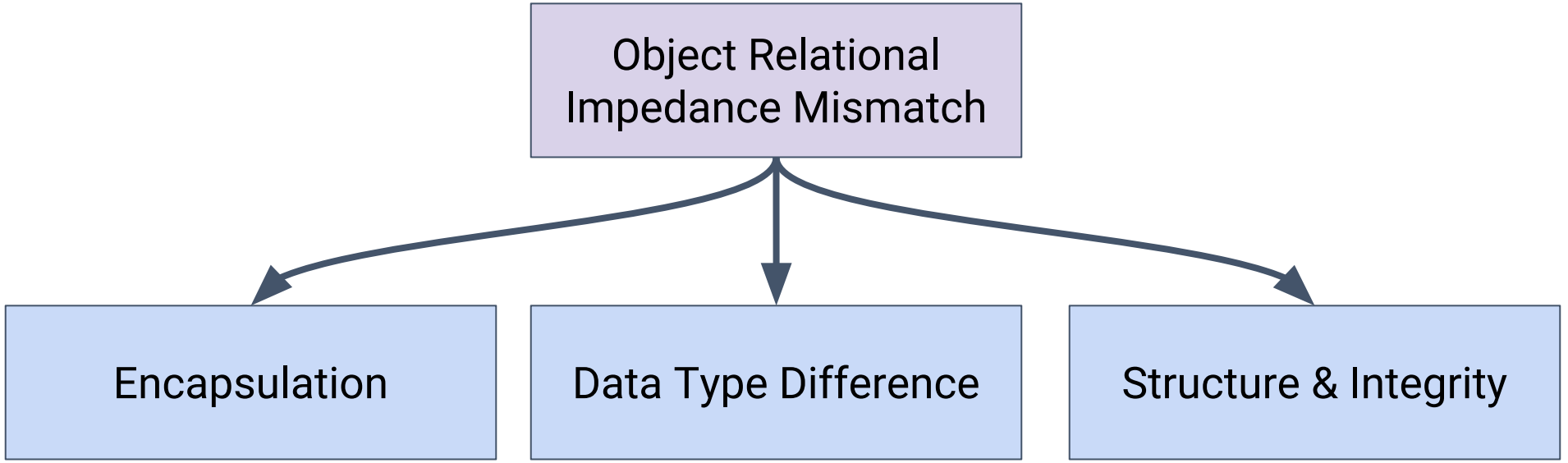
Drawbacks of using an ORM

- There is a **learning curve** on how to use an ORM framework. (How to use it “properly”)
- Performance for usual queries is normal, but SQL will always do better for certain projects. **(This is debatable, as LINQ allows custom queries).** For example, there are times when a very specific SQL query is needed. **(An example is a *top-n* query)**
- Martin Fowler who is a famous software developer and author considers ORM to be **leaky abstractions.**
- **Leaky abstractions** means that your abstraction exposes some of the implementation details, or that you need to be aware of the implementation details when using the abstraction.
- At the end of the day, the **ORM will only “reduce” the object-relational impedance mismatch.**

ORM Advantages & Disadvantages



Object Relational Impedance Mismatch

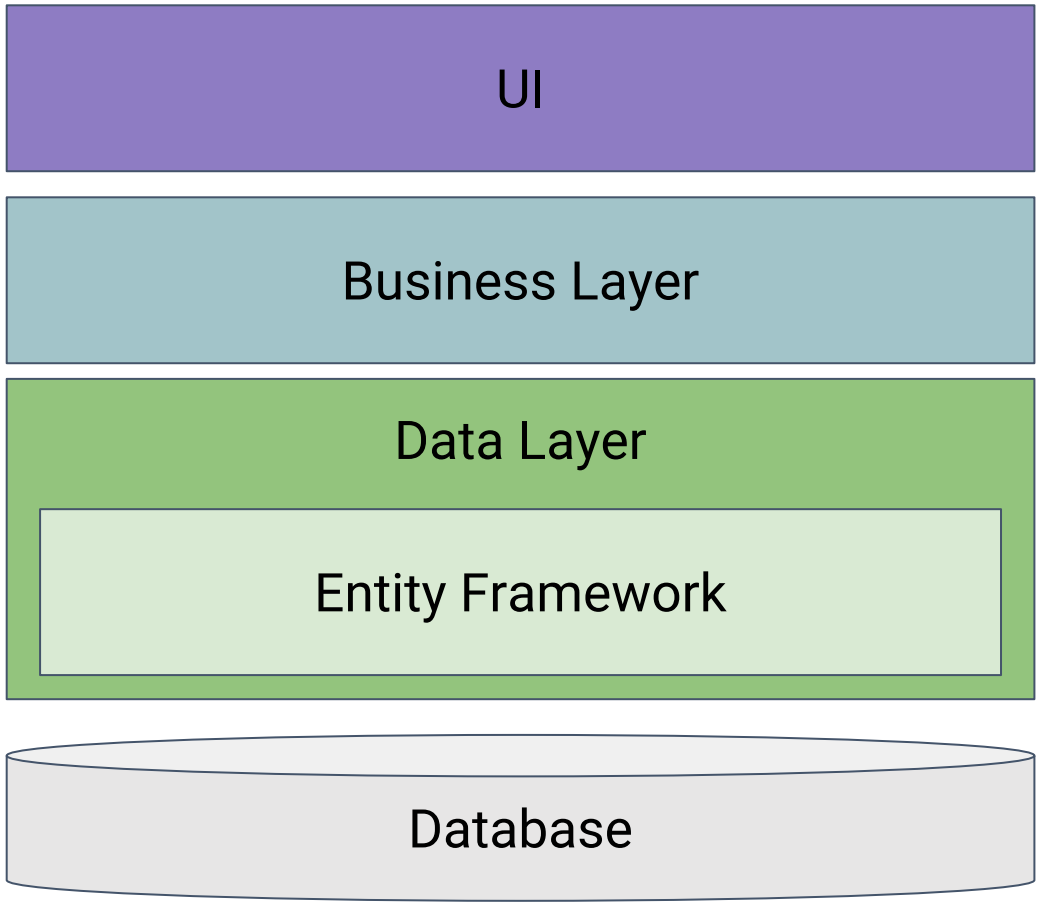


Because what we are doing is going from POCO (Plain Old C# Object) to database tables, we have these issues. The **EF frameworks aims to reduce** these issues that arises. These are the 3 major ones.

Object-relational Impedance Mismatch

- The set of difficulties that arises when you try to combine an object-oriented code with a relational database.
- **Encapsulation** – in OOP every class has internal and private implementation that is contained and maintained by the class instance. (RDMS uses “public” data)
- **Data type differences** – no pointer/reference data types are allowed in the relational systems. Even more, there are substantial differences between the scalar data types in both OOP and RDBMS models, which cause translation/mapping difficulties.
- **Structure and integrity** – in OOP it is considered normal to have highly nested structures (objects containing sets of different objects recursively, thus producing quite complicated object graphs). This poses difficulties in translating these structures to relational schemas, where all data is represented in a named set of global, flat relation variables.
- The Entity Framework aims to **reduces the impedance mismatch between the object-oriented world of .NET Framework developers and the world of relational databases. (Reducing it does not mean eliminating it)**

How Entity Framework fits..



- Entity Framework fits between the business entities (domain classes) and the database.
- It saves data stored in the properties of business entities and also retrieves data from the database and converts it to business entities objects automatically.

Entity Framework Features

- Cross-platform: EF Core is a cross-platform framework which can run on Windows, Linux and Mac. **(We are not using EF Core)**

- Modelling: EF (Entity Framework) creates an EDM (Entity Data Model) based on POCO (Plain Old CLR Object) entities with get/set properties of different data types. It uses this model when querying or saving entity data to the underlying database.

- Querying: EF allows us to use LINQ queries (C#/VB.NET) to retrieve data from the underlying database. The database provider will translate this LINQ queries to the database-specific query language (e.g. SQL for a relational database). EF also allows us to execute raw SQL queries directly to the database.

- Change Tracking: EF keeps track of changes occurred to instances of your entities (Property values) which need to be submitted to the database.

