# Entity Framework Core

# Agenda

- Introduction to ORM and EF
- Introduction to EF Core
- EF Core version History
- What EF Core does not support
- EF Core New features
- EF Core Data Modeling

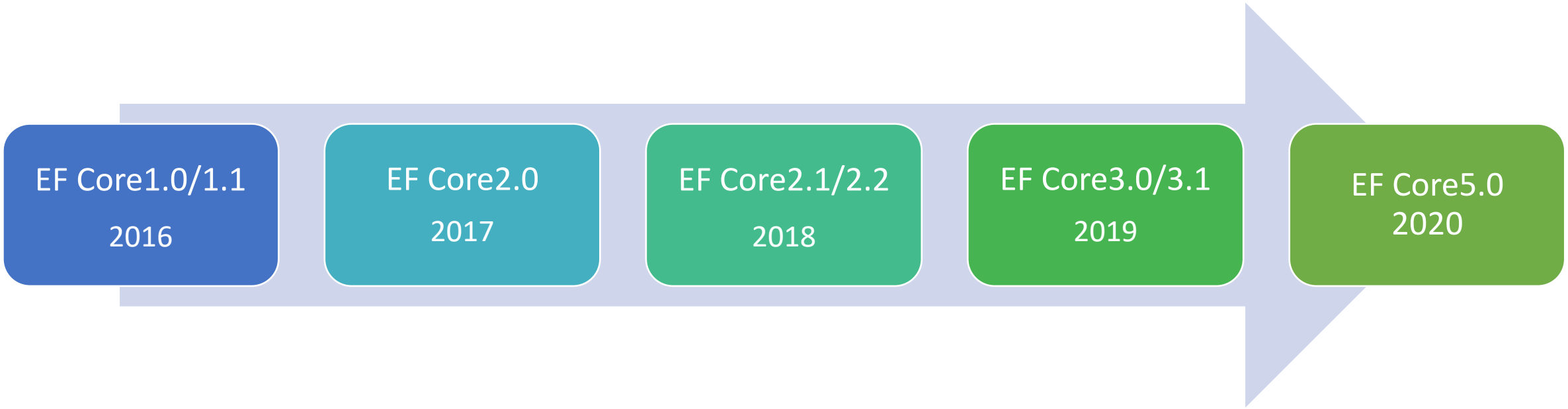
# Introduction to ORM

- Map database tables to classes and objects.
- Helps to query DB in object-oriented fashion.
- Used by many programming languages like C#, Java, Php etc.
- Provides type Safety at compile time.

# Entity Framework Core

- A Light weight and extensible version of Entity Framework
- Not an update to Entity Framework
- Built from scratch to query data on cross-platform environment
- Open Source

# Entity Framework Core Version History



# Where EF Core Can be Used?

- .NET 4.7.2 (EF Core 2.0) based Application
- .NET Core based Application
- Cosmos DB, SQLite
- Oracle, MySQL, SQL Server
- DB2, PostgreSQL, MariaDB

# What EF Core Does not Support?

- No EDMX(XML) based modeling
- Automatic Migration
- Entity SQL

# EF Core Data Modeling

- Describes a model by using C# classes then create DB
- Works entirely in an object-oriented fashion not worry about the structure of the DB.
- Provides full control over the code

```
[Table("Employees", Schema = "dbo")]
0 references
public class Employee
{
    [Key]
    0 references
    public int EmployeeId { get; set; }
    [Required]
    [Column(TypeName = "varchar(200)")]
    0 references
    public string Name { get; set; }
}
```

