# NoSQL Databases

In this demo we are going to create an application that creates a MongoDB (no SQL) version of the Film Review app we looked at previously.

# Project Set Up

Before creating anything in Visual Studio or Visual Studio Code you must first create a MongoDB Atlas database by carrying out the following steps:

1. Sign in to cloud.mongodb.com Register and sign in
2. Create an organisation calling it QACS and take the defaults
3. Create a new project calling it "QACS Project" and take the defaults.
4. Select "Build a Database" and select to Create a "Free Shared" database
5. Pick AWS , Ireland (eu-west-1) as the cloud provider & region. Leave the rest to default and click Create Cluster.
6. Choose to authenticate via the Username and Password option and enter a username of demo-user and an auto-generated secure password, for example: demo-user / WzOrLL6kGyXGw1xq) . Copy and make a note of the password and then select Create User.
7. Click the "Add My Current IP Address" button to add it to your IP access list.
8. Click Finish and Close.
9. Click 'connect' on the database deployments page. Select "Connect your application" pick the C#/.NET driver and version 2.13 or later.
10. Select the "Include full driver code example" check box and **copy the sample code (you will paste this into your C# project in a later step)**.
11. Click close
12. On the Database Deployment page select the period ellipses button for your cluster (cluster0) and select "Load Sample Dataset"

## Setup Steps for Visual Studio

From within VS (using point and click):

* Open the WorkingWithDatabases solution
* Add console project called NoSQLDBDemo.csproj to the solution
* Use NuGet to add a reference to the following packages:
  + MongoDB.Driver

## Setup Steps for Visual Studio Code

From Command Prompt:

* Create console project and add it to the current solution by typing:

dotnet new console --output no\_sql\_demo

dotnet sln add no\_sql\_demo

* Install the following packages by typing:

dotnet add no\_sql\_demo package mongodb.driver

## Now do the following regardless of project type:

**Add code that creates a FilmReviews database with Films and Reviews collections**

* Replace the Program.cs file’s content with the following:

using MongoDB.Driver;

using NoSQLFilmReviews.Models;

namespace NoSQLDBDemo

{

internal class Program

{

static void Main(string[] args)

{

}

}

}

* Paste the code you copied in step 10 above into the Main method. It will look something like the following (the connection string will have some differences):

var settings = MongoClientSettings.FromConnectionString(

"mongodb+srv://demo-user:5Z8yRIFkTsm4vcya@cluster0.s7jjkdj.mongodb.net/”

+ “?retryWrites=true&w=majority");

settings.ServerApi = new ServerApi(ServerApiVersion.V1);

var client = new MongoClient(settings);

var database = client.GetDatabase("test");

* Add the following code to the foot of the Main method:

// Get list of databases within the cluster connected to

var databases = client.ListDatabases().ToList();

foreach (var db in databases)

{

Console.WriteLine(db["name"]);

}

* Review the code and try to work out what it’s doing.
* Build and run the code and confirm it behaves as you expected.

**Add logic to create a Film Review database**

* Add a Models folder to the project.
* Add a Film class to the project.
* Replace the code in the Film class file with the following:

namespace NoSQLDBDemo.Models

{

internal class Film

{

public int Id { get; set; }

public string? Title { get; set; }

public string? Overview { get; set; }

public DateTime? ReleaseDate { get; set; }

public long? Revenue { get; set; }

public string? HomepageURL { get; set; }

public List<Review>? Reviews { get; set; }

}

}

* Add a Review class to the project.
* Replace the code in the Review class file with the following:

namespace NoSQLDBDemo.Models

{

internal class Review

{

public int Id { get; set; }

public int FilmId { get; set; }

public string? Commentary { get; set; }

public int Rating { get; set; }

}

}

* Add the following code to the foot of the Main method of the Program class:

Film film1 = new Film {

Id = 1,

Title = "Titanic",

Overview = "Blab, blah ,blah",

ReleaseDate = new DateTime(1997, 11, 18),

Revenue = 1845034188,

Reviews = new List<Review>()