EF6 vs EF Core

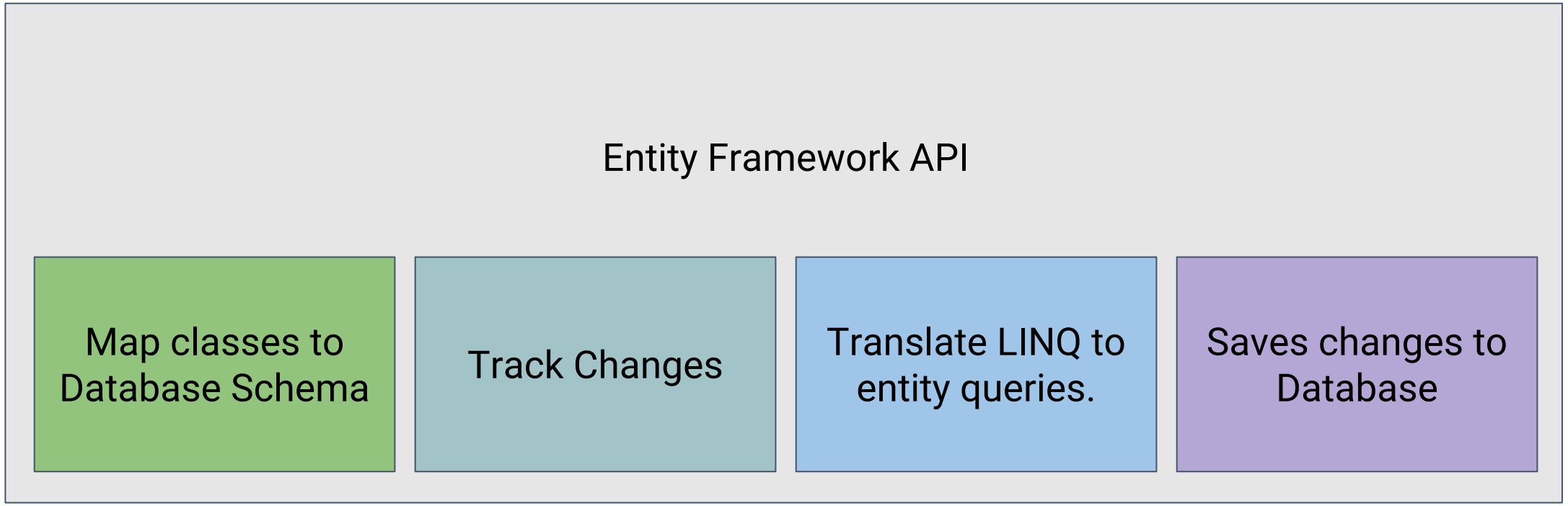
- There are two different versions of Entity Framework, Entity Framework Core and Entity Framework 6.
- Entity Framework 6 (EF6) is a tried and tested data access technology with many years of features and stabilization.
- Starting with the EF4.1 release it has shipped as the EntityFramework NuGet package - currently one of the most popular packages on NuGet.org.
- Entity Framework Core (EF Core) is a lightweight, extensible, and cross-platform version of Entity Framework.
- EF Core introduces many improvements and new features when compared with EF6
- **For this subject, we will be using EF6 instead of EF Core.**

EF Core and EF6 Feature by Feature Comparison

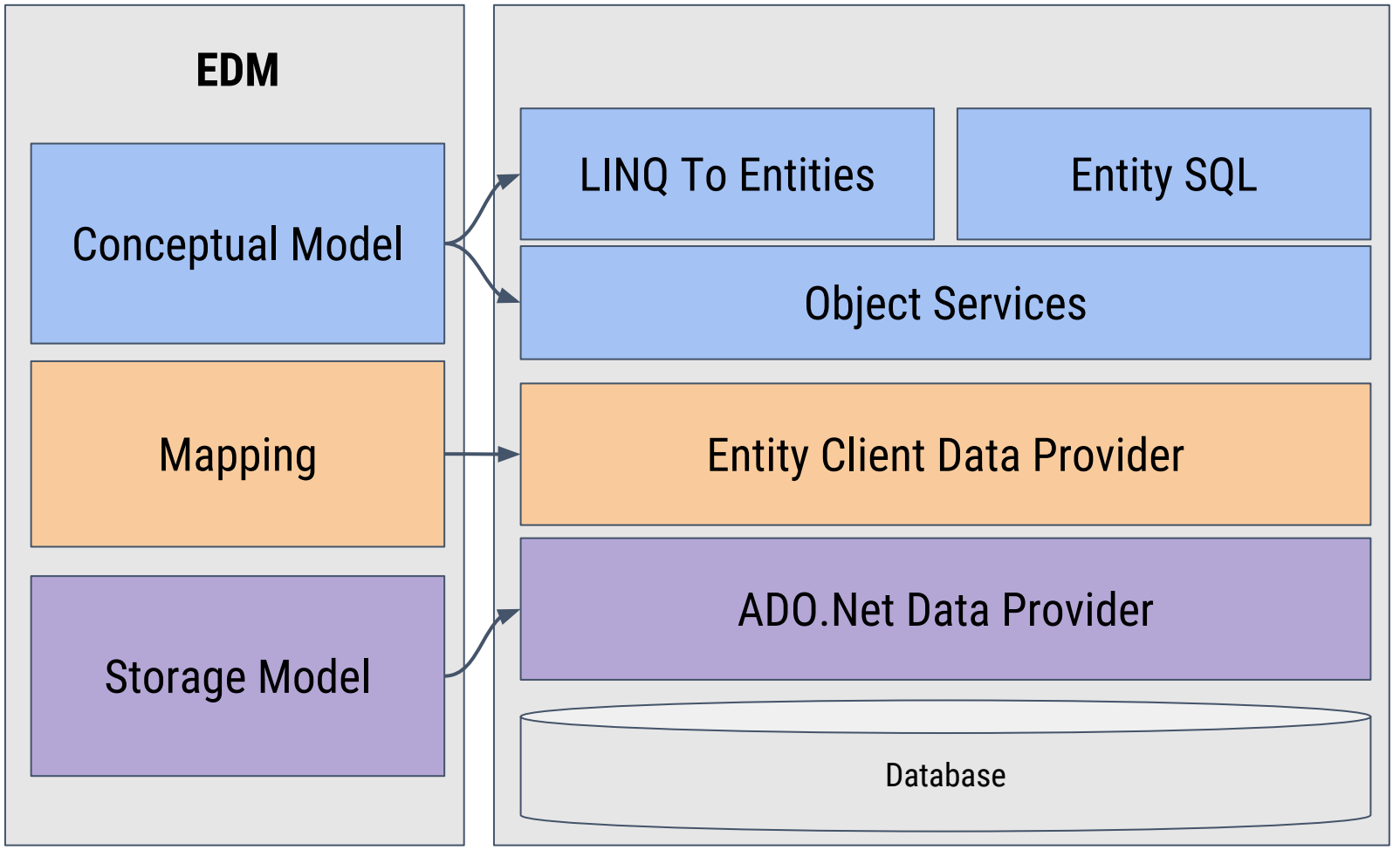
	EF 6	EF Core
Release Date	First released in 2008 however there are continuous updates.	• 2016, (2.1 is the latest version updated recently)
Operating System	• Windows	• Windows, Linux and OSX
Stability	• Mature	• New and Evolving
Requires	• .NET Framework 3.5	• .NET Framework or .NET CORE

How does the Entity Framework work?

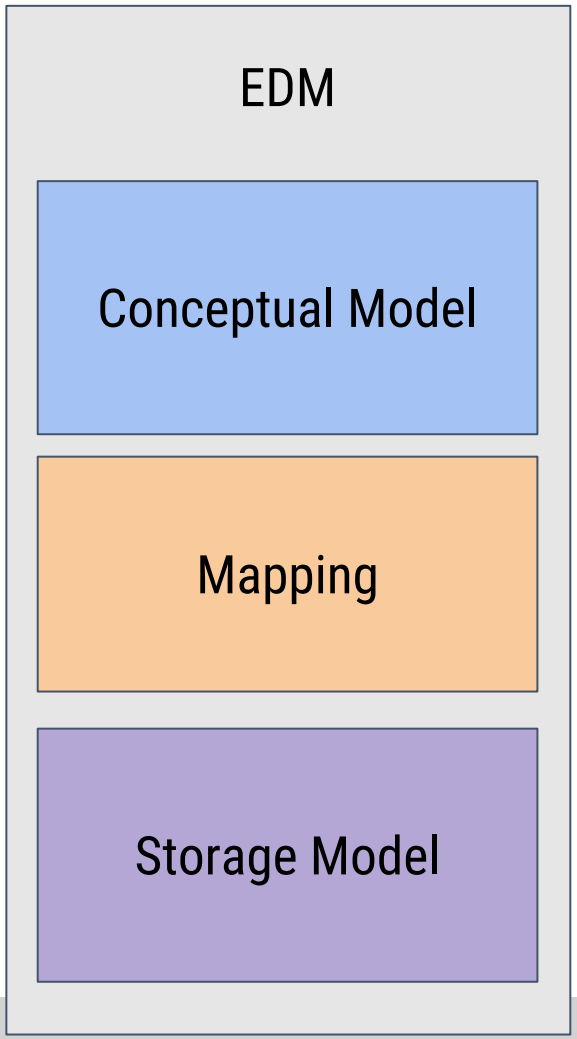
- Entity Framework API (EF6 & EF Core) includes the ability to **map domain (entity) classes to the database schema,** translate & execute LINQ queries to SQL, track changes occurred on entities during their lifetime, and save changes to the database.



