




ENTITY FRAMEWORK HANDS-ON LABS



Venkat Shiva Reddy
SKILL ASSURE

Database Apps, Entity Framework, SQL Server, XML & JSON – Practical Hands On Lab

The goal of this **practical hands on lab** is to learn how to **develop database applications with C#, Entity Framework and SQL Server**. If you follow the tutorial steps below, you will learn how to create a database, map the database to EF data model (database first and code first), query the database, import and export data, parse and create XML and JSON.

Note: the source code in all examples below is intentionally given as image to avoid copy-pasting.

Requirements

- Basic **C#** programming skills (C#, .NET and Visual Studio basics) – learn at <http://www.introprogramming.info/english-intro-csharp-book/read-online/>
- Basic **SQL** and **SQL Server** skills – learn at <https://softuni.bg/courses/databases/>, <http://www.w3schools.com/sql/>
- Basic ORM and **Entity Framework** skills – learn at <http://www.entityframeworktutorial.net>, <https://softuni.bg/courses/database-applications/>

Problem 0. Restore the Database

You are given a **MS SQL Server database** “Geography” holding continents, countries, currencies, monasteries and rivers, available as **SQL script**. **Restore the database “Geography”** by running the below provided SQL script:

```
-- Create the database [Geography] if it does not exist
IF NOT EXISTS (SELECT name FROM master.dbo.sysdatabases WHERE name =
'Geography')
    CREATE DATABASE Geography
GO

USE Geography
GO

-- Drop all existing Geography tables, so that we can create them from
scratch
IF OBJECT_ID('Monasteries') IS NOT NULL
    DROP TABLE Monasteries
IF OBJECT_ID('CountriesRivers') IS NOT NULL
```

```

    DROP TABLE CountriesRivers
IF OBJECT_ID('Rivers') IS NOT NULL
    DROP TABLE Rivers
IF OBJECT_ID('Countries') IS NOT NULL
    DROP TABLE Countries
IF OBJECT_ID('Continents') IS NOT NULL
    DROP TABLE Continents
IF OBJECT_ID('Currencies') IS NOT NULL
    DROP TABLE Currencies

-- Create tables
CREATE TABLE Continents(
    ContinentCode char(2) NOT NULL,
    ContinentName nvarchar(50) NOT NULL,
    CONSTRAINT PK_Continents PRIMARY KEY CLUSTERED (ContinentCode)
)
GO

CREATE TABLE Countries(
    CountryCode char(2) NOT NULL,
    IsoCode char(3) NOT NULL,
    CountryName varchar(45) NOT NULL,
    CurrencyCode char(3),
    ContinentCode char(2) NOT NULL,
    Population int NOT NULL,
    AreaInSqKm int NOT NULL,
    Capital varchar(30) NOT NULL,
    CONSTRAINT PK_Countries PRIMARY KEY CLUSTERED (CountryCode)
)
GO

CREATE TABLE Currencies(
    CurrencyCode char(3) NOT NULL,
    Description nvarchar(200) NOT NULL,
    CONSTRAINT PK_Currencies PRIMARY KEY CLUSTERED (CurrencyCode)
)
GO

CREATE TABLE Rivers(
    Id int IDENTITY NOT NULL,
    RiverName nvarchar(50) NOT NULL,
    Length int NOT NULL,
    DrainageArea int,
    AverageDischarge int,
    Outflow nvarchar(50) NOT NULL,
    CONSTRAINT PK_Rivers PRIMARY KEY CLUSTERED (Id)
)
GO

CREATE TABLE CountriesRivers(
    RiverId int NOT NULL,
    CountryCode char(2) NOT NULL,
    CONSTRAINT PK_CountriesRivers PRIMARY KEY CLUSTERED (CountryCode, RiverId)
)
GO

```

```

CREATE TABLE Monasteries(
    Id INT PRIMARY KEY IDENTITY,
    Name NVARCHAR(50),
    CountryCode CHAR(2)
)
GO

```

```
-- Insert table data
```

```
INSERT Continents (ContinentCode, ContinentName) VALUES
```

```

(N'AF', N'Africa'),
(N'AN', N'Antarctica'),
(N'AS', N'Asia'),
(N'EU', N'Europe'),
(N'NA', N'North America'),
(N'OC', N'Oceania'),
(N'SA', N'South America')

```

```
INSERT Countries (CountryCode, IsoCode, CountryName, CurrencyCode,
ContinentCode, Population, AreaInSqKm, Capital) VALUES
```

```

(N'AD', N'AND', N'Andorra', N'EUR', N'EU', 84000, 468, N'Andorra la Vella'),
(N'AE', N'ARE', N'United Arab Emirates', N'AED', N'AS', 4975593, 82880, N'Abu
Dhabi'),
(N'AF', N'AFG', N'Afghanistan', N'AFN', N'AS', 29121286, 647500, N'Kabul'),
(N'AG', N'ATG', N'Antigua and Barbuda', N'XCD', N'NA', 86754, 443, N'St.
John's'),
(N'AI', N'AIA', N'Anguilla', N'XCD', N'NA', 13254, 102, N'The Valley'),
(N'AL', N'ALB', N'Albania', N'ALL', N'EU', 2986952, 28748, N'Tirana'),
(N'AM', N'ARM', N'Armenia', N'AMD', N'AS', 2968000, 29800, N'Yerevan'),
(N'AO', N'AGO', N'Angola', N'AOA', N'AF', 13068161, 1246700, N'Luanda'),
(N'AQ', N'ATA', N'Antarctica', NULL, N'AN', 0, 14000000, N''),
(N'AR', N'ARG', N'Argentina', N'ARS', N'SA', 41343201, 2766890, N'Buenos
Aires'),
(N'AS', N'ASM', N'American Samoa', N'USD', N'OC', 57881, 199, N'Pago Pago'),
(N'AT', N'AUT', N'Austria', N'EUR', N'EU', 8205000, 83858, N'Vienna'),
(N'AU', N'AUS', N'Australia', N'AUD', N'OC', 21515754, 7686850, N'Canberra'),
(N'AW', N'ABW', N'Aruba', N'AWG', N'NA', 71566, 193, N'Oranjestad'),
(N'AX', N'ALA', N'Åland', N'EUR', N'EU', 26711, 1580, N'Mariehamn'),
(N'AZ', N'AZE', N'Azerbaijan', N'AZN', N'AS', 8303512, 86600, N'Baku'),
(N'BA', N'BIH', N'Bosnia and Herzegovina', N'BAM', N'EU', 4590000, 51129,
N'Sarajevo'),
(N'BB', N'BRB', N'Barbados', N'BBD', N'NA', 285653, 431, N'Bridgetown'),
(N'BD', N'BGD', N'Bangladesh', N'BDT', N'AS', 156118464, 144000, N'Dhaka'),
(N'BE', N'BEL', N'Belgium', N'EUR', N'EU', 10403000, 30510, N'Brussels'),
(N'BF', N'BFA', N'Burkina Faso', N'XOF', N'AF', 16241811, 274200,
N'Ouagadougou'),
(N'BG', N'BGR', N'Bulgaria', N'BGN', N'EU', 7148785, 110910, N'Sofia'),
(N'BH', N'BHR', N'Bahrain', N'BHD', N'AS', 738004, 665, N'Manama'),
(N'BI', N'BDI', N'Burundi', N'BIF', N'AF', 9863117, 27830, N'Bujumbura'),
(N'BJ', N'BEN', N'Benin', N'XOF', N'AF', 9056010, 112620, N'Porto-Novo'),
(N'BL', N'BLM', N'Saint Barthélemy', N'EUR', N'NA', 8450, 21, N'Gustavia'),
(N'BM', N'BMU', N'Bermuda', N'BMD', N'NA', 65365, 53, N'Hamilton'),
(N'BN', N'BRN', N'Brunei', N'BND', N'AS', 395027, 5770, N'Bandar Seri
Begawan'),
(N'BO', N'BOL', N'Bolivia', N'BOB', N'SA', 9947418, 1098580, N'Sucre'),
(N'BQ', N'BES', N'Bonaire', N'USD', N'NA', 18012, 328, N''),

```

(N'BR', N'BRA', N'Brazil', N'BRL', N'SA', 201103330, 8511965, N'Brasília'),
(N'BS', N'BHS', N'Bahamas', N'BSD', N'NA', 301790, 13940, N'Nassau'),
(N'BT', N'BTN', N'Bhutan', N'BTN', N'AS', 699847, 47000, N'Thimphu'),
(N'BV', N'BVT', N'Bouvet Island', N'NOK', N'AN', 0, 49, N''),
(N'BW', N'BWA', N'Botswana', N'BWP', N'AF', 2029307, 600370, N'Gaborone'),
(N'BY', N'BLR', N'Belarus', N'BYR', N'EU', 9685000, 207600, N'Minsk'),
(N'BZ', N'BLZ', N'Belize', N'BZD', N'NA', 314522, 22966, N'Belmopan'),
(N'CA', N'CAN', N'Canada', N'CAD', N'NA', 33679000, 9984670, N'Ottawa'),
(N'CC', N'CCK', N'Cocos Islands', N'AUD', N'AS', 628, 14, N'West Island'),
(N'CD', N'COD', N'Democratic Republic of the Congo', N'CDF', N'AF', 70916439, 2345410, N'Kinshasa'),
(N'CF', N'CAF', N'Central African Republic', N'XAF', N'AF', 4844927, 622984, N'Bangui'),
(N'CG', N'COG', N'Republic of the Congo', N'XAF', N'AF', 3039126, 342000, N'Brazzaville'),
(N'CH', N'CHE', N'Switzerland', N'CHF', N'EU', 7581000, 41290, N'Berne'),
(N'CI', N'CIV', N'Ivory Coast', N'XOF', N'AF', 21058798, 322460, N'Yamoussoukro'),
(N'CK', N'COK', N'Cook Islands', N'NZD', N'OC', 21388, 240, N'Avarua'),
(N'CL', N'CHL', N'Chile', N'CLP', N'SA', 16746491, 756950, N'Santiago'),
(N'CM', N'CMR', N'Cameroon', N'XAF', N'AF', 19294149, 475440, N'Yaoundé'),
(N'CN', N'CHN', N'China', N'CNY', N'AS', 1330044000, 9596960, N'Beijing'),
(N'CO', N'COL', N'Colombia', N'COP', N'SA', 47790000, 1138910, N'Bogotá'),
(N'CR', N'CRI', N'Costa Rica', N'CRC', N'NA', 4516220, 51100, N'San José'),
(N'CU', N'CUB', N'Cuba', N'CUP', N'NA', 11423000, 110860, N'Havana'),
(N'CV', N'CPV', N'Cape Verde', N'CVE', N'AF', 508659, 4033, N'Praia'),
(N'CW', N'CUW', N'Curacao', N'ANG', N'NA', 141766, 444, N'Willemstad'),
(N'CX', N'CXR', N'Christmas Island', N'AUD', N'AS', 1500, 135, N'The Settlement'),
(N'CY', N'CYP', N'Cyprus', N'EUR', N'EU', 1102677, 9250, N'Nicosia'),
(N'CZ', N'CZE', N'Czech Republic', N'CZK', N'EU', 10476000, 78866, N'Prague'),
(N'DE', N'DEU', N'Germany', N'EUR', N'EU', 81802257, 357021, N'Berlin'),
(N'DJ', N'DJI', N'Djibouti', N'DJF', N'AF', 740528, 23000, N'Djibouti'),
(N'DK', N'DNK', N'Denmark', N'DKK', N'EU', 5484000, 43094, N'Copenhagen'),
(N'DM', N'DMA', N'Dominica', N'XCD', N'NA', 72813, 754, N'Roseau'),
(N'DO', N'DOM', N'Dominican Republic', N'DOP', N'NA', 9823821, 48730, N'Santo Domingo'),
(N'DZ', N'DZA', N'Algeria', N'DZD', N'AF', 34586184, 2381740, N'Algiers'),
(N'EC', N'ECU', N'Ecuador', N'USD', N'SA', 14790608, 283560, N'Quito'),
(N'EE', N'EST', N'Estonia', N'EUR', N'EU', 1291170, 45226, N'Tallinn'),
(N'EG', N'EGY', N'Egypt', N'EGP', N'AF', 80471869, 1001450, N'Cairo'),
(N'EH', N'ESH', N'Western Sahara', N'MAD', N'AF', 273008, 266000, N'El Aaiún'),
(N'ER', N'ERI', N'Eritrea', N'ERN', N'AF', 5792984, 121320, N'Asmara'),
(N'ES', N'ESP', N'Spain', N'EUR', N'EU', 46505963, 504782, N'Madrid'),
(N'ET', N'ETH', N'Ethiopia', N'ETB', N'AF', 88013491, 1127127, N'Addis Ababa'),
(N'FI', N'FIN', N'Finland', N'EUR', N'EU', 5244000, 337030, N'Helsinki'),
(N'FJ', N'FJI', N'Fiji', N'FJD', N'OC', 875983, 18270, N'Suva'),
(N'FK', N'FLK', N'Falkland Islands', N'FKP', N'SA', 2638, 12173, N'Stanley'),
(N'FM', N'FSM', N'Micronesia', N'USD', N'OC', 107708, 702, N'Palikir'),
(N'FO', N'FRO', N'Faroe Islands', N'DKK', N'EU', 48228, 1399, N'Tórshavn'),
(N'FR', N'FRA', N'France', N'EUR', N'EU', 64768389, 547030, N'Paris'),
(N'GA', N'GAB', N'Gabon', N'XAF', N'AF', 1545255, 267667, N'Libreville'),
(N'GB', N'GBR', N'United Kingdom', N'GBP', N'EU', 62348447, 244820, N'London'),