# EF Core Built-In Conventions

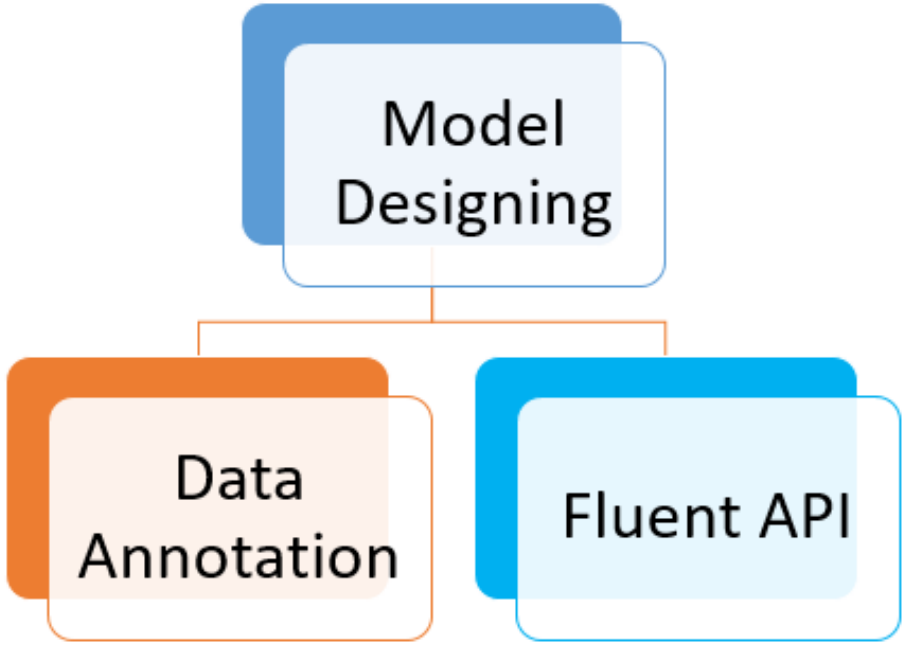
- Naming Convention
- Type Convention
- Primary Key Convention
- Relationship Convention

```
//
public class Category {
 public int CategoryId {get; set;} // primary key
 public string Name {get; set;} // nvarchar(max)

 // optional: navigation property
 public virtual ICollection<Product> Products {get; set;}
}
//
public class Product {
 public int ProductId {get; set;} // primary key
 public string Name {get; set;} // nvarchar(max)
 public decimal UnitPrice {get; set;} // decimal(18,2)
 public int CategoryId {get; set;} // foreign key

 // navigation property
 public virtual Category Category {get; set;}
}
```

# EF Core Data Modeling

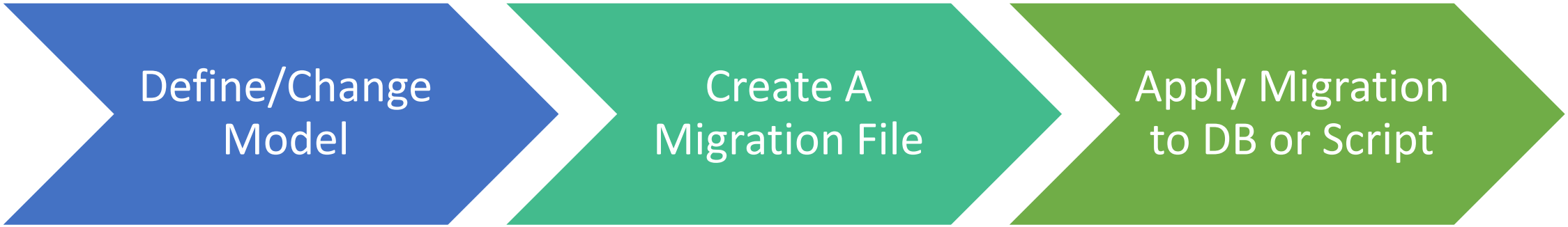



```
[Key]
0 references
public int EmployeeId { get; set; }
[Required]
[Column(TypeName = "varchar(200)")]
0 references
public string Name { get; set; }
```

```
builder.Entity<Employee>().HasKey(e => e.EmployeeId);
builder.Entity<Employee>().Property(e => e.Name)
                          .IsRequired()
                          .IsUnicode(false)
                          .HasMaxLength(200);
```

DotNetTricks

# EF Core Migration




add-migration <migration_name>

update-database -verbose

script-migration

DotNetTricks

# Script Migration

- Used to generate SQL DDL scripts
- Good for Production database

```
>Script-Migration
>Script-Migration -From [migrationName] -To [migrationName]
```

# Script Migration vs. Update Database

- Script Migration
  - Provide more control
  - Good for Production database
- Update Database
  - Good for local development database

# Database Reverse Engineering

```
> scaffold-dbcontext "data source=Shailendra\SqlExpress;
initial catalog=EFDB; persist security info=True;user id=sa;
password=dotnettricks"
Microsoft.EntityFrameworkCore.SqlServer
```