Entity Framework Architecture

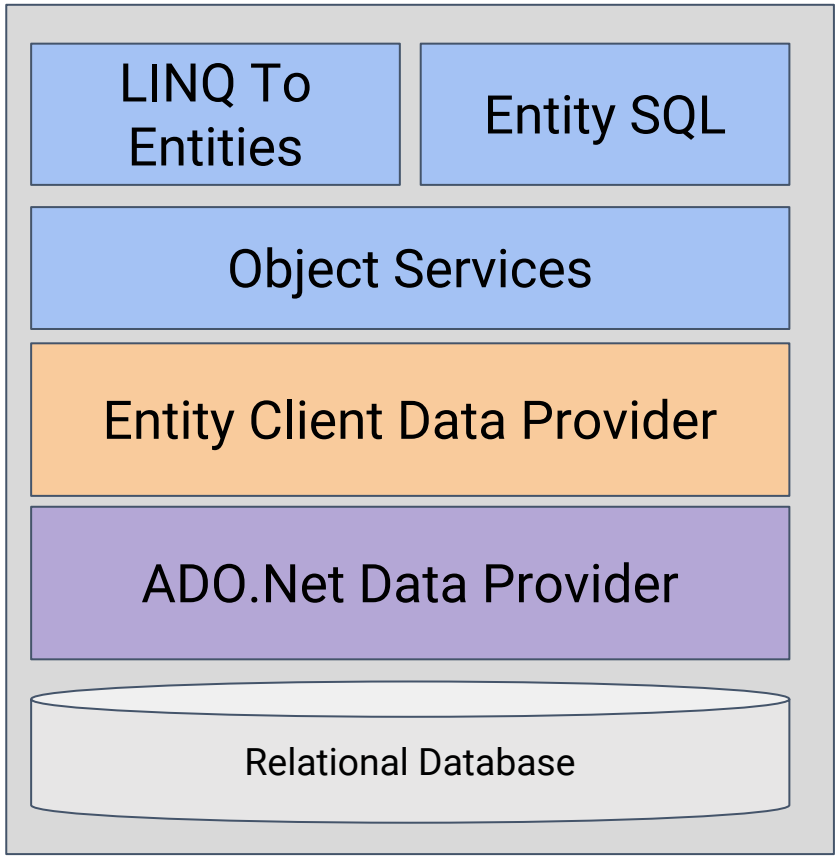


Entity Data Model explained



- The Entity Data Model (EDM) is a **set of concepts that describe the structure of data, regardless of its stored form.**
- EDM is an in-memory representation of the entire metadata: **conceptual model, storage model, and mapping** between them.
- Conceptual Model: The **conceptual model contains the model classes and their relationships**. This will be independent from your database table design.
- Mapping: Mapping consists of information about how the conceptual model is mapped to the storage model.
- Storage Model: The storage model is the database design model which includes tables, views, stored procedures, and their relationships and keys.
- The Entity Data Model Tools in Visual Studio allow you to create an .edmx file from a database or a graphical model and then update that file when either the database or model changes.

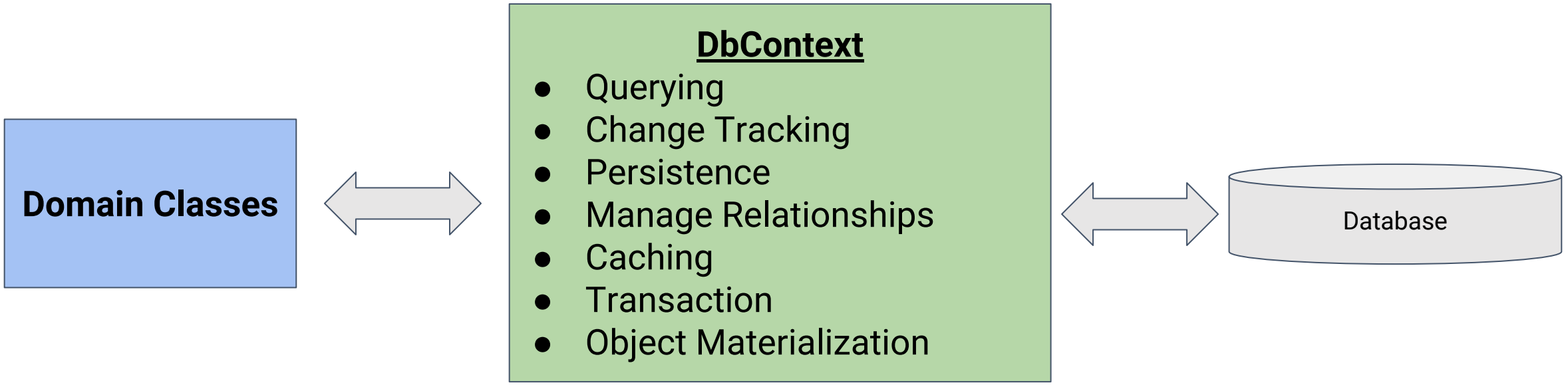
Individual Parts Explained...



- LINQ to Entities: LINQ-to-Entities (L2E) is a **query language used to write queries against the object model.** It returns entities, which are defined in the conceptual model.
- Entity SQL: Entity SQL is another query language (For EF 6 only) just like LINQ to Entities.
- Object service is a main entry point for accessing data from the database and returning it back.
- Entity Client Data Provider: **The main responsibility of this layer is to convert LINQ-to-Entities or Entity SQL queries into a SQL query which is understood by the underlying database**. It communicates with the ADO.Net data provider which in turn sends or retrieves data from the database.
- ADO.Net Data Provider: This layer communicates with the database using standard ADO.Net.

Context Class in Entity Framework

- The context class in Entity Framework is a class which derives from DbContext in EF 6 and EF Core both.
- It is an important class in Entity Framework, which represents a session with the underlying database.



What is an Entity in the EF?

- An entity in Entity Framework is a **class in the domain of your application** which is included as a DbSet<TEntity> type property in the derived context class.
- The Entity Framework API **maps each entity to a table and each property of an entity to a column in the database.**
- An Entity can include two types of properties: **Scalar Properties** and **Navigation Properties**

Scalar Property

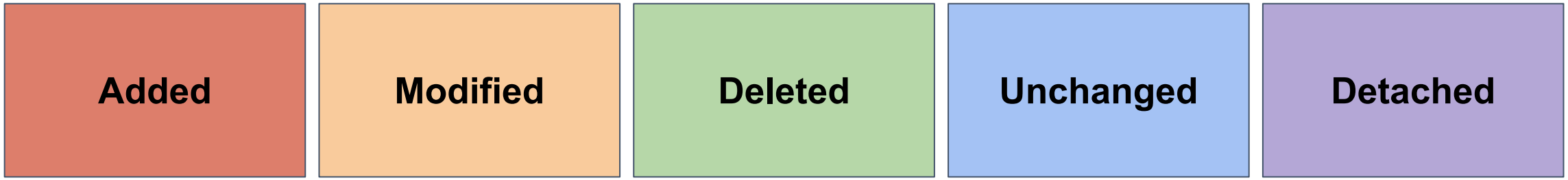
- The primitive type properties are called scalar properties. A scalar property stores the actual data. a scalar property maps to a single column in the database table.

Navigation Property

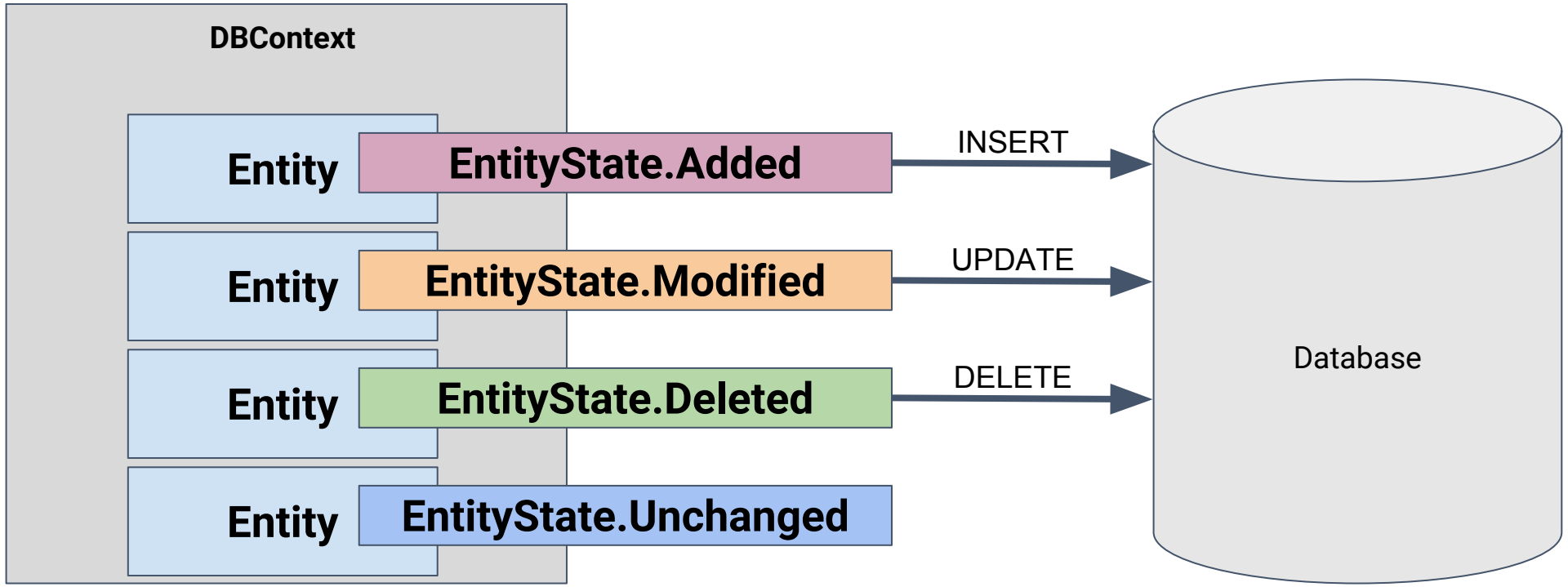
- The navigation property represents a relationship to another entity.
- There are two types of navigation property. **Reference Navigation and Collection Navigation**