(N'GD', N'GRD', N'Grenada', N'XCD', N'NA', 107818, 344, N'St. George's'),
(N'GE', N'GEO', N'Georgia', N'GEL', N'AS', 4630000, 69700, N'Tbilisi'),
(N'GF', N'GUF', N'French Guiana', N'EUR', N'SA', 195506, 91000, N'Cayenne'),
(N'GG', N'GGY', N'Guernsey', N'GBP', N'EU', 65228, 78, N'St Peter Port'),
(N'GH', N'GHA', N'Ghana', N'GHS', N'AF', 24339838, 239460, N'Accra'),
(N'GI', N'GIB', N'Gibraltar', N'GIP', N'EU', 27884, 6.5, N'Gibraltar'),
(N'GL', N'GRL', N'Greenland', N'DKK', N'NA', 56375, 2166086, N'Nuuk'),
(N'GM', N'GMB', N'Gambia', N'GMD', N'AF', 1593256, 11300, N'Banjul'),
(N'GN', N'GIN', N'Guinea', N'GNF', N'AF', 10324025, 245857, N'Conakry'),
(N'GP', N'GLP', N'Guadeloupe', N'EUR', N'NA', 443000, 1780, N'Basse-Terre'),
(N'GQ', N'GNQ', N'Equatorial Guinea', N'XAF', N'AF', 1014999, 28051, N'Malabo'),
(N'GR', N'GRC', N'Greece', N'EUR', N'EU', 11000000, 131940, N'Athens'),
(N'GS', N'SGS', N'South Georgia and the South Sandwich Islands', N'GBP', N'AN', 30, 3903, N'Grytviken'),
(N'GT', N'GTM', N'Guatemala', N'GTQ', N'NA', 13550440, 108890, N'Guatemala City'),
(N'GU', N'GUM', N'Guam', N'USD', N'OC', 159358, 549, N'Hagåtña'),
(N'GW', N'GNB', N'Guinea-Bissau', N'XOF', N'AF', 1565126, 36120, N'Bissau'),
(N'GY', N'GUY', N'Guyana', N'GYD', N'SA', 748486, 214970, N'Georgetown'),
(N'HK', N'HKG', N'Hong Kong', N'HKD', N'AS', 6898686, 1092, N'Hong Kong'),
(N'HM', N'HMD', N'Heard Island and McDonald Islands', N'AUD', N'AN', 0, 412, N''),
(N'HN', N'HND', N'Honduras', N'HNL', N'NA', 7989415, 112090, N'Tegucigalpa'),
(N'HR', N'HRV', N'Croatia', N'HRK', N'EU', 4491000, 56542, N'Zagreb'),
(N'HT', N'HTI', N'Haiti', N'HTG', N'NA', 9648924, 27750, N'Port-au-Prince'),
(N'HU', N'HUN', N'Hungary', N'HUF', N'EU', 9982000, 93030, N'Budapest'),
(N'ID', N'IDN', N'Indonesia', N>IDR', N'AS', 242968342, 1919440, N'Jakarta'),
(N'IE', N'IRL', N'Ireland', N'EUR', N'EU', 4622917, 70280, N'Dublin'),
(N'IL', N'ISR', N'Israel', N>ILS', N'AS', 7353985, 20770, N''),
(N'IM', N'IMN', N'Isle of Man', N'GBP', N'EU', 75049, 572, N'Douglas'),
(N'IN', N'IND', N'India', N'INR', N'AS', 1173108018, 3287590, N>New Delhi'),
(N'IO', N'IOT', N'British Indian Ocean Territory', N'USD', N'AS', 4000, 60, N''),
(N'IQ', N'IRQ', N'Iraq', N'IQD', N'AS', 29671605, 437072, N'Baghdad'),
(N'IR', N'IRN', N'Iran', N'IRR', N'AS', 76923300, 1648000, N'Tehran'),
(N'IS', N>ISL', N'Iceland', N'ISK', N'EU', 308910, 103000, N'Reykjavik'),
(N'IT', N'ITA', N'Italy', N'EUR', N'EU', 60340328, 301230, N'Rome'),
(N'JE', N'JEY', N'Jersey', N'GBP', N'EU', 90812, 116, N>Saint Helier'),
(N>JM', N>JAM', N>Jamaica', N>JMD', N'NA', 2847232, 10991, N'Kingston'),
(N>JO', N>JOR', N>Jordan', N>JOD', N'AS', 6407085, 92300, N'Amman'),
(N>JP', N>JPN', N'Japan', N>JPY', N'AS', 127288000, 377835, N'Tokyo'),
(N>KE', N>KEN', N'Kenya', N>KES', N'AF', 40046566, 582650, N>Nairobi'),
(N>KG', N>KGZ', N>Kyrgyzstan', N>KGS', N'AS', 5508626, 198500, N'Bishkek'),
(N>KH', N>KHM', N>Cambodia', N>KHR', N'AS', 14453680, 181040, N>Phnom Penh'),
(N>KI', N>KIR', N>Kiribati', N>AUD', N'OC', 92533, 811, N>Tarawa'),
(N>KM', N>COM', N>Comoros', N>KMF', N'AF', 773407, 2170, N>Moroni'),
(N>KN', N>KNA', N>Saint Kitts and Nevis', N'XCD', N'NA', 51134, 261, N>Basseterre'),
(N>KP', N>PRK', N>North Korea', N>KPW', N'AS', 22912177, 120540, N>Pyongyang'),
(N>KR', N>KOR', N>South Korea', N>KRW', N'AS', 48422644, 98480, N>Seoul'),
(N>KW', N>KWT', N>Kuwait', N>KWD', N'AS', 2789132, 17820, N>Kuwait City'),
(N>KY', N>CYM', N>Cayman Islands', N>KYD', N'NA', 44270, 262, N>George Town'),
(N>KZ', N>KAZ', N>Kazakhstan', N>KZT', N'AS', 15340000, 2717300, N>Astana'),
(N>LA', N>LAO', N>Laos', N>LAK', N'AS', 6368162, 236800, N>Vientiane'),

(N'LB', N'LBN', N'Lebanon', N'LBP', N'AS', 4125247, 10400, N'Beirut'),
(N'LC', N'LCA', N'Saint Lucia', N'XCD', N'NA', 160922, 616, N'Castries'),
(N'LI', N'LIE', N'Liechtenstein', N'CHF', N'EU', 35000, 160, N'Vaduz'),
(N'LK', N'LKA', N'Sri Lanka', N'LKR', N'AS', 21513990, 65610, N'Colombo'),
(N'LR', N'LBR', N'Liberia', N'LRD', N'AF', 3685076, 111370, N'Monrovia'),
(N'LS', N'LSO', N'Lesotho', N'LSL', N'AF', 1919552, 30355, N'Maseru'),
(N'LT', N'LTU', N'Lithuania', N'EUR', N'EU', 2944459, 65200, N'Vilnius'),
(N'LU', N'LUX', N'Luxembourg', N'EUR', N'EU', 497538, 2586, N'Luxembourg'),
(N'LV', N'LVA', N'Latvia', N'EUR', N'EU', 2217969, 64589, N'Riga'),
(N'LY', N'LYB', N'Libya', N'LYD', N'AF', 6461454, 1759540, N'Tripoli'),
(N'MA', N'MAR', N'Morocco', N'MAD', N'AF', 31627428, 446550, N'Rabat'),
(N'MC', N'MCO', N'Monaco', N'EUR', N'EU', 32965, 1.95, N'Monaco'),
(N'MD', N'MDA', N'Moldova', N'MDL', N'EU', 4324000, 33843, N'Chisinau'),
(N'ME', N'MNE', N'Montenegro', N'EUR', N'EU', 666730, 14026, N'Podgorica'),
(N'MF', N'MAF', N'Saint Martin', N'EUR', N'NA', 35925, 53, N'Marigot'),
(N'MG', N'MDG', N'Madagascar', N'MGA', N'AF', 21281844, 587040,
N'Antananarivo'),
(N'MH', N'MHL', N'Marshall Islands', N'USD', N'OC', 65859, 181.3, N'Majuro'),
(N'MK', N'MKD', N'Macedonia', N'MKD', N'EU', 2062294, 25333, N'Skopje'),
(N'ML', N'MLI', N'Mali', N'XOF', N'AF', 13796354, 1240000, N'Bamako'),
(N'MM', N'MMR', N'Myanmar', N'MMK', N'AS', 53414374, 678500, N'Nay Pyi Taw'),
(N'MN', N'MNG', N'Mongolia', N'MNT', N'AS', 3086918, 1565000, N'Ulan Bator'),
(N'MO', N'MAC', N'Macao', N'MOP', N'AS', 449198, 254, N'Macao'),
(N'MP', N'MNP', N'Northern Mariana Islands', N'USD', N'OC', 53883, 477,
N'Saipan'),
(N'MQ', N'MTQ', N'Martinique', N'EUR', N'NA', 432900, 1100, N'Fort-de-
France'),
(N'MR', N'MRT', N'Mauritania', N'MRO', N'AF', 3205060, 1030700,
N'Nouakchott'),
(N'MS', N'MSR', N'Montserrat', N'XCD', N'NA', 9341, 102, N'Plymouth'),
(N'MT', N'MLT', N'Malta', N'EUR', N'EU', 403000, 316, N'Valletta'),
(N'MU', N'MUS', N'Mauritius', N'MUR', N'AF', 1294104, 2040, N'Port Louis'),
(N'MV', N'MDV', N'Maldives', N'MVR', N'AS', 395650, 300, N'Malé'),
(N'MW', N'MWI', N'Malawi', N'MWK', N'AF', 15447500, 118480, N'Lilongwe'),
(N'MX', N'MEX', N'Mexico', N'MXN', N'NA', 112468855, 1972550, N'Mexico
City'),
(N'MY', N'MYS', N'Malaysia', N'MYR', N'AS', 28274729, 329750, N'Kuala
Lumpur'),
(N'MZ', N'MOZ', N'Mozambique', N'MZN', N'AF', 22061451, 801590, N'Maputo'),
(N'NA', N'NAM', N'Namibia', N'NAD', N'AF', 2128471, 825418, N'Windhoek'),
(N'NC', N'NCL', N'New Caledonia', N'XPF', N'OC', 216494, 19060, N'Noumea'),
(N'NE', N'NER', N'Niger', N'XOF', N'AF', 15878271, 1267000, N'Niamey'),
(N'NF', N'NFK', N'Norfolk Island', N'AUD', N'OC', 1828, 34.6, N'Kingston'),
(N'NG', N'NGA', N'Nigeria', N'NGN', N'AF', 154000000, 923768, N'Abuja'),
(N'NI', N'NIC', N'Nicaragua', N'NIO', N'NA', 5995928, 129494, N'Managua'),
(N'NL', N'NLD', N'Netherlands', N'EUR', N'EU', 16645000, 41526,
N'Amsterdam'),
(N'NO', N'NOR', N'Norway', N'NOK', N'EU', 5009150, 324220, N'Oslo'),
(N'NP', N'NPL', N'Nepal', N'NPR', N'AS', 28951852, 140800, N'Kathmandu'),
(N'NR', N'NRU', N'Nauru', N'AUD', N'OC', 10065, 21, N''),
(N'NU', N'NIU', N'Niue', N'NZD', N'OC', 2166, 260, N'Alofi'),
(N'NZ', N'NZL', N'New Zealand', N'NZD', N'OC', 4252277, 268680,
N'Wellington'),
(N'OM', N'OMN', N'Oman', N'OMR', N'AS', 2967717, 212460, N'Muscat'),
(N'PA', N'PAN', N'Panama', N'PAB', N'NA', 3410676, 78200, N'Panama City'),
(N'PE', N'PER', N'Peru', N'PEN', N'SA', 29907003, 1285220, N'Lima'),