};

Film film2 = new Film {

Id = 2,

Title = "ET",

Overview = "An extra-terrestrial comes to earth, gets sick, phones home and is rescued",

ReleaseDate = new DateTime(2004, 03, 19),

Revenue = 72258126, Reviews = new List<Review>()

};

database = client.GetDatabase("FilmReviews");

var reviews = database.GetCollection<Review>("Reviews");

var films = database.GetCollection<Film>("Films");

// Ensure fresh start each time code is run

reviews.DeleteMany(r => true);

films.DeleteMany(f => true);

films.InsertOne(film1);

films.InsertOne(film2);

* Review the code and try to work out what it’s doing.
* Build and run the code and confirm it behaves as you expected.
* Review the content of the database by returning to the cloud.mongodb.com web site anddrilling into the QACS project, cluster0, Browse Collections and expanding the FilmReviews database and seeing what is in the Films Collection.
* Next we need to add some Reviews to the database. Add the following code to the Main method, **just before the two (final) lines that Insert the two films into the Films collection**:

Review review = new Review

{

Id = 1000,

FilmId = film1.Id,

Commentary = "Truly awful",

Rating = 2,

};

reviews.InsertOne(review);

film1.Reviews.Add(review);

review = new Review

{

Id = 1001,

FilmId = film1.Id,

Commentary = "Gives you a sinking feeling in your stomach",

Rating = 8,

};

reviews.InsertOne(review);

film1.Reviews.Add(review);

review = new Review

{

Id = 1002,

FilmId = film1.Id,

Commentary = "Chilly",

Rating = 6,

};

reviews.InsertOne(review);

film1.Reviews.Add(review);

review = new Review

{

Id = 1003,

FilmId = film2.Id,

Rating = 9,

};

reviews.InsertOne(review);

film2.Reviews.Add(review);

* Review the code and try to work out what it’s doing.
* Build and run the code and confirm it behaves as you expected.
* Review the content of the database by returning to the cloud.mongodb.com web site anddrilling into the QACS project, cluster0, Browse Collections and expanding the FilmReviews database and seeing what is in the Reviews and Films Collections.

**Add code that queries the database**

* Add the following code to the foot of the Main method of the Program class:

// Get all films with at least 2 reviews

List<Film> filmsWithReviews = films.Find(f => f.Reviews.Count >= 2).ToList();

foreach (Film film in filmsWithReviews)

{

System.Console.WriteLine(film.Title);

foreach (Review rev in film.Reviews)

{

Console.WriteLine($"{rev.Commentary} Rating ({rev.Rating})");

}

}

* Review the code and try to work out what it’s doing.
* Build and run the code and confirm it behaves as you expected.

**Add code that removes a film from the database**

* Add the following code to the foot of the Main method of the Program class:

// Delete an item from a collection

films.FindOneAndDelete(f => f.Title == "ET");

* Review the code and try to work out what it’s doing.
* Build and run the code and confirm it behaves as you expected.

**Add code that inserts two films into the database with missing/additional values**

* Add the following code to the foot of the Main method of the Program class:

films.InsertOne(new Film

{

Id = 3,

Title = "Up",

Overview = "The ultimate house move",

ReleaseDate = new DateTime(2012, 04, 21)

});

films.InsertOne(new Film

{

Id = 4,

Title = "Where Eagles Dare",

Overview = "A Wonkymotion remake of the original. Truly awe inspiring.",

ReleaseDate = new DateTime(2022, 04, 21),

HomepageURL = "https://www.youtube.com/watch?v=ENrgZ4KAnNw"

});

// Retrieve films and show revenue, even though some films don't have them

foreach (Film film in films.Find(d => true).ToList())

{

Console.WriteLine($"{film.Title} has a revenue of "

+ $"${(film.Revenue != null ? film.Revenue : 0)}, "

+ $"Hompage: {(film.HomepageURL != null ? film.HomepageURL : "No Hompage")}");

}

* Review the code and try to work out what it’s doing.
* Build and run the code and confirm it behaves as you expected.