Entity States

- EF API **maintains the state of each entity during an its lifetime.**
- Each entity has a state based on the operation performed on it via the context class.
- The entity state represented by an enum **System.Data.Entity.EntityState in EF 6** and **Microsoft.EntityFrameworkCore.EntityState in EF Core** with the following values:



Entity States continued...



*The DbContext does not keep track of the detached state of an entity.

Development Workflow

Before we start, we must ask certain questions. These questions are divided into two categories. Things outside your control and things inside your control.

Things outside your control

- Are you targeting a new database? (A database that does not exist)
- Or does that database exist that has been created and populated with tables?

Things inside your control

- Do you want to write code? (Code First Approach)
- Or do you want to use a designer? (Conceptual Model)

The Four Different Workflow

	Model First	Code First
New Database	**Model First** ● Create Model in Designer ● Database created from Model ● Classes auto-generated from Model	**Code First (New Database)** ● Define classes and mapping in code ● Database created from model ● Use Migrations to evolve Database
Existing Database	**Database First** ● Reverse engineer model in Designer ● Classes auto-generated from Model	**Code First (Existing Database)** ● Define classes and mapping in code ● Reverse Engineer tools available.

Number of Approaches?

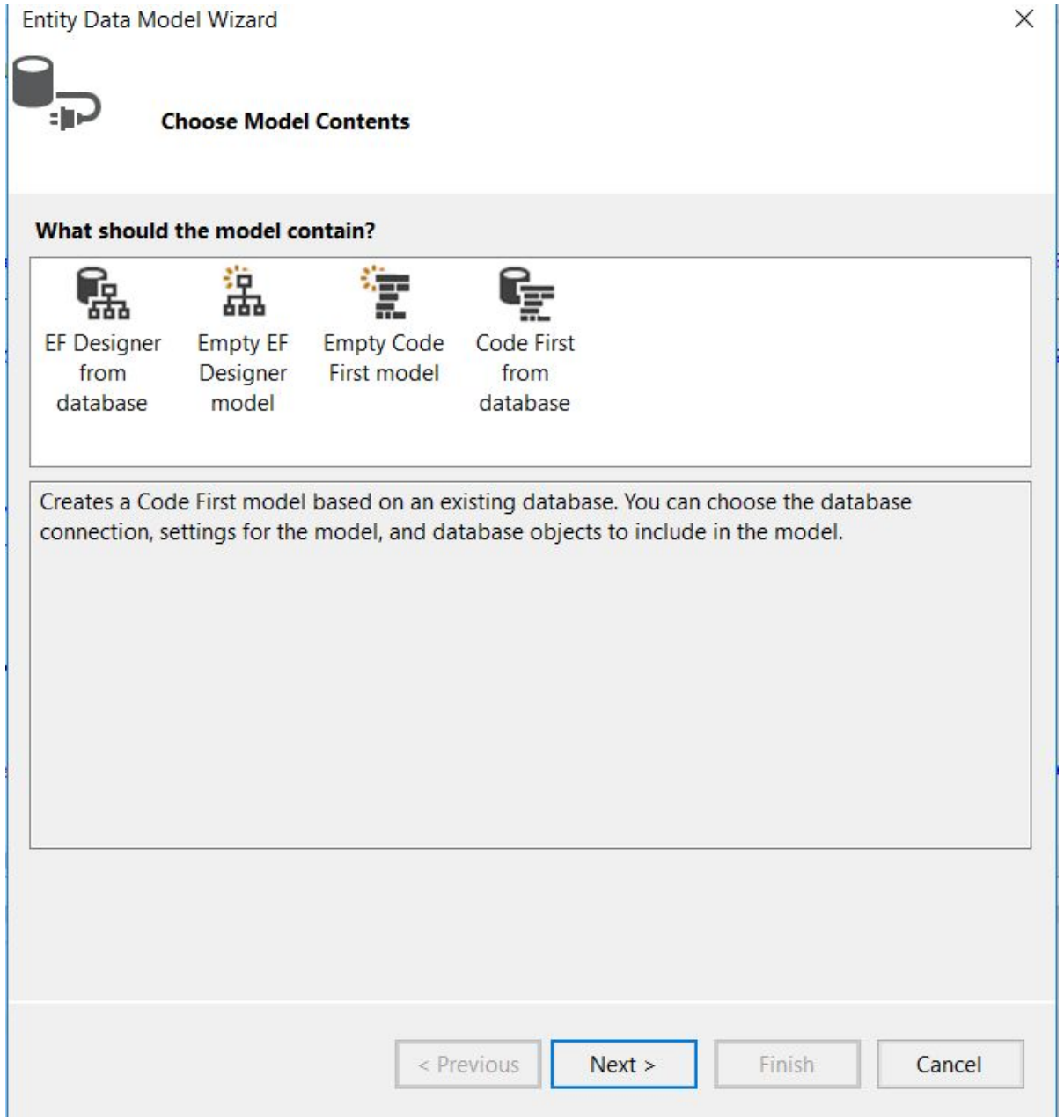
So, there are 2 major categories of workflow (Code First and Model First).

Both of these workflows can be done with or without a database present.

This is the main reasoning why the community **often state that there are 3 major ways**

- **Database First Approach** (This is where a database is created first using Data Definition Language. So, you are writing SQL first)
- **Model First Approach** (This is where a **Visual Designer** is used or the model is mapped out via XML configuration files)
- **Code First Approach** (This is when Code (C# classes) are written or created)

One of the more popular ways is to **first create the database**, and then automatically generated the model from the database. This will be explained both in the lectures and the lab exercises. Of course, you should know how to do this as you have completed the database unit. :D



Microsoft Entity Framework

The four different ways.

1. EF Designer From Database.
   (Model First from existing Database)

2. Empty EF Designer Model
   (Model First with no existing Database)

3. Empty Code First Model
   (Code First with no existing database)

4. Code First from Database
   (Code First from existing Database)

Singular vs Plural Table Names

One of the topics that is often debated amongst developers is the usage of **singular vs plural** in the naming of tables.

It is common to prefer Singular names.

Reasons

1. **Concept**. You can think of a bag containing potatoes. It does not matter if it contains 0,1 or a million potatoes, it is still the same bag. Tables are just containers, the table must describe what it contains, not much data it contains.

2. **Convenience.** It is easier to come out with singular names than plural ones. Object can have irregular plurals or not plural at all but always have a single one. There are some exceptions like **News**.

3. **Simplicity.**

Certain frameworks would require you to have plural table names. EF for example tends to pluralise table names while Oracle prefers Singular table names. But it is more important to be consistent.

What is Model First?