(N'PF', N'PYF', N'French Polynesia', N'XPF', N'OC', 270485, 4167, N'Papeete'),
(N'PG', N'PNG', N'Papua New Guinea', N'PGK', N'OC', 6064515, 462840, N'Port Moresby'),
(N'PH', N'PHL', N'Philippines', N'PHP', N'AS', 99900177, 300000, N'Manila'),
(N'PK', N'PAK', N'Pakistan', N'PKR', N'AS', 184404791, 803940, N'Islamabad'),
(N'PL', N'POL', N'Poland', N'PLN', N'EU', 38500000, 312685, N'Warsaw'),
(N'PM', N'SPM', N'Saint Pierre and Miquelon', N'EUR', N'NA', 7012, 242, N'Saint-Pierre'),
(N'PN', N'PCN', N'Pitcairn Islands', N'NZD', N'OC', 46, 47, N'Adamstown'),
(N'PR', N'PRI', N'Puerto Rico', N'USD', N'NA', 3916632, 9104, N'San Juan'),
(N'PS', N'PSE', N'Palestine', N'ILS', N'AS', 3800000, 5970, N''),
(N'PT', N'PRT', N'Portugal', N'EUR', N'EU', 10676000, 92391, N'Lisbon'),
(N'PW', N'PLW', N'Palau', N'USD', N'OC', 19907, 458, N'Melekeok - Palau State Capital'),
(N'PY', N'PRY', N'Paraguay', N'PYG', N'SA', 6375830, 406750, N'Asunción'),
(N'QA', N'QAT', N'Qatar', N'QAR', N'AS', 840926, 11437, N'Doha'),
(N'RE', N'REU', N'Reunion', N'EUR', N'AF', 776948, 2517, N'Saint-Denis'),
(N'RO', N'ROU', N'Romania', N'RON', N'EU', 21959278, 237500, N'Bucharest'),
(N'RS', N'SRB', N'Serbia', N'RSD', N'EU', 7344847, 88361, N'Belgrade'),
(N'RU', N'RUS', N'Russia', N'RUB', N'EU', 140702000, 17100000, N'Moscow'),
(N'RW', N'RWA', N'Rwanda', N'RWF', N'AF', 11055976, 26338, N'Kigali'),
(N'SA', N'SAU', N'Saudi Arabia', N'SAR', N'AS', 25731776, 1960582, N'Riyadh'),
(N'SB', N'SLB', N'Solomon Islands', N'SBD', N'OC', 559198, 28450, N'Honiara'),
(N'SC', N'SYC', N'Seychelles', N'SCR', N'AF', 88340, 455, N'Victoria'),
(N'SD', N'SDN', N'Sudan', N'SDG', N'AF', 35000000, 1861484, N'Khartoum'),
(N'SE', N'SWE', N'Sweden', N'SEK', N'EU', 9555893, 449964, N'Stockholm'),
(N'SG', N'SGP', N'Singapore', N'SGD', N'AS', 4701069, 692.7, N'Singapore'),
(N'SH', N'SHN', N'Saint Helena', N'SHP', N'AF', 7460, 410, N'Jamestown'),
(N'SI', N'SVN', N'Slovenia', N'EUR', N'EU', 2007000, 20273, N'Ljubljana'),
(N'SJ', N'SJM', N'Svalbard and Jan Mayen', N'NOK', N'EU', 2550, 62049, N'Longyearbyen'),
(N'SK', N'SVK', N'Slovakia', N'EUR', N'EU', 5455000, 48845, N'Bratislava'),
(N'SL', N'SLE', N'Sierra Leone', N'SLL', N'AF', 5245695, 71740, N'Freetown'),
(N'SM', N'SMR', N'San Marino', N'EUR', N'EU', 31477, 61.2, N'San Marino'),
(N'SN', N'SEN', N'Senegal', N'XOF', N'AF', 12323252, 196190, N'Dakar'),
(N'SO', N'SOM', N'Somalia', N'SOS', N'AF', 10112453, 637657, N'Mogadishu'),
(N'SR', N'SUR', N'Suriname', N'SRD', N'SA', 492829, 163270, N'Paramaribo'),
(N'SS', N'SSD', N'South Sudan', N'SSP', N'AF', 8260490, 644329, N'Juba'),
(N'ST', N'STP', N'São Tomé and Príncipe', N'STD', N'AF', 175808, 1001, N'São Tomé'),
(N'SV', N'SLV', N'El Salvador', N'USD', N'NA', 6052064, 21040, N'San Salvador'),
(N'SX', N'SXM', N'Sint Maarten', N'ANG', N'NA', 37429, 21, N'Philipsburg'),
(N'SY', N'SYR', N'Syria', N'SYP', N'AS', 22198110, 185180, N'Damascus'),
(N'SZ', N'SWZ', N'Swaziland', N'SZL', N'AF', 1354051, 17363, N'Mbabane'),
(N'TC', N'TCA', N'Turks and Caicos Islands', N'USD', N'NA', 20556, 430, N'Cockburn Town'),
(N'TD', N'TCD', N'Chad', N'XAF', N'AF', 10543464, 1284000, N'N'Djamena'),
(N'TF', N'ATF', N'French Southern Territories', N'EUR', N'AN', 140, 7829, N'Port-aux-Français'),
(N'TG', N'TGO', N'Togo', N'XOF', N'AF', 6587239, 56785, N'Lomé'),
(N'TH', N'THA', N'Thailand', N'THB', N'AS', 67089500, 514000, N'Bangkok'),
(N'TJ', N'TJK', N'Tajikistan', N'TJS', N'AS', 7487489, 143100, N'Dushanbe'),
(N'TK', N'TKL', N'Tokelau', N'NZD', N'OC', 1466, 10, N''),

(N'TL', N'TLS', N'East Timor', N'USD', N'OC', 1154625, 15007, N'Dili'),
 (N'TM', N'TKM', N'Turkmenistan', N'TMT', N'AS', 4940916, 488100,
 N'Ashgabat'),
 (N'TN', N'TUN', N'Tunisia', N'TND', N'AF', 10589025, 163610, N'Tunis'),
 (N'TO', N'TON', N'Tonga', N'TOP', N'OC', 122580, 748, N'Nuku'alofa'),
 (N'TR', N'TUR', N'Turkey', N'TRY', N'AS', 77804122, 780580, N'Ankara'),
 (N'TT', N'TTO', N'Trinidad and Tobago', N'TTD', N'NA', 1228691, 5128, N'Port
 of Spain'),
 (N'TV', N'TUV', N'Tuvalu', N'AUD', N'OC', 10472, 26, N'Funafuti'),
 (N'TW', N'TWN', N'Taiwan', N'TWD', N'AS', 22894384, 35980, N'Taipei'),
 (N'TZ', N'TZA', N'Tanzania', N'TZS', N'AF', 41892895, 945087, N'Dodoma'),
 (N'UA', N'UKR', N'Ukraine', N'UAH', N'EU', 45415596, 603700, N'Kyiv'),
 (N'UG', N'UGA', N'Uganda', N'UGX', N'AF', 33398682, 236040, N'Kampala'),
 (N'UM', N'UMI', N'U.S. Minor Outlying Islands', N'USD', N'OC', 0, 0, N''),
 (N'US', N'USA', N'United States', N'USD', N'NA', 310232863, 9629091,
 N'Washington'),
 (N'UY', N'URY', N'Uruguay', N'UYU', N'SA', 3477000, 176220, N'Montevideo'),
 (N'UZ', N'UZB', N'Uzbekistan', N'UZS', N'AS', 27865738, 447400, N'Tashkent'),
 (N'VA', N'VAT', N'Vatican City', N'EUR', N'EU', 921, 0.44, N'Vatican'),
 (N'VC', N'VCT', N'Saint Vincent and the Grenadines', N'XCD', N'NA', 104217,
 389, N'Kingstown'),
 (N'VE', N'VEN', N'Venezuela', N'VEF', N'SA', 27223228, 912050, N'Caracas'),
 (N'VG', N'VGB', N'British Virgin Islands', N'USD', N'NA', 21730, 153, N'Road
 Town'),
 (N'VI', N'VIR', N'U.S. Virgin Islands', N'USD', N'NA', 108708, 352,
 N'Charlotte Amalie'),
 (N'VN', N'VNM', N'Vietnam', N'VND', N'AS', 89571130, 329560, N'Hanoi'),
 (N'VU', N'VUT', N'Vanuatu', N'VUV', N'OC', 221552, 12200, N'Port Vila'),
 (N'WF', N'WLF', N'Wallis and Futuna', N'XPF', N'OC', 16025, 274, N'Mata-
 Utu'),
 (N'WS', N'WSM', N'Samoa', N'WST', N'OC', 192001, 2944, N'Apia'),
 (N'XK', N'XKX', N'Kosovo', N'EUR', N'EU', 1800000, 10908, N'Pristina'),
 (N'YE', N'YEM', N'Yemen', N'YER', N'AS', 23495361, 527970, N'Sanaa'),
 (N'YT', N'MYT', N'Mayotte', N'EUR', N'AF', 159042, 374, N'Mamoutzou'),
 (N'ZA', N'ZAF', N'South Africa', N'ZAR', N'AF', 49000000, 1219912,
 N'Pretoria'),
 (N'ZM', N'ZMB', N'Zambia', N'ZMW', N'AF', 13460305, 752614, N'Lusaka'),
 (N'ZW', N'ZWE', N'Zimbabwe', N'ZWD', N'AF', 11651858, 390580, N'Harare')

