**Code First Approach with Azure Database**

In software development, the code-first approach prioritizes defining your data model within your application code using programming languages like C# or Java. This model serves as the blueprint for your database schema, offering several advantages:

* **Stronger Domain Modeling:** You can represent your real-world entities and relationships accurately, leading to a more intuitive and maintainable codebase.
* **Schema Management:** Automatic migrations track and apply changes to the database schema, streamlining database evolution and simplifying deployments.
* **Flexibility:** You can tailor the database structure to your specific application needs, while also adapting to future requirements.

When using this approach with Azure Database, we'll primarily leverage tools like Entity Framework Core (EF Core), a popular object-relational mapper (ORM) that simplifies data access and interactions between your application and the database.

**Step-by-Step Guide:**

1. **Project Setup:**
   * Create a new project in your preferred IDE (e.g., Visual Studio).
   * Install the required NuGet packages:
     + Microsoft.EntityFrameworkCore.SqlServer (or corresponding provider for your chosen database type)
     + Microsoft.EntityFrameworkCore.Design (for migrations)
2. **Define Your Model:**
   * Create C# classes representing your domain entities.
   * Use attributes or fluent API to map these classes to database tables and columns.
   * Define relationships between entities (e.g., one-to-many, many-to-many) using appropriate navigation properties.
3. **Create an Azure Database:**
   * Go to the Azure portal and create an Azure SQL Database instance.
   * Choose the desired pricing tier, performance settings, and region.
   * Ensure appropriate firewall rules are in place to allow your application to connect.
4. **Set Up Connection String:**
   * In your application's configuration (e.g., appsettings.json), store the connection string to your Azure Database.
   * This string includes information like server name, database name, credentials, and potentially encryption settings.
5. **Implement Data Access Logic:**
   * Instantiate a DbContext object using the connection string.
   * Use DbContext methods to perform CRUD operations (Create, Read, Update, Delete) on your data entities.
   * Take advantage of EF Core's built-in query capabilities for more complex data retrieval.
6. **Enable Migrations:**
   * Run the command dotnet ef migrations add InitialCreate in your project's terminal to generate an initial migration file.
   * This file tracks the changes you've made to your model and creates SQL scripts to apply them to the database.
   * Review the generated migration file to understand the exact changes being made.
7. **Apply Migrations:**
   * Run the command dotnet ef database update to execute the migration and update the Azure Database schema accordingly.
   * This step creates the tables and columns based on your model definitions.

**Example Code (C#):**

C#

// DbContext class

public class MyDbContext : DbContext

{

public MyDbContext(DbContextOptions<MyDbContext> options) : base(options) {}

public DbSet<User> Users { get; set; }

public DbSet<Product> Products { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<User>(entity =>

{

entity.HasKey(u => u.Id);

entity.Property(u => u.Name).HasMaxLength(50);

entity.HasMany(u => u.Orders).WithOne(o => o.User);

});

modelBuilder.Entity<Product>(entity =>

{

entity.HasKey(p => p.Id);

entity.Property(p => p.Name).HasMaxLength(100);

entity.Property(p => p.Price).HasPrecision(18, 2);

entity.HasMany(p => p.OrderItems).WithOne(oi => oi.Product);

});

}

}

// Data access example

public async Task CreateUser(User user)

{

using (var context = new MyDbContext())

{

context.Users.Add(user);

await context.SaveChangesAsync();

}

}

Use code [with caution.](https://gemini.google.com/faq#coding)

content\_copy

**Key Considerations:**

* **Azure Portal Management:** While this guide focuses on code-first, the Azure portal also offers schema management tools. Consider employing a combination for flexibility and ease of use.
* **Security:**

#### **Sources**

[info](https://gemini.google.com/faq#citation)

1. [github.com/2236721325/Learning](https://github.com/2236721325/Learning)
2. [github.com/Alexandr133123/WebAppRepository](https://github.com/Alexandr133123/WebAppRepository)
3. [github.com/AlexandreWN/Prova2](https://github.com/AlexandreWN/Prova2)