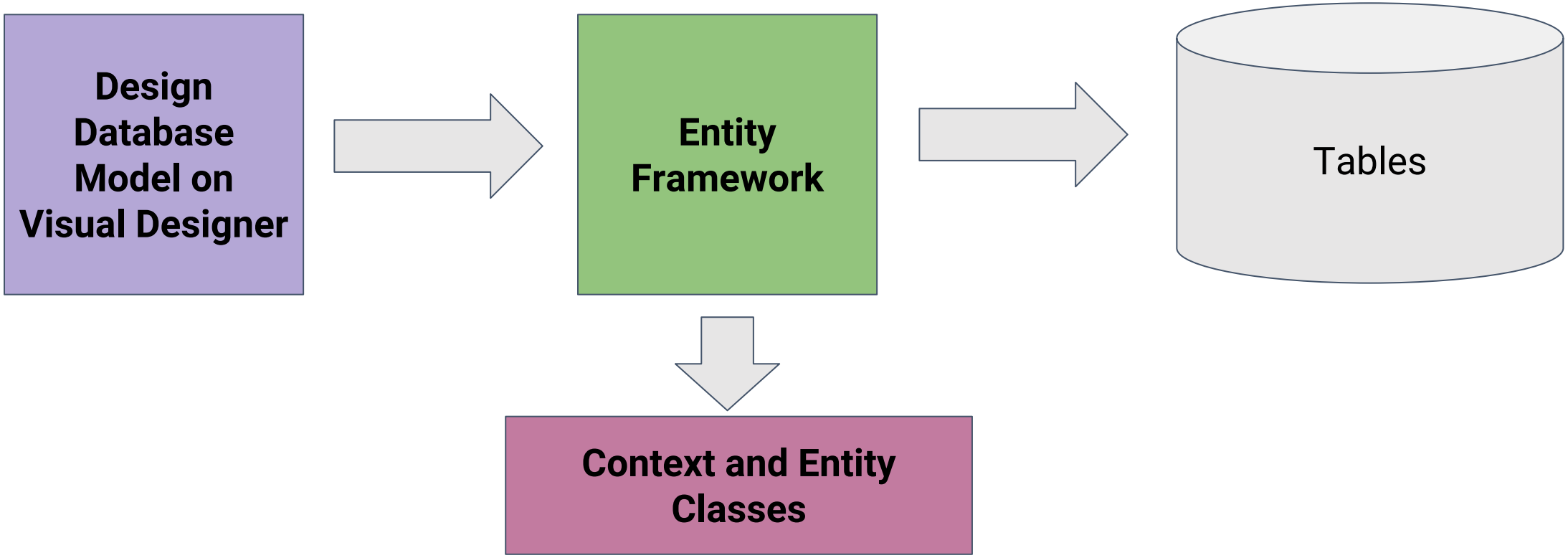
- Entity Framework Tools will be used to generate model from an existing database. This generates a default **conceptual model and mapping**, which you can customize by using the **Entity Data Model Designer**. You can also use tools to graphically create a conceptual model by using the Entity Data Model Designer, and then generate a database based on the metadata built by the tool from that model. (There are two ways here actually)

- A conceptual model is a specific representation of the structure of data as entities and relationships, and is generally defined in a domain-specific language (DSL) that implements the concepts of the EDM.

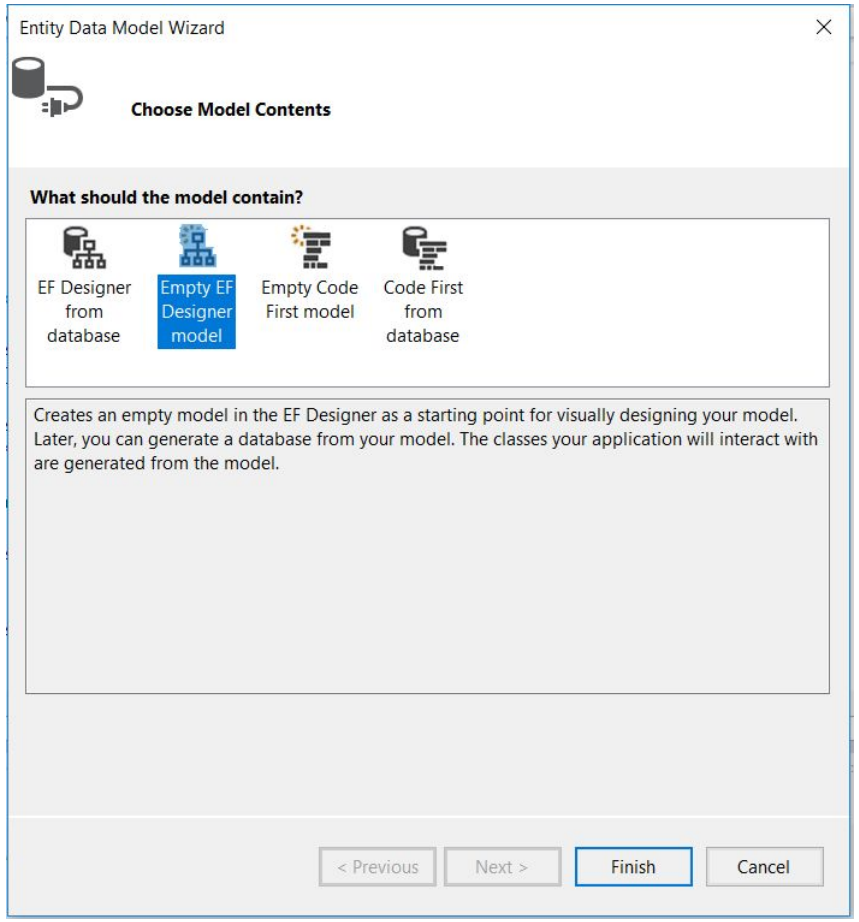
Model First with EF

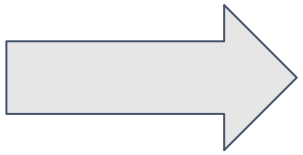
***EF Core** does not support this approach. This is one of the main reasons we are not using it.

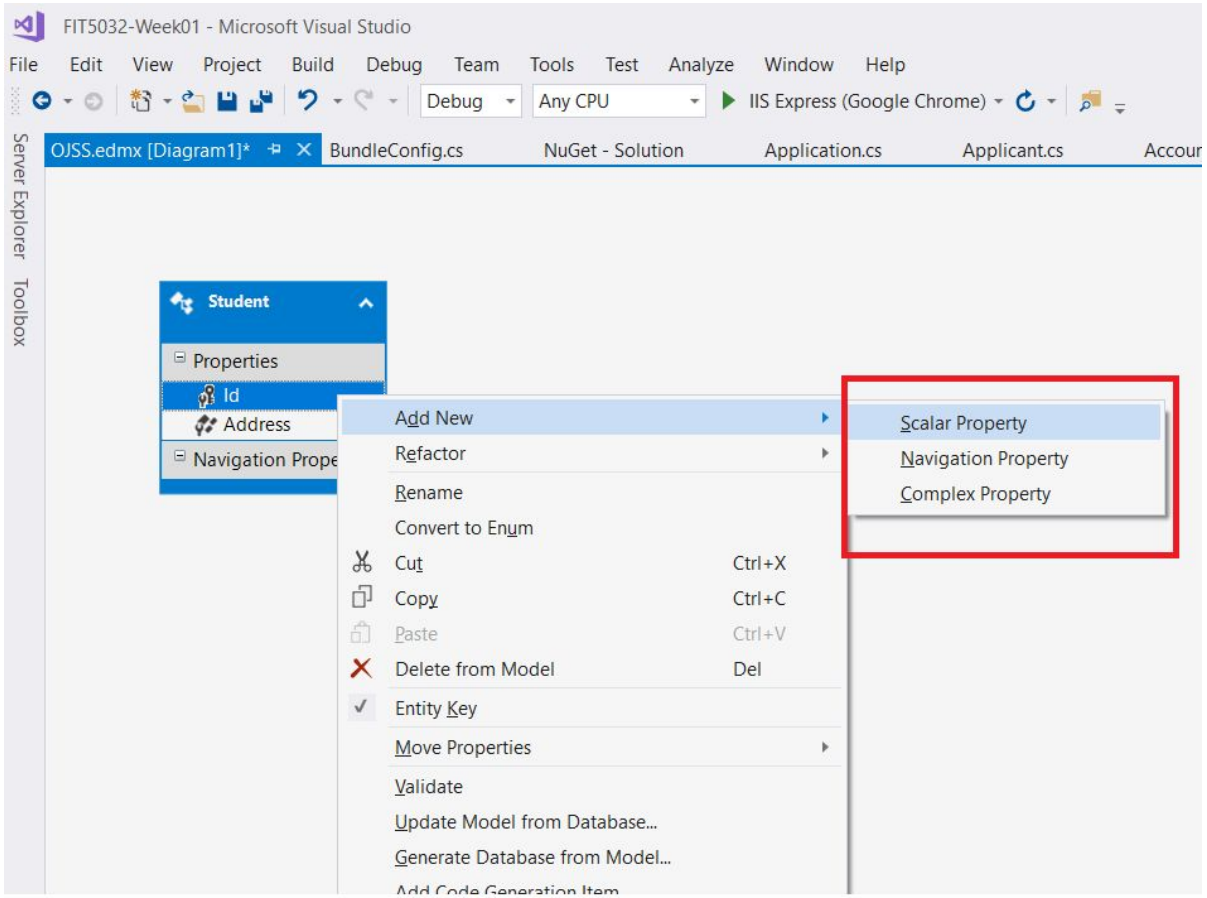
In the Model-First approach, you create the entities, relationships, and inheritance hierarchies directly on the design surface of EDMX and then generate the database from your model.