INSERT Currencies (CurrencyCode, Description) VALUES

(N'AED', N'United Arab Emirates Dirham'),
 (N'AFN', N'Afghanistan Afghani'),
 (N'ALL', N'Albania Lek'),
 (N'AMD', N'Armenia Dram'),
 (N'ANG', N'Netherlands Antilles Guilder'),
 (N'AOA', N'Angola Kwanza'),
 (N'ARS', N'Argentina Peso'),
 (N'AUD', N'Australia Dollar'),
 (N'AWG', N'Aruba Guilder'),
 (N'AZN', N'Azerbaijan New Manat'),
 (N'BAM', N'Bosnia and Herzegovina Convertible Marka'),
 (N'BBD', N'Barbados Dollar'),
 (N'BDT', N'Bangladesh Taka'),
 (N'BGN', N'Bulgaria Lev'),
 (N'BHD', N'Bahrain Dinar'),
 (N'BIF', N'Burundi Franc'),
 (N'BMD', N'Bermuda Dollar'),

(N'BND', N'Brunei Darussalam Dollar'),
(N'BOB', N'Bolivia Boliviano'),
(N'BRL', N'Brazil Real'),
(N'BSD', N'Bahamas Dollar'),
(N'BTN', N'Bhutan Ngultrum'),
(N'BWP', N'Botswana Pula'),
(N'BYR', N'Belarus Ruble'),
(N'BZD', N'Belize Dollar'),
(N'CAD', N'Canada Dollar'),
(N'CDF', N'Congo/Kinshasa Franc'),
(N'CHF', N'Switzerland Franc'),
(N'CLP', N'Chile Peso'),
(N'CNY', N'China Yuan Renminbi'),
(N'COP', N'Colombia Peso'),
(N'CRC', N'Costa Rica Colon'),
(N'CUC', N'Cuba Convertible Peso'),
(N'CUP', N'Cuba Peso'),
(N'CVE', N'Cape Verde Escudo'),
(N'CZK', N'Czech Republic Koruna'),
(N'DJF', N'Djibouti Franc'),
(N'DKK', N'Denmark Krone'),
(N'DOP', N'Dominican Republic Peso'),
(N'DZD', N'Algeria Dinar'),
(N'EGP', N'Egypt Pound'),
(N'ERN', N'Eritrea Nakfa'),
(N'ETB', N'Ethiopia Birr'),
(N'EUR', N'Euro Member Countries'),
(N'FJD', N'Fiji Dollar'),
(N'FKP', N'Falkland Islands (Malvinas) Pound'),
(N'GBP', N'United Kingdom Pound'),
(N'GEL', N'Georgia Lari'),
(N'GGP', N'Guernsey Pound'),
(N'GHS', N'Ghana Cedi'),
(N'GIP', N'Gibraltar Pound'),
(N'GMD', N'Gambia Dalasi'),
(N'GNF', N'Guinea Franc'),
(N'GTQ', N'Guatemala Quetzal'),
(N'GYD', N'Guyana Dollar'),
(N'HKD', N'Hong Kong Dollar'),
(N'HNL', N'Honduras Lempira'),
(N'HRK', N'Croatia Kuna'),
(N'HTG', N'Haiti Gourde'),
(N'HUF', N'Hungary Forint'),
(N>IDR', N'Indonesia Rupiah'),
(N'ILS', N'Israel Shekel'),
(N'IMP', N'Isle of Man Pound'),
(N'INR', N'India Rupee'),
(N'IQD', N'Iraq Dinar'),
(N'IRR', N'Iran Rial'),
(N'ISK', N'Iceland Krona'),
(N'JEP', N'Jersey Pound'),
(N'JMD', N'Jamaica Dollar'),
(N'JOD', N'Jordan Dinar'),
(N'JPY', N'Japan Yen'),
(N'KES', N'Kenya Shilling'),
(N'KGS', N'Kyrgyzstan Som'),
(N'KHR', N'Cambodia Riel'),

(N'KMF', N'Comoros Franc'),
(N'KPW', N'Korea (North) Won'),
(N'KRW', N'Korea (South) Won'),
(N'KWD', N'Kuwait Dinar'),
(N'KYD', N'Cayman Islands Dollar'),
(N'KZT', N'Kazakhstan Tenge'),
(N'LAK', N'Laos Kip'),
(N'LBP', N'Lebanon Pound'),
(N'LKR', N'Sri Lanka Rupee'),
(N'LRD', N'Liberia Dollar'),
(N'LSL', N'Lesotho Loti'),
(N'LYD', N'Libya Dinar'),
(N'MAD', N'Morocco Dirham'),
(N'MDL', N'Moldova Leu'),
(N'MGA', N'Madagascar Ariary'),
(N'MKD', N'Macedonia Denar'),
(N'MMK', N'Myanmar (Burma) Kyat'),
(N'MNT', N'Mongolia Tughrik'),
(N'MOP', N'Macau Pataca'),
(N'MRO', N'Mauritania Ouguiya'),
(N'MUR', N'Mauritius Rupee'),
(N'MVR', N'Maldives (Maldiv Islands) Rufiyaa'),
(N'MWK', N'Malawi Kwacha'),
(N'MXN', N'Mexico Peso'),
(N'MYR', N'Malaysia Ringgit'),
(N'MZN', N'Mozambique Metical'),
(N'NAD', N'Namibia Dollar'),
(N'NGN', N'Nigeria Naira'),
(N'NIO', N'Nicaragua Cordoba'),
(N'NOK', N'Norway Krone'),
(N'NPR', N'Nepal Rupee'),
(N'NZD', N'New Zealand Dollar'),
(N'OMR', N'Oman Rial'),
(N'PAB', N'Panama Balboa'),
(N'PEN', N'Peru Nuevo Sol'),
(N'PGK', N'Papua New Guinea Kina'),
(N'PHP', N'Philippines Peso'),
(N'PKR', N'Pakistan Rupee'),
(N'PLN', N'Poland Zloty'),
(N'PYG', N'Paraguay Guarani'),
(N'QAR', N'Qatar Riyal'),
(N'RON', N'Romania New Leu'),
(N'RSD', N'Serbia Dinar'),
(N'RUB', N'Russia Ruble'),
(N'RWF', N'Rwanda Franc'),
(N'SAR', N'Saudi Arabia Riyal'),
(N'SBD', N'Solomon Islands Dollar'),
(N'SCR', N'Seychelles Rupee'),
(N'SDG', N'Sudan Pound'),
(N'SEK', N'Sweden Krona'),
(N'SGD', N'Singapore Dollar'),
(N'SHP', N'Saint Helena Pound'),
(N'SLL', N'Sierra Leone Leone'),
(N'SOS', N'Somalia Shilling'),
(N'SPL', N'Seborga Luigino'),
(N'SRD', N'Suriname Dollar'),
(N'SSP', N'South Sudanese Pound'),

(N'STD', N'São Tomé and Príncipe Dobra'),
 (N'SVC', N'El Salvador Colon'),
 (N'SYP', N'Syria Pound'),
 (N'SZL', N'Swaziland Lilangeni'),
 (N'THB', N'Thailand Baht'),
 (N'TJS', N'Tajikistan Somoni'),
 (N'TMT', N'Turkmenistan Manat'),
 (N'TND', N'Tunisia Dinar'),
 (N'TOP', N'Tonga Pa'anga'),
 (N'TRY', N'Turkey Lira'),
 (N'TTD', N'Trinidad and Tobago Dollar'),
 (N'TVD', N'Tuvalu Dollar'),
 (N'TWD', N'Taiwan New Dollar'),
 (N'TZS', N'Tanzania Shilling'),
 (N'UAH', N'Ukraine Hryvnia'),
 (N'UGX', N'Uganda Shilling'),
 (N'USD', N'United States Dollar'),
 (N'UYU', N'Uruguay Peso'),
 (N'UZS', N'Uzbekistan Som'),
 (N'VEF', N'Venezuela Bolivar'),
 (N'VND', N'Viet Nam Dong'),
 (N'VUV', N'Vanuatu Vatu'),
 (N'WST', N'Samoa Tala'),
 (N'XAF', N'Communauté Financière Africaine (BEAC) CFA Franc BEAC'),
 (N'XCD', N'East Caribbean Dollar'),
 (N'XDR', N'International Monetary Fund (IMF) Special Drawing Rights'),
 (N'XOF', N'Communauté Financière Africaine (BCEAO) Franc'),
 (N'XPF', N'Comptoirs Français du Pacifique (CFP) Franc'),
 (N'YER', N'Yemen Rial'),
 (N'ZAR', N'South Africa Rand'),
 (N'ZMW', N'Zambia Kwacha'),
 (N'ZWD', N'Zimbabwe Dollar')

SET IDENTITY_INSERT Rivers ON

INSERT Rivers (Id, RiverName, Length, DrainageArea, AverageDischarge, Outflow) VALUES

(1, N'Nile', 6650, 3254555, 5100, N'Mediterranean'),
 (2, N'Amazon', 6400, 7050000, 219000, N'Atlantic Ocean'),
 (3, N'Yangtze', 6300, 1800000, 31900, N'East China Sea'),
 (4, N'Mississippi', 6275, 2980000, 16200, N'Gulf of Mexico'),
 (5, N'Yenisei', 5539, 2580000, 19600, N'Kara Sea'),
 (6, N'Yellow River', 5464, 745000, 2110, N'Bohai Sea'),
 (7, N'Ob', 5410, 2990000, 12800, N'Gulf of Ob'),
 (8, N'Paraná', 4880, 2582672, 18000, N'Río de la Plata'),
 (9, N'Congo', 4700, 3680000, 41800, N'Atlantic Ocean'),
 (10, N'Amur', 4444, 1855000, 11400, N'Sea of Okhotsk'),
 (11, N'Lena', 4400, 2490000, 17100, N'Laptev Sea'),
 (12, N'Mekong', 4350, 810000, 16000, N'South China Sea'),
 (13, N'Mackenzie', 4241, 1790000, 10300, N'Beaufort Sea'),
 (14, N'Niger', 4200, 2090000, 9570, N'Gulf of Guinea'),
 (15, N'Murray', 3672, 1061000, 7670, N'Southern Ocean'),
 (16, N'Tocantins', 3650, 950000, 13598, N'Atlantic Ocean, Amazon'),
 (17, N'Volga', 3645, 1380000, 8080, N'Caspian Sea'),
 (18, N'Shatt al-Arab', 3596, 884000, 8560, N'Persian Gulf'),
 (19, N'Madeira', 3380, 1485200, 31200, N'Amazon'),
 (20, N'Purús', 3211, 63166, 8400, N'Amazon'),
 (21, N'Yukon', 3185, 850000, 6210, N'Bering Sea'),

```

(22, N'Indus', 3180, 960000, 7160, N'Arabian Sea'),
(23, N'São Francisco', 3180, 610000, 3300, N'Atlantic Ocean'),
(24, N'Syr Darya', 3078, 219000, 7030, N'Aral Sea'),
(25, N'Salween', 3060, 324000, 3153, N'Andaman Sea'),
(26, N'Saint Lawrence', 3058, 1030000, 10100, N'Gulf of Saint Lawrence'),
(27, N'Rio Grande', 3057, 570000, 820, N'Gulf of Mexico'),
(28, N'Lower Tunguska', 2989, 473000, 3600, N'Yenisei'),
(29, N'Brahmaputra', 2948, 1730000, 19200, N'Ganges'),
(30, N'Danube', 2888, 817000, 7130, N'Black Sea')
SET IDENTITY_INSERT Rivers OFF

```

```

INSERT CountriesRivers (RiverId, CountryCode) VALUES

```

```

(9, N'AO'),
(8, N'AR'),
(30, N'AT'),
(15, N'AU'),
(29, N'BD'),
(14, N'BF'),
(30, N'BG'),
(1, N'BI'),
(9, N'BI'),
(14, N'BJ'),
(2, N'BO'),
(8, N'BO'),
(19, N'BO'),
(2, N'BR'),
(8, N'BR'),
(16, N'BR'),
(19, N'BR'),
(20, N'BR'),
(23, N'BR'),
(29, N'BT'),
(4, N'CA'),
(13, N'CA'),
(21, N'CA'),
(26, N'CA'),
(1, N'CD'),
(9, N'CD'),
(9, N'CF'),
(9, N'CG'),
(9, N'CM'),
(14, N'CM'),
(3, N'CN'),
(6, N'CN'),
(7, N'CN'),
(10, N'CN'),
(12, N'CN'),
(22, N'CN'),
(25, N'CN'),
(29, N'CN'),
(2, N'CO'),
(30, N'DE'),
(14, N'DZ'),
(2, N'EC'),
(1, N'EG'),
(1, N'ER'),
(1, N'ET'),

```

(14, N'GN'),
(2, N'GY'),
(30, N'HR'),
(30, N'HU'),
(22, N'IN'),
(29, N'IN'),
(18, N'IQ'),
(1, N'KE'),
(24, N'KG'),
(12, N'KH'),
(7, N'KZ'),
(24, N'KZ'),
(12, N'LA'),
(14, N'ML'),
(12, N'MM'),
(25, N'MM'),
(5, N'MN'),
(7, N'MN'),
(10, N'MN'),
(27, N'MX'),
(14, N'NE'),
(14, N'NG'),
(29, N'NP'),
(2, N'PE'),
(19, N'PE'),
(20, N'PE'),
(22, N'PK'),
(8, N'PY'),
(30, N'RO'),
(30, N'RS'),
(5, N'RU'),
(7, N'RU'),
(10, N'RU'),
(11, N'RU'),
(17, N'RU'),
(28, N'RU'),
(1, N'RW'),
(9, N'RW'),
(1, N'SD'),
(30, N'SK'),
(1, N'SS'),
(18, N'SY'),
(14, N'TD'),
(12, N'TH'),
(25, N'TH'),
(24, N'TJ'),
(18, N'TR'),
(1, N'TZ'),
(9, N'TZ'),
(1, N'UG'),
(4, N'US'),
(21, N'US'),
(26, N'US'),
(27, N'US'),
(8, N'UY'),
(24, N'UZ'),
(2, N'VE'),

```
(12, N'VN'),
(9, N'ZM')
```

```
INSERT INTO Monasteries(Name, CountryCode) VALUES
('Troyan Monastery "Holy Mother"'s Assumption"', 'BG'),
('Rila Monastery "St. Ivan of Rila"', 'BG'),
('Bachkovo Monastery "Virgin Mary"', 'BG'),
('Thrangu Tashi Yangtse Monastery', 'NP'),
('Kopan Monastery', 'NP'),
('Shechen Tennyi Dargyeling Monastery', 'NP'),
('Benchen Monastery', 'NP'),
('Wa Sau Toi', 'CN'),
('Rakya Monastery', 'CN'),
('Southern Shaolin Monastery', 'CN'),
('Dabei Monastery', 'CN'),
('Lhunshigyia Monastery', 'CN'),
('The Holy Monastery of Stavronikita', 'GR'),
('Monasteries of Meteora', 'GR'),
('Taung Kalat Monastery', 'MM'),
('Pa-Auk Forest Monastery', 'MM'),
('Taktsang Palphug Monastery', 'BT'),
('Sümela Monastery', 'TR')
```

```
GO
```

```
-- Add integrity constraints
```

```
ALTER TABLE Countries WITH CHECK ADD CONSTRAINT FK_Countries_Continents
FOREIGN KEY(ContinentCode) REFERENCES Continents (ContinentCode)
GO
```

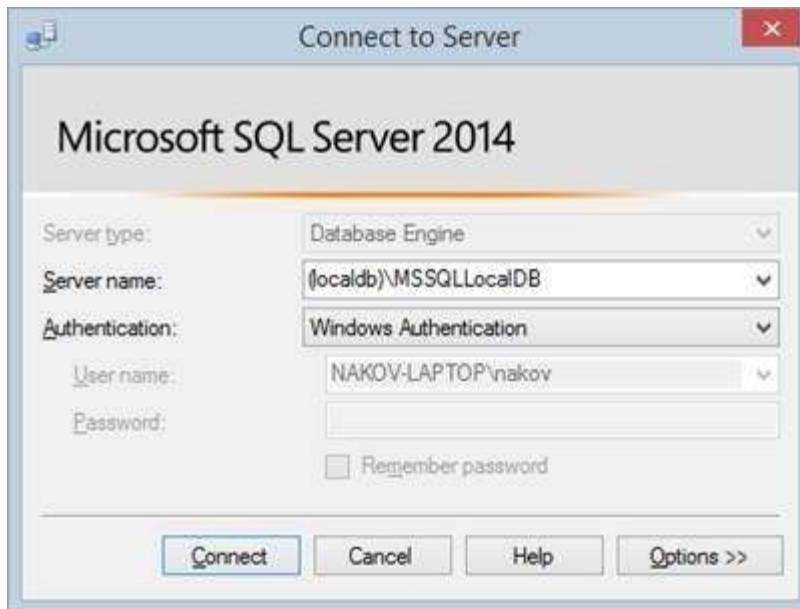
```
ALTER TABLE Countries WITH CHECK ADD CONSTRAINT FK_Countries_Currencies
FOREIGN KEY(CurrencyCode) REFERENCES Currencies (CurrencyCode)
GO
```

```
ALTER TABLE CountriesRivers WITH CHECK ADD CONSTRAINT
FK_CountriesRivers_Countries
FOREIGN KEY(CountryCode) REFERENCES Countries (CountryCode)
GO
```

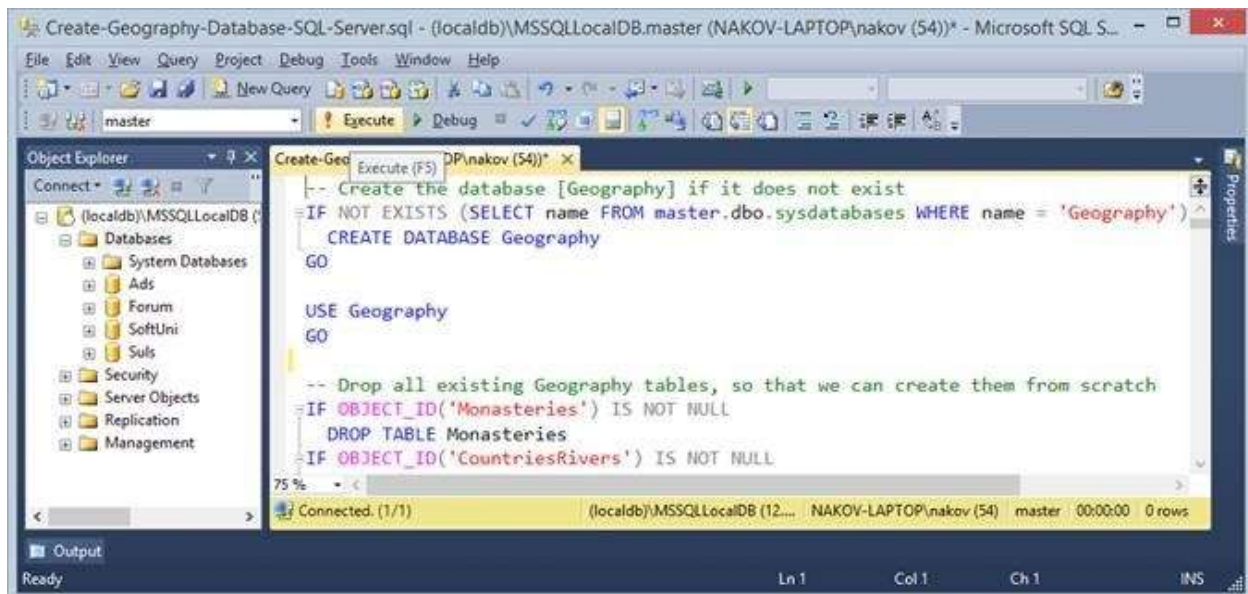
```
ALTER TABLE CountriesRivers WITH CHECK ADD CONSTRAINT
FK_CountriesRivers_Rivers
FOREIGN KEY(RiverId) REFERENCES Rivers (Id)
GO
```

```
ALTER TABLE Monasteries WITH CHECK ADD CONSTRAINT FK_Monasteries_Countries
FOREIGN KEY (CountryCode) REFERENCES Countries(CountryCode)
GO
```

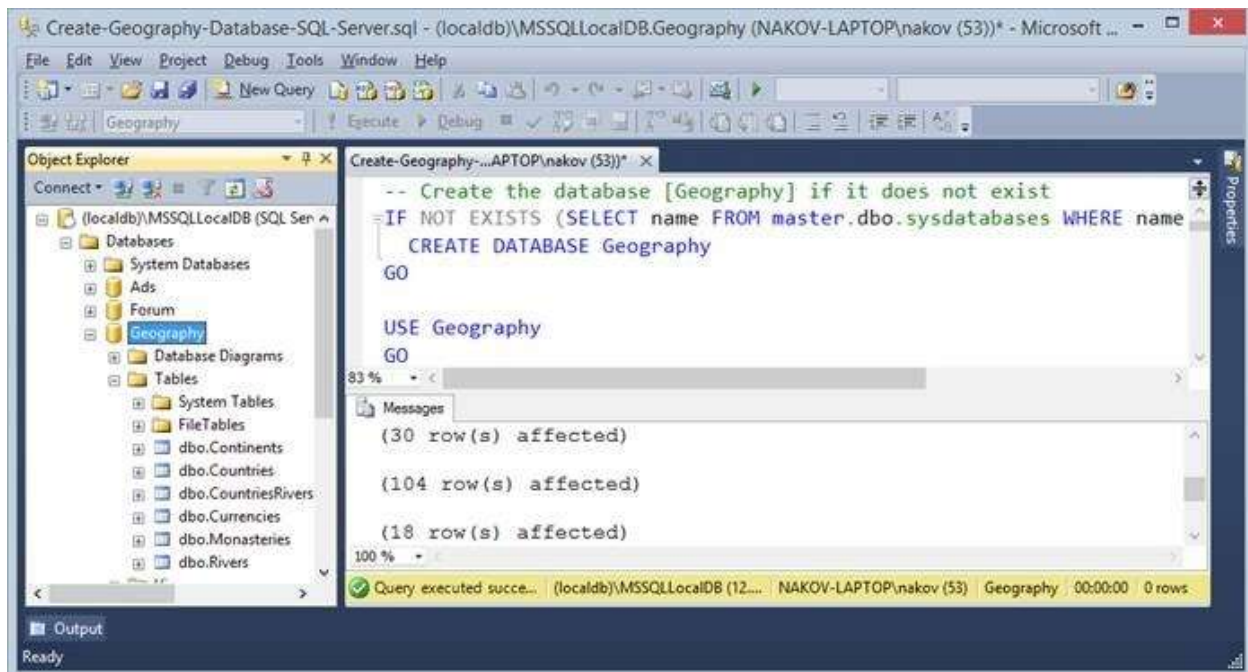
Step 1. Connect to MS SQL Server using SQL Server Management Studio. You may use MS SQL Express Edition, MS SQL Developer Edition of MS SQL LocalDB (version 2012, 2014 or later). Example (connecting to SQL 2014 LocalDB):