Visual Studio Model First Approach

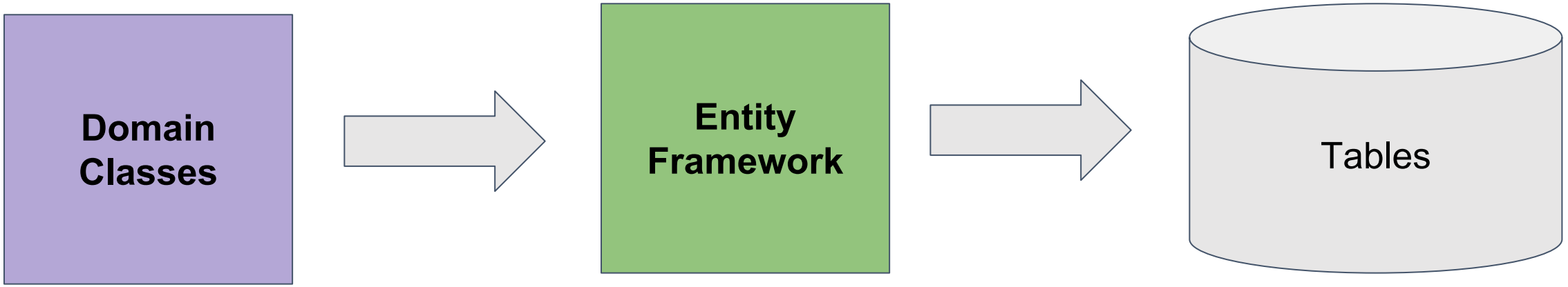






What is Code First?

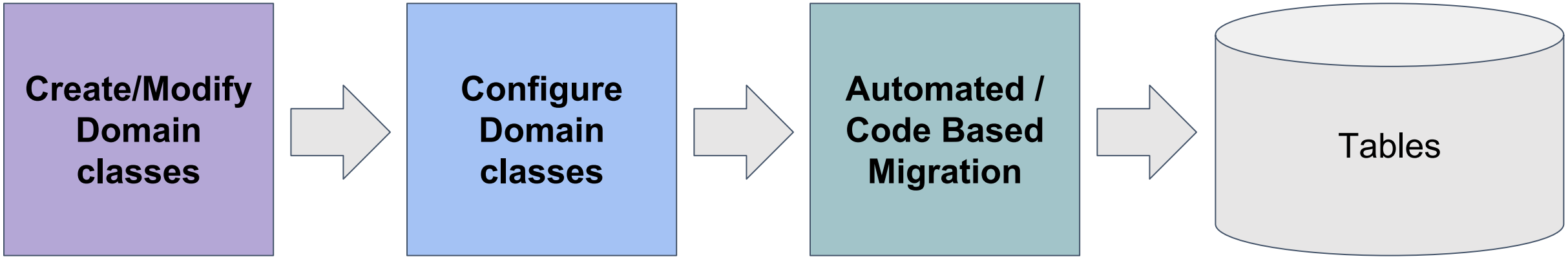
- Entity Framework introduced the Code-First approach with Entity Framework 4.1.
- Code-First is mainly useful in **Domain Driven Design.**
- In the Code-First approach, **you focus on the domain of your application** and start creating classes for your domain entity rather than design your database first and then create the classes which match your database design.
- EF API will create the database based on your domain classes and configuration



The Code First Workflow (More detailed)

The development workflow in the code-first approach would be:

1. Create or modify domain classes
2. Configure these domain classes using Fluent-API or data annotation attributes
3. Create or update the database schema using automated migration or code-based migration.



Configure Domain Classes

- Code-First builds the conceptual model from your domain classes using default conventions
- EF 6 Code-First leverages a programming pattern referred to as convention over configuration.
- There are two ways to configure your domain classes:
  - By using Data Annotation Attributes
  - By using Fluent API

Data Annotations Attributes

- Data Annotations attributes are .NET attributes which can be applied on an entity class or properties to override default conventions in EF 6 and EF Core.
- Data annotation attributes are included in the System.ComponentModel.DataAnnotations and System.ComponentModel.DataAnnotations.Schema namespaces in EF 6 as well as in EF Core.
- Data annotations only give you a subset of configuration options. **Fluent API provides a full set of configuration options available in Code-First.**
- Data annotation attributes are included in the System.ComponentModel.DataAnnotations and System.ComponentModel.DataAnnotations.Schema namespaces in EF 6 as well as in EF Core.

Fluent API

- Entity Framework Fluent API is used to configure domain classes to override conventions. EF Fluent API is based on a **Fluent API design pattern (a.k.a Fluent Interface)** where the result is formulated by **method chaining**.

- **As a developer it is very important to understand what a design pattern is.**

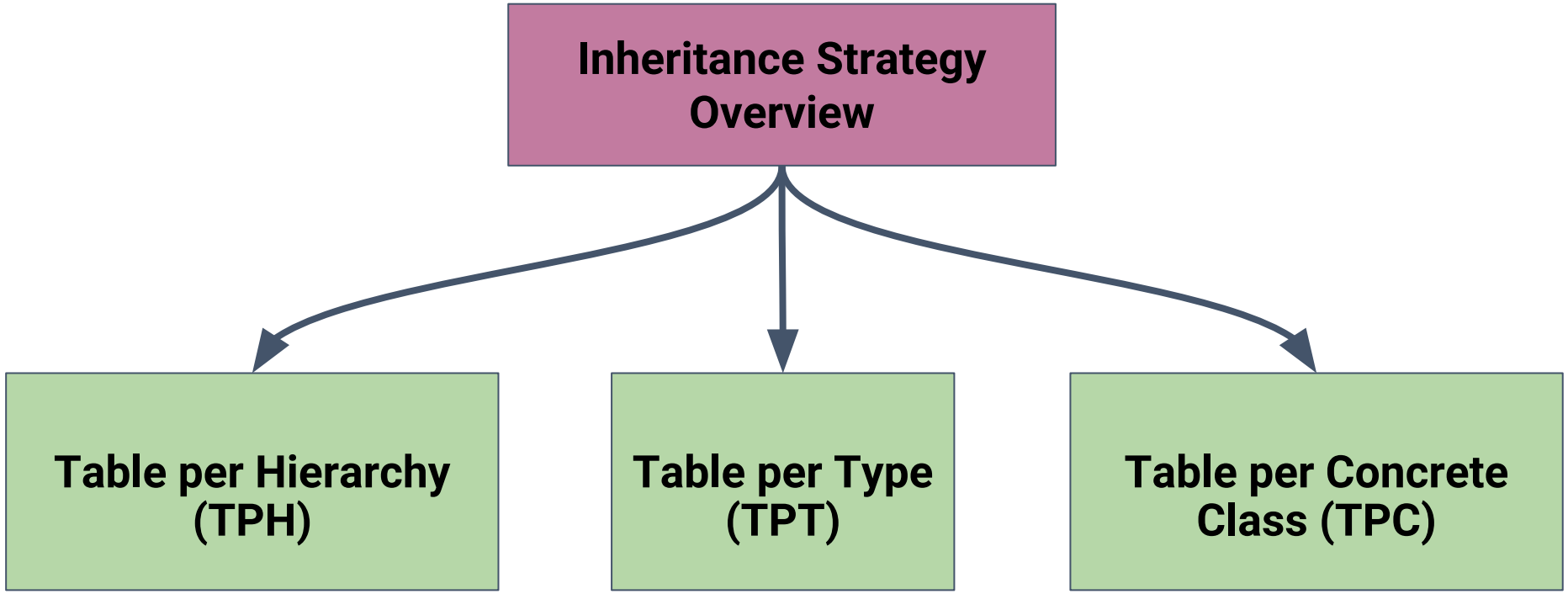
- In software engineering, a fluent interface (as first coined by Eric Evans and Martin Fowler) is a method for designing object oriented APIs b**ased extensively on method chaining** with the goal of making the readability of the source code close to that of ordinary written prose, essentially creating a domain-specific language within the interface.

- In Entity Framework 6, the DbModelBuilder class acts as a Fluent API using which we can configure many different things.

- It provides **more options** of configurations than Data Annotation attributes.

- **Entity Framework gives precedence to Fluent API over Data Annotations attributes.**

Inheritance Strategy Overview



Inheritance Strategy

It is important to understand that object-oriented techniques include **"has a" and "is a"** relationships, whereas SQL-based relational model has only a **"has a" relationship** between tables.

There are three different approaches to represent an inheritance hierarchy in Code-First:

- **Table per Hierarchy (TPH)**: This approach suggests one table for the entire class inheritance hierarchy. The table includes a discriminator column which distinguishes between inheritance classes. This is a default inheritance mapping strategy in Entity Framework.
- **Table per Type (TPT)**: This approach suggests a separate table for each domain class.
- **Table per Concrete Class (TPC)**: This approach suggests one table for one concrete class, but not for the abstract class. So, if you inherit the abstract class in multiple concrete classes, then the properties of the abstract class will be part of each table of the concrete class.

Table per Hierarchy (TPH)

- An entire class hierarchy can be mapped to a **single table**. This table includes columns for all properties of all classes in the hierarchy. The concrete subclass represented by a particular row is identified by the value of a type discriminator column. You don't have to do anything special in Code First to enable TPH. It's the default inheritance mapping strategy.