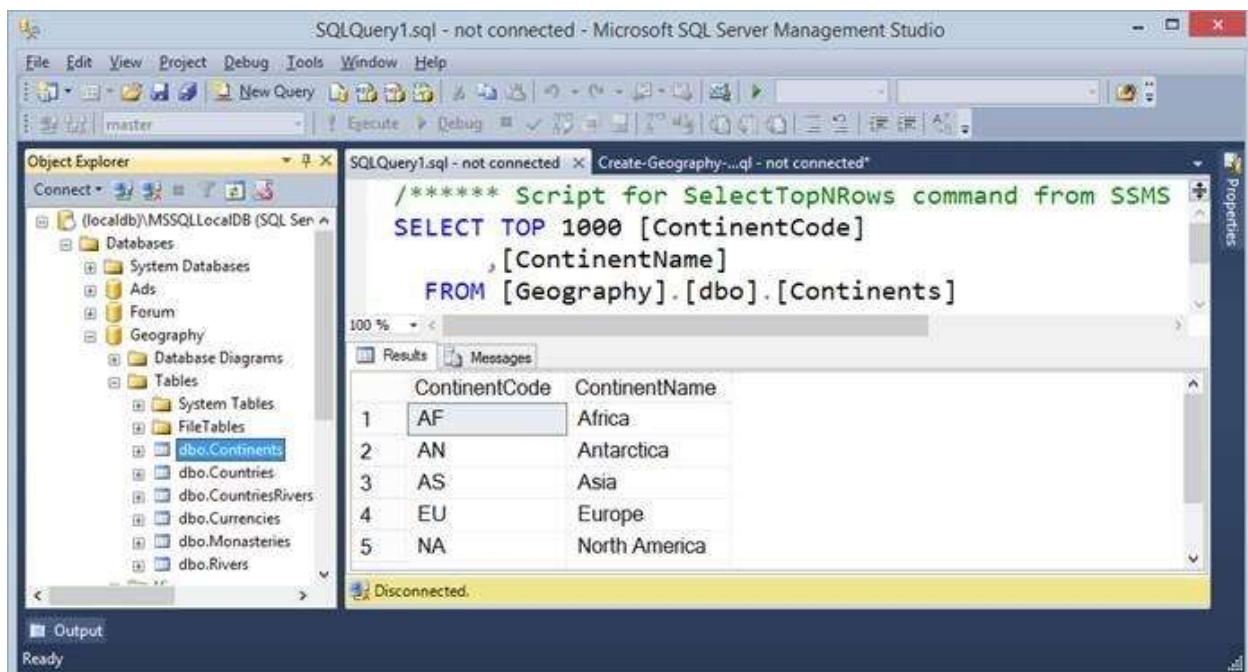
Step 2. Run the above provided SQL script to create the database schema and load sample data in the tables (copy and paste the above sql script into the SQL Server Management Studio and execute it):



The SQL script should **execute without errors**:



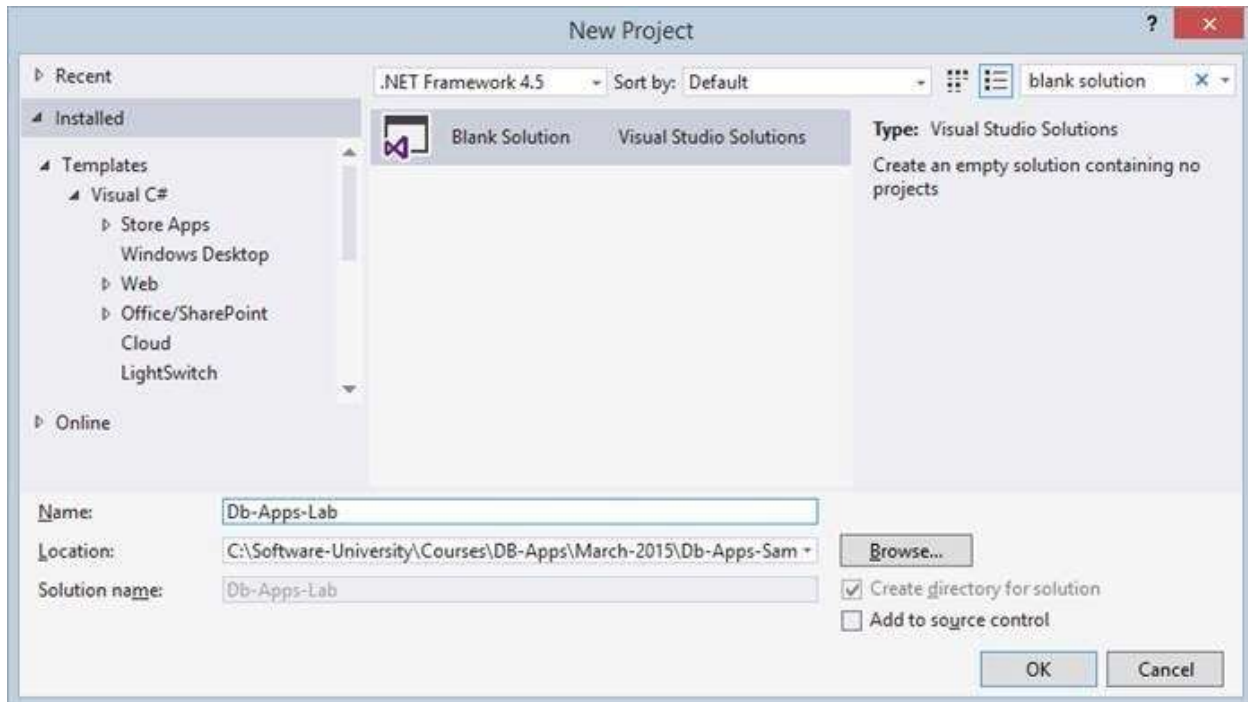
Step 3. Refresh the Databases. You should see the database “**Geography**”. It should hold several tables (**Continents**, **Countries**, **Currencies**, **Monasteries**, **Rivers** and **CountriesRivers**). Ensure you have data in all tables:



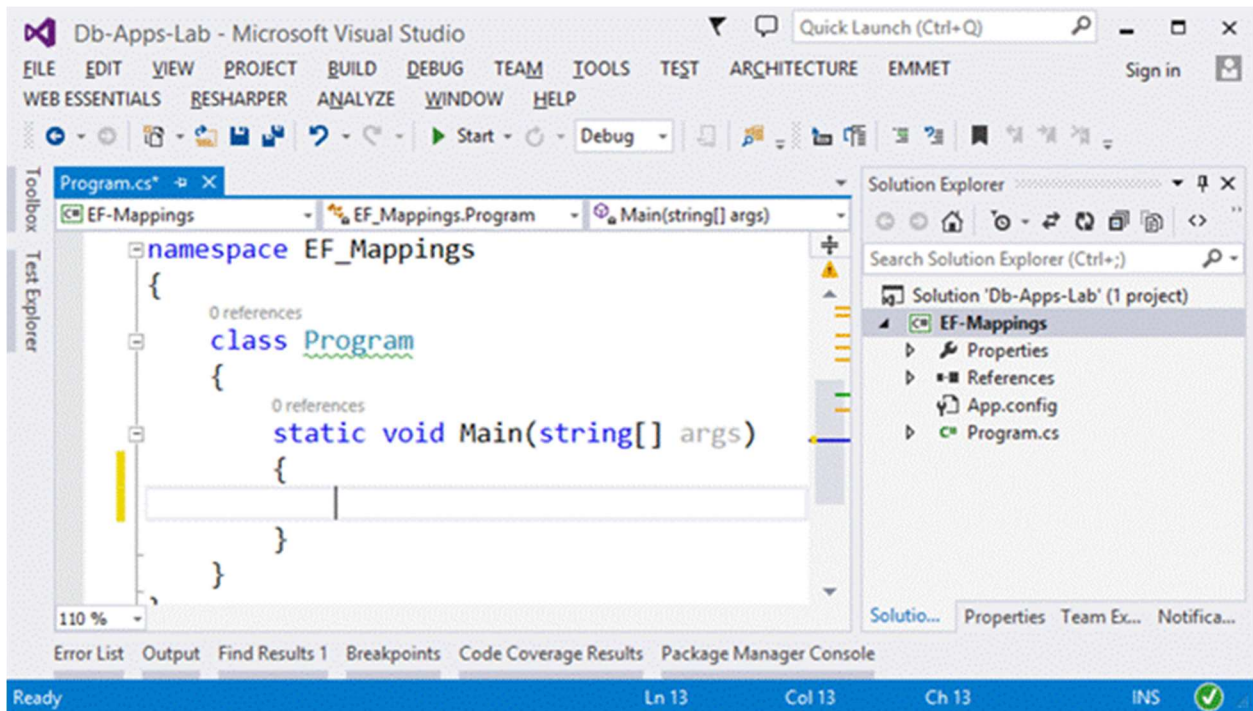
Problem 1. Entity Framework Mappings (Database First)

Create an **Entity Framework (EF) data model** of the existing database “Geography” (map the database tables to C# classes). Use the “**database first**” model in EF. To test your EF data model, **list all continent names**.

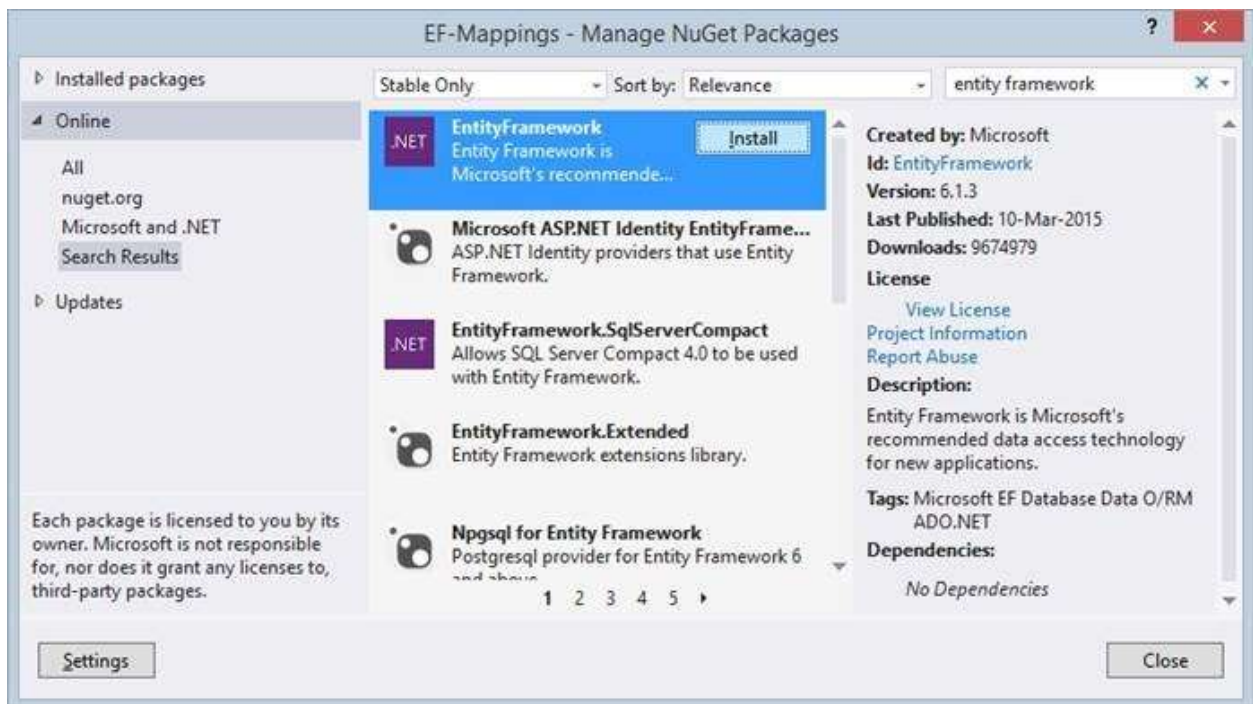
Step 1. Start Visual Studio. Use Visual Studio 2017 or later. Create a **new Blank Solution** called “**Db-Apps-Lab**”:



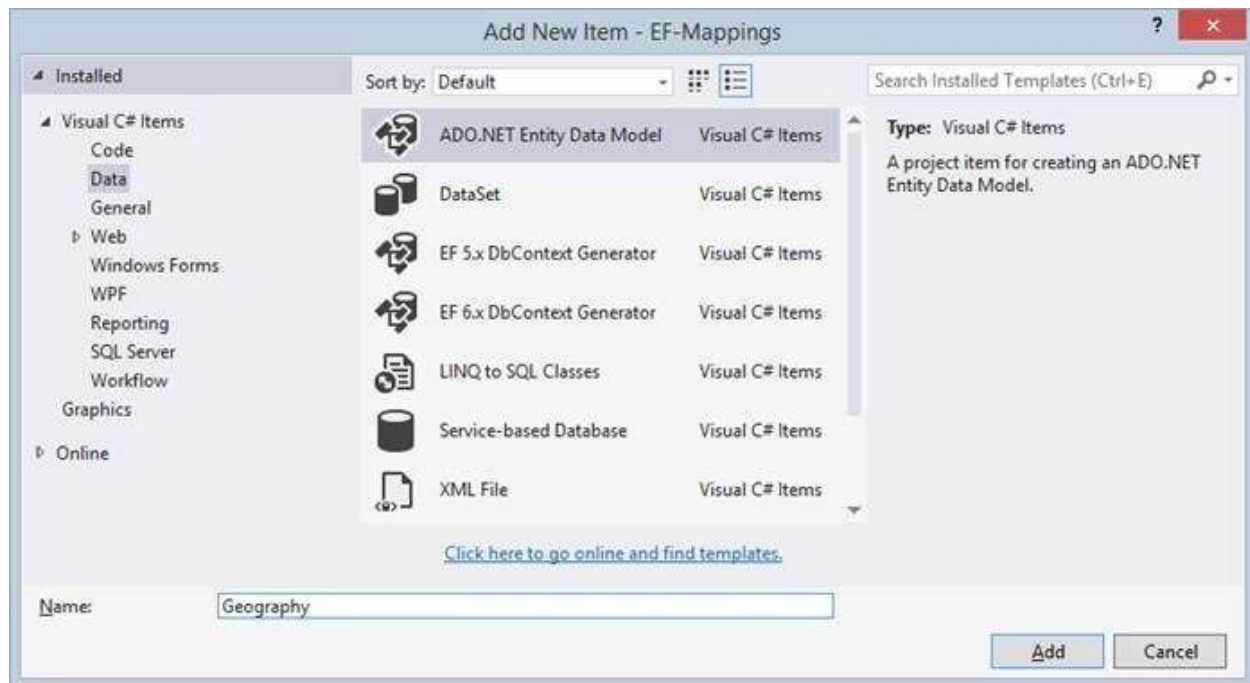
Step 2. Create a new **C# Console Application** in your blank VS solution called “**EF-Mappings**”:



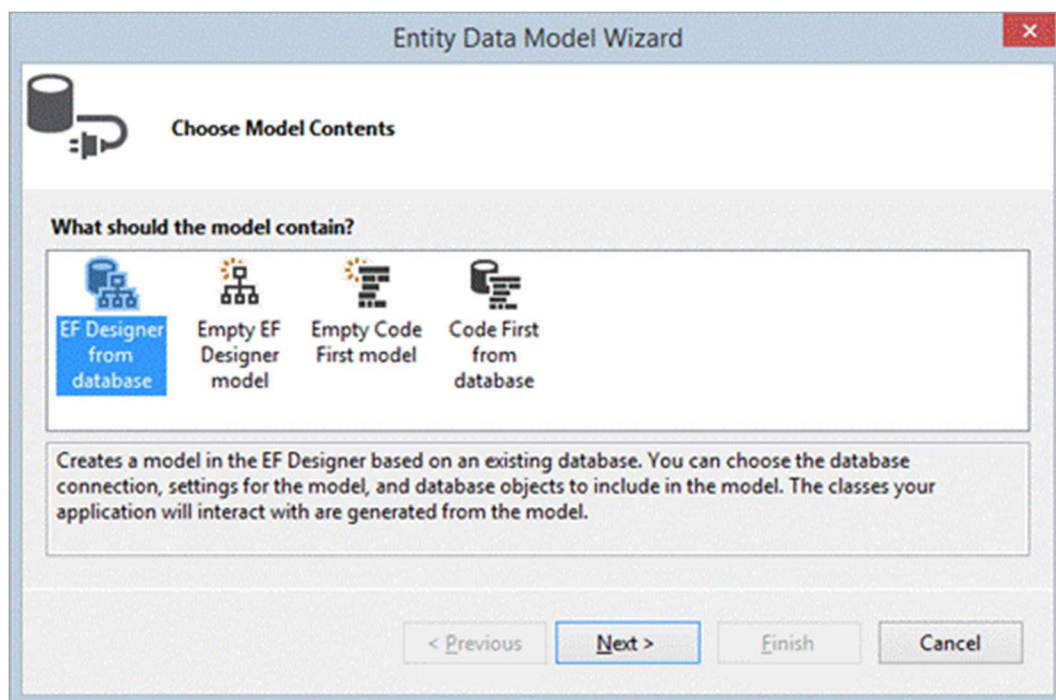
Step 3. Add a reference to “Entity Framework” through the NuGet Package Manger in Visual Studio:



Step 4. Add a new ADO.NET Entity Data Model named “Geography”:




Step 5. Create a database first EF mappings (choose the **EF Designer from existing database**):



Step 6. Create a database connection to your “**Geography**” database in MS SQL Server:

Entity Data Model Wizard

 Choose Your Data Connection

Which data connection should your application use to connect to the database?

nakov-laptop\localdb#40996000.Geography.dbo ▼ New Connection...

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

☐ No, exclude sensitive data from the connection string. I will set it in my application code.

☐ Yes, include the sensitive data in the connection string.

Connection string:

```
metadata=res://*/Geography.csdl|res://*/Geography.ssdl|
res://*/Geography.msl;provider=System.Data.SqlClient;provider connection string="data source=
(localdb)\MSSQLLocalDB;initial catalog=Geography;integrated
security=True;MultipleActiveResultSets=True;App=EntityFramework"
```

☒ Save connection settings in App.Config as:

GeographyEntities

< Previous Next > Finish Cancel

Step 7. Select all tables to be mapped in the EF data model:

Entity Data Model Wizard

Choose Your Database Objects and Settings

Which database objects do you want to include in your model?

- ☒ Tables
 - ☒ dbo
 - ☒ Continents
 - ☒ Countries
 - ☒ CountriesRivers
 - ☒ Currencies
 - ☒ Monasteries
 - ☒ Rivers
 - ☐ Views
 - ☐ Stored Procedures and Functions

☒ Pluralize or singularize generated object names

☒ Include foreign key columns in the model

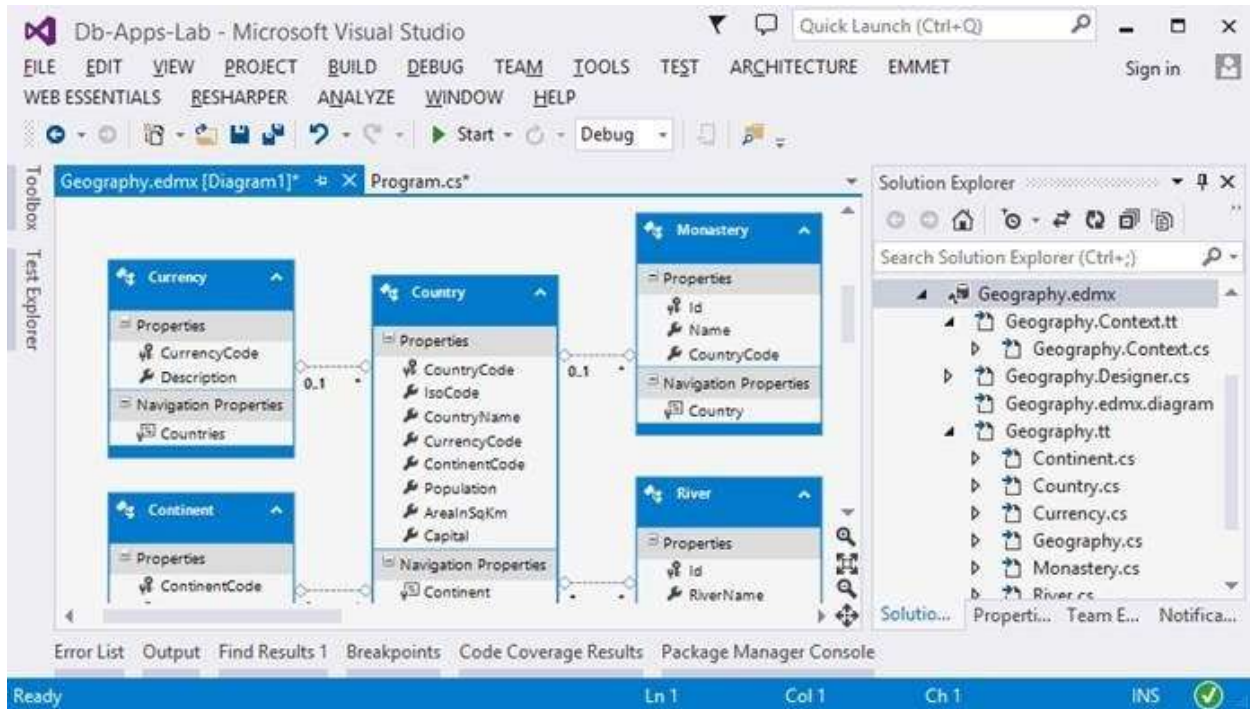
☐ Import selected stored procedures and functions into the entity model

Model Namespace:

GeographyModel

< Previous Next > Finish Cancel

Step 8. Visual Studio will create for you a EF database first data model (**EDMX** file):



Step 9. To list all continent names, write some code:

The screenshot shows the Microsoft Visual Studio interface with the 'ListContinents.cs*' file open. The code is as follows:

```
static void Main()
{
    var context = new GeographyEntities();
    foreach (var continent in context.Continents)
    {
        Console.WriteLine(continent.ContinentName);
    }
}
```

The Solution Explorer on the right shows the project structure, including 'EF-Mappings', 'Properties', 'References', 'App.config', 'Geography.edmx', 'ListContinents.cs', and 'packages.config'. The status bar at the bottom indicates 'Ready' and 'Ln 16 Col 60 Ch 60 INS'.

Step 10. Run your program. It should list the continent names from the database:



Step 11. Rename the class “Program” to “ListContinents“. Also, rename the file name. Rename your project from “EF-Mappings” to “1. EF-Mappings” (this is the solution of the first lab problem and you will have more problems: 2, 3, 4, ...).