- This mapping strategy is a **winner in terms of both performance and simplicity.** It's the best-performing way to represent polymorphism—both polymorphic and non polymorphic queries perform well—and it's even easy to implement by hand. Ad-hoc reporting is possible without complex joins or unions. Schema evolution is straightforward.

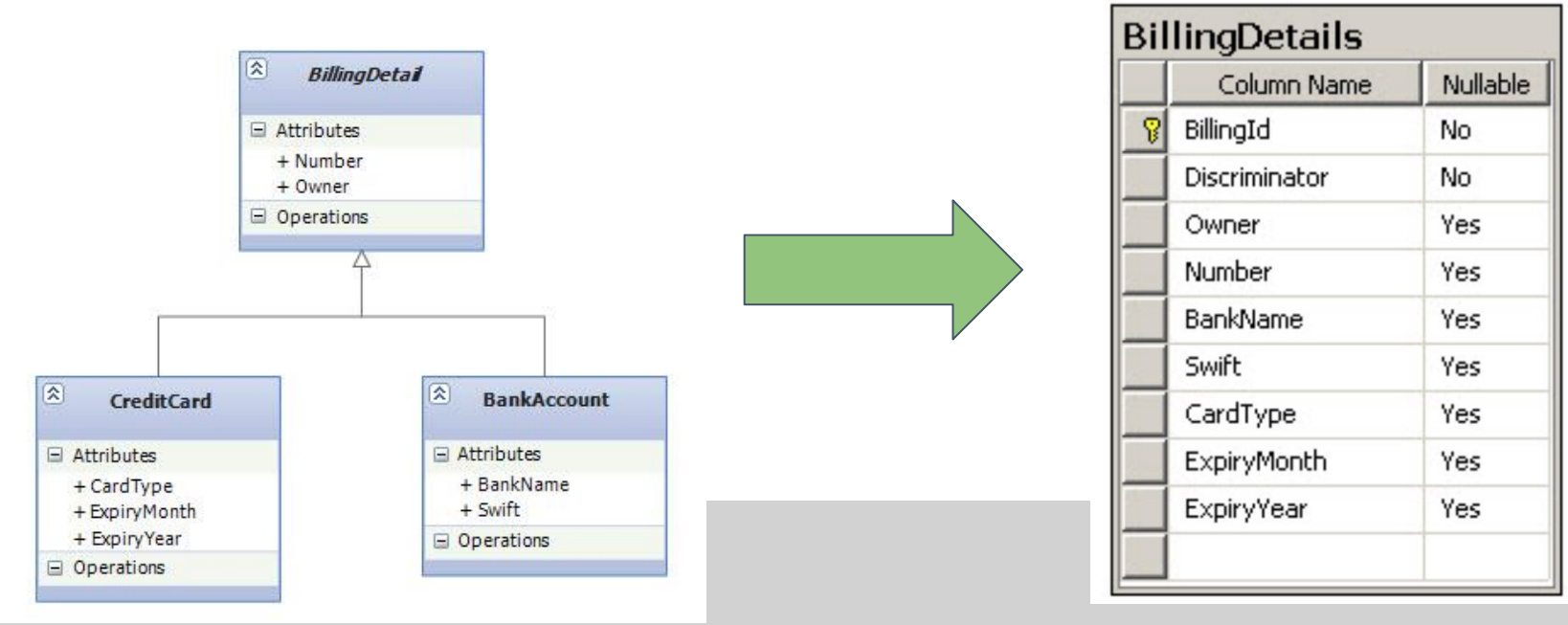
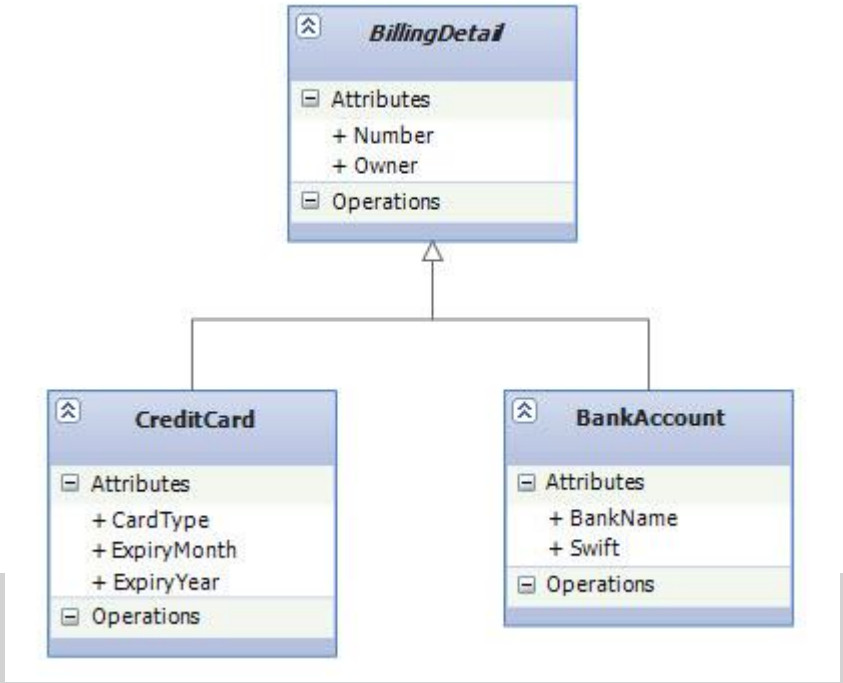


Table per Type (TPT)

- Table per Type is about representing inheritance relationships as **relational foreign key associations.**
- Every class/subclass that declares persistent properties—including abstract classes—has its own table.
- The table for subclasses contains columns only for each non inherited property (each property declared by the subclass itself) along with a primary key that is also a foreign key of the base class table.
- The primary advantage of this strategy is that the **SQL schema is normalized.**



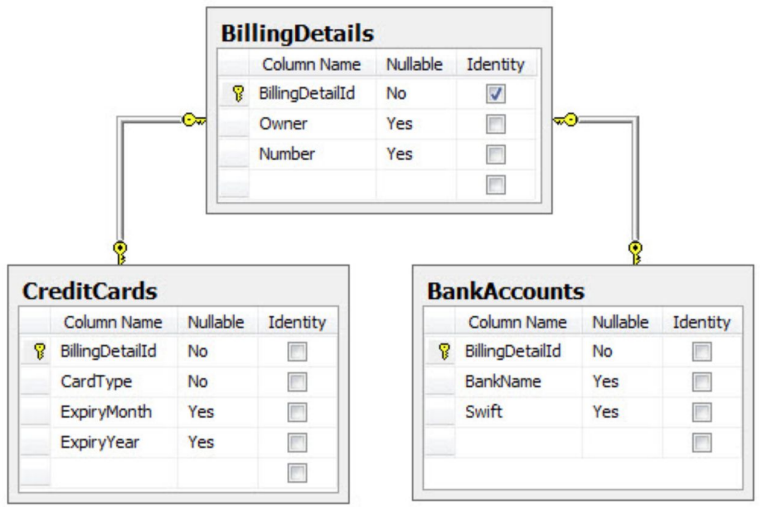
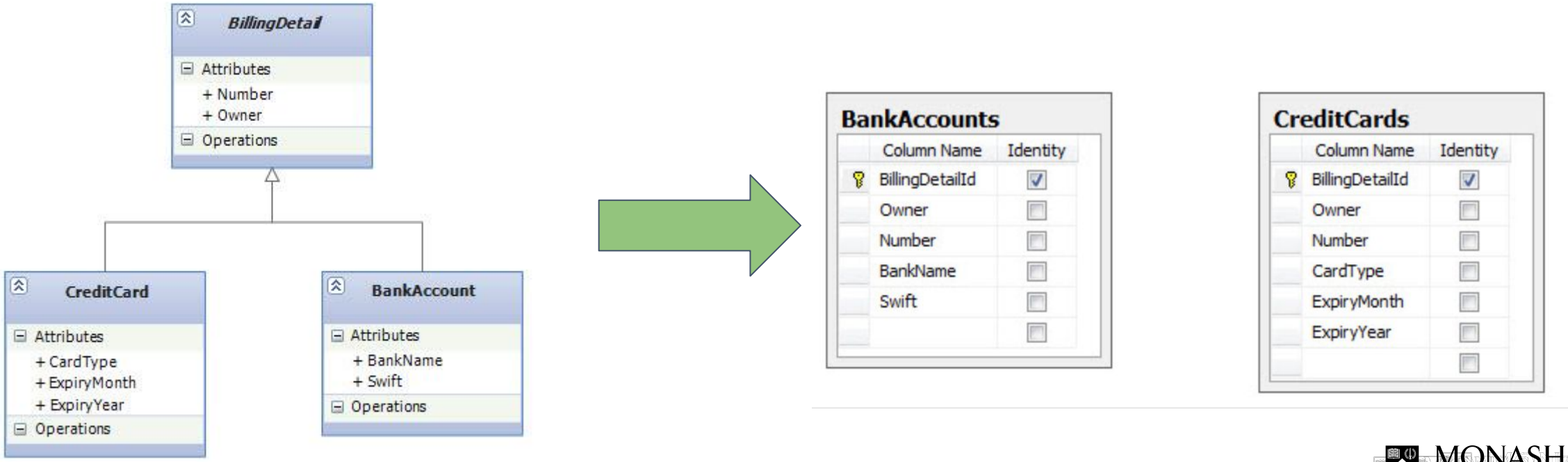


Table per Concrete Type (TPC)

- In Table per Concrete type (aka Table per Concrete class) we use exactly one table for each (non abstract) class.
- All properties of a class, including inherited properties, can be mapped to columns of this table.
- The SQL schema is **not aware of the inheritance; effectively, we've mapped two unrelated tables to a more expressive class structure.**



Choosing an inheritance strategy

- There is no one single "best strategy fits all scenarios" exists.
- Each of the approaches have their own advantages and drawbacks.
- These are the general guidelines.
  - If you don’t require polymorphic associations or queries, lean toward TPC
  - If you do require polymorphic associations or queries, and subclasses declare relatively few properties (particularly if the main difference between subclasses is in their behavior), lean toward TPH.
  - If you do require polymorphic associations or queries, and subclasses declare many properties (subclasses differ mainly by the data they hold), lean toward TPT. Or, depending on the width and depth of your inheritance hierarchy and the possible cost of joins versus unions, use TPC.

Advantages & Disadvantages of Code First

Microsoft Entity Framework

Advantages

- You are able to “version” control your database. This is because it is done via code first, there is a version control record of it via SVN, Git or Mercurial.
- Code First reduces the amount of automatically generated code significantly.
- This is more of a “developer” centric approach.

Disadvantages

- Code First is slightly difficult if you are new to the C# environment and you already know SQL. (This means that, if you have a DBA, creating the database first will be faster)
- Code First heavily depends on the architecture of the systems in use, using it may be more difficult in scenarios where the database layer is important. (Data Centric Approach might be better at times)

Advantages & Disadvantages of Database First

Note that this is Database First not Model First.

Advantages

- It assumes that the database design does not change over time.
- Knowledge of SQL will make this task significantly easier in comparison to Code First.

Disadvantages

- It is required to have an understanding of SQL to create the Database. The reason certain developers dislike this is because they assume that with proper drivers that can connect to different databases they can avoid writing SQL for specific databases.
- Changes to the database can negatively impact development as auto-generated codes may not be working as intended. (Additional features need to be implemented with adjusting the text template created)
- These also sometimes lack the correct annotation attributes.

Code First vs Model First

- At the end of the day when deciding whether to go Code First or Model First, various factors needs to be taken into consideration and it is not as easy to decide.
- Some of the perceived benefits may be disadvantages and some of the disadvantages may be none at all.
- **The skills of the developers, database administrators and the current system architecture** needs to be taken into consideration before making a decision.
- So, there is no one “go to” strategy when making this decision.
- On smaller and personal projects however, it will be much more clear, as you would be the person making the decisions. **(This will be evident for your Portfolio work)**

Querying in Entity Framework

- You can build and execute queries using Entity Framework to fetch the data from the underlying database.

- EF 6 supports different types of queries which in turn convert into SQL queries for the underlying database.

- Entity framework supports three types of queries:
  - LINQ-to-Entities
  - Entity SQL
  - Native SQL

Eager Loading

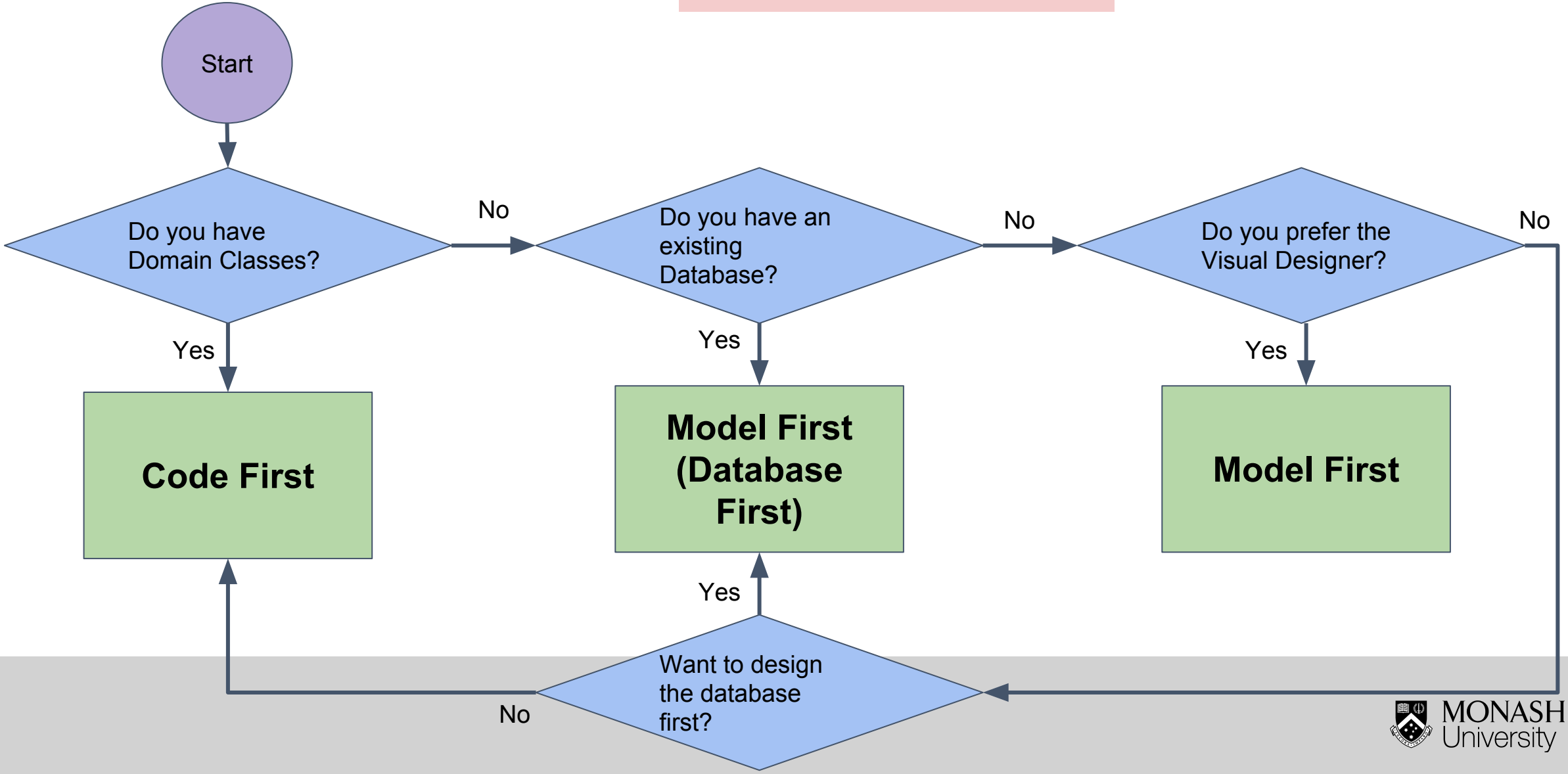
- Eager loading is the process whereby a query for one type of entity also loads related entities as part of the query.
- This is done so that we do not need to write a separate query for the related entity type.
- Eager loading is achieved using the Include() method.
- For example, the if you have an entity called blog and this blog may have zero or many post, once the blog is loaded, all its posts will be loaded as well.
- It is also possible to eagerly load multiple levels of related entities.

Lazy Loading

- Lazy loading is the process whereby an entity or collection of entities is automatically loaded from the database the first time that a property referring to the entity/entities is accessed.
- The loading of related data, only happens when it is request.
- For lazy loading to work EF has to create a proxy object that overrides your virtual properties with an implementation that loads the referenced entity when it is first accessed. If you don't mark the property as virtual then lazy loading won't work with it.
- When using POCO entity types, lazy loading is achieved by creating instances of derived proxy types and then **overriding virtual properties to add the loading hook.** For example, when using the Blog entity class the related Posts will be loaded the first time the Posts navigation property is accessed.
- Lazy loading and serialization don’t mix well, and if you aren’t careful you can end up querying for your entire database just because lazy loading is enabled.

A General Guideline

Please remember this is a general guideline, real life decisions are often times way more complex than this….



ASP.NET Scaffolding

- ASP.NET Scaffolding is a **code generation framework** for ASP.NET Web applications.
- It makes it easy to add boilerplate code to your project that interacts with a data model.
- You can add scaffolding to your project when you want to quickly add code that interacts with data models. Using scaffolding can reduce the amount of time to develop standard data operations in your project.
- With scaffolding we can quickly add needed controllers and view that will interact with our models. So in way, we can shortcut the development process.

In the labs...

You will learn how to use

- The Code First approach
- The Model First approach
- Importance of good naming conventions during the design
- Understand how easily you can develop a data driven website