Problem 2. Export Rivers as JSON

Write a **C# application** based on your EF data model for **exporting all rivers along with their countries** in the following JSON format:

rivers.json

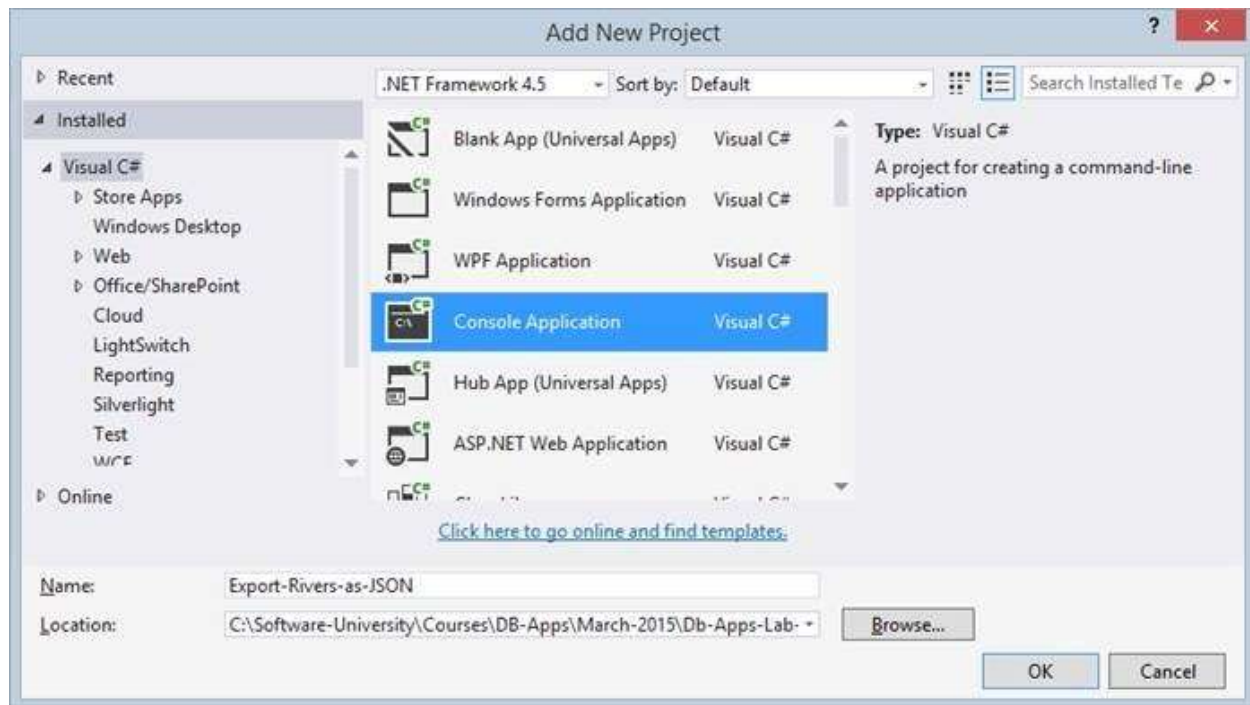
```
1[
2 { "riverName": "Nile", "riverLength": 6650, "countries": ["Burundi","Democratic Republi
3Sudan","Sudan","Tanzania","Uganda"] },
4 { "riverName": "Amazon", "riverLength": 6400, "countries": ["Bolivia","Brazil","Colombi
5 { "riverName": "Yangtze", "riverLength": 6300, "countries":["China"] },
6 ...
7]
```

Write the output in a JSON file named rivers.json. Include in the output the rivers with no countries (if any). The JSON file code formatting is not important.

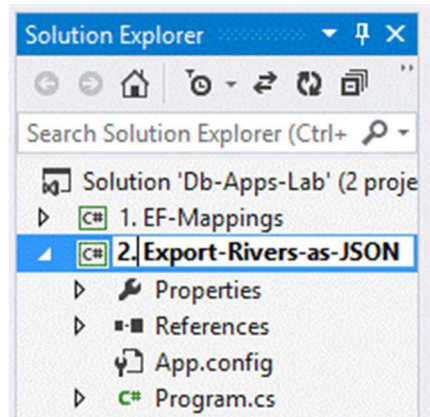
Order the **rivers** by **length** (from the longest) and the countries for each river **alphabetically**.

For better performance, ensure your program executes a **single DB query** and retrieves from the database only the required data (without any unneeded rows and columns).

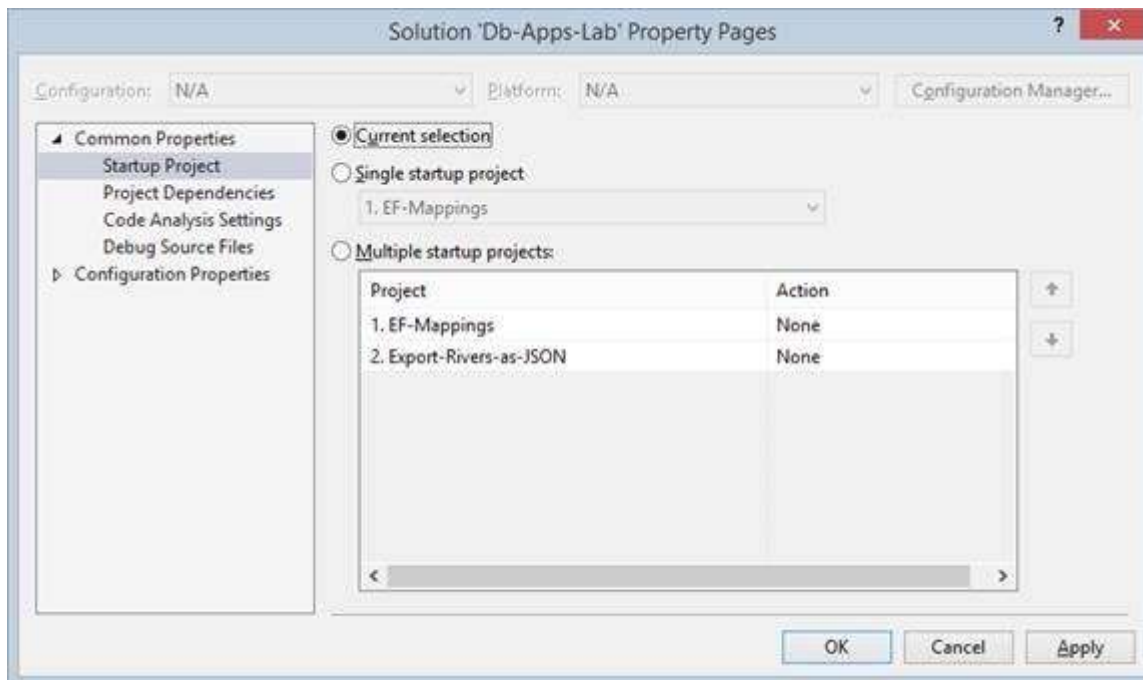
Step 1. Create a new **Console Application** called “Export-Rivers-as-JSON“ in your VS solution:



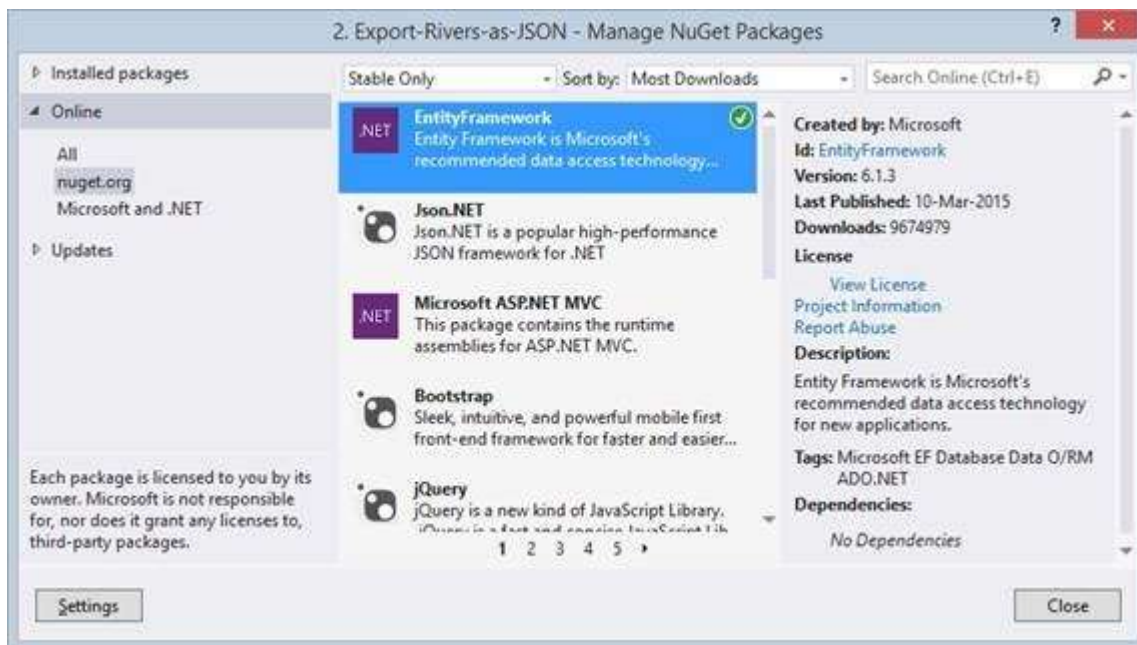
Step 2. Rename the project from “**Export-Rivers-as-JSON**” to “**2. Export-Rivers-as-JSON**”:



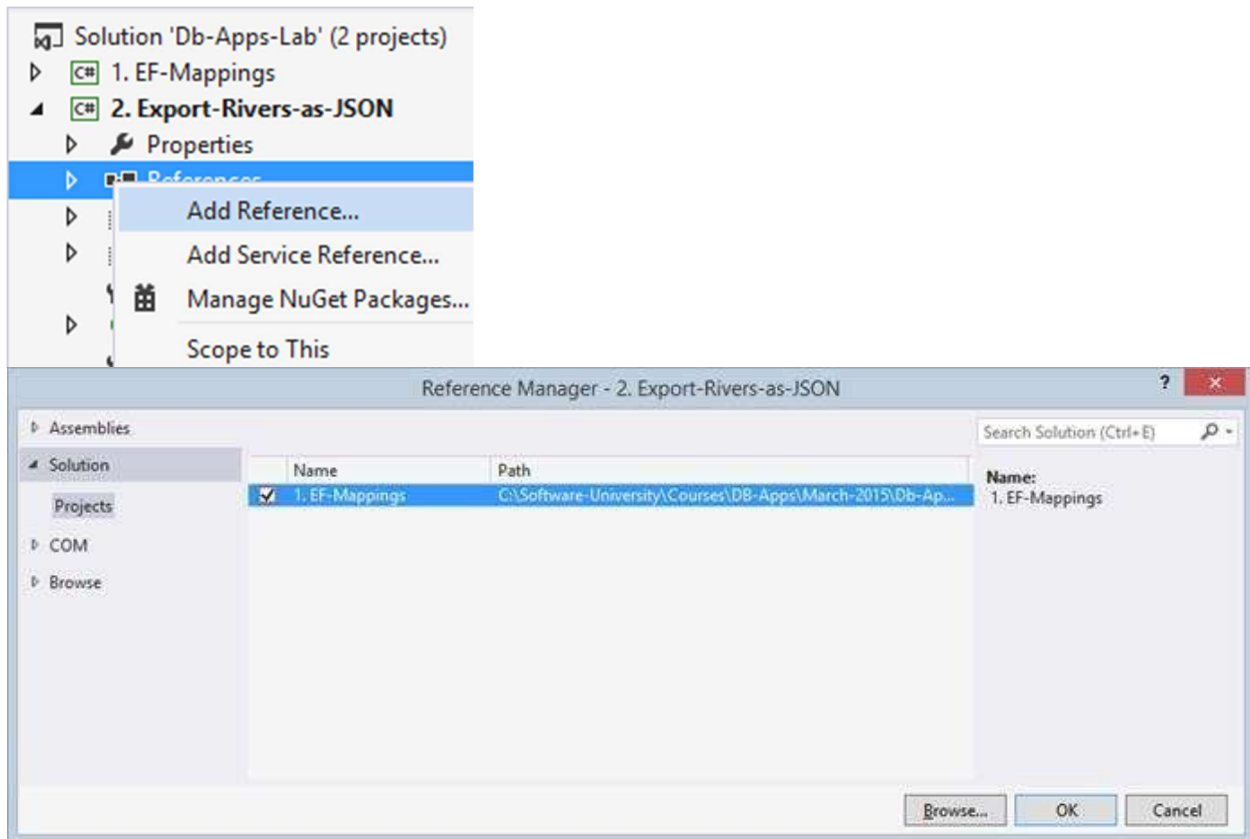
Step 3. Set the current VS project as startup project:



Step 4. Add a **NuGet** reference to the package “**Entity Framework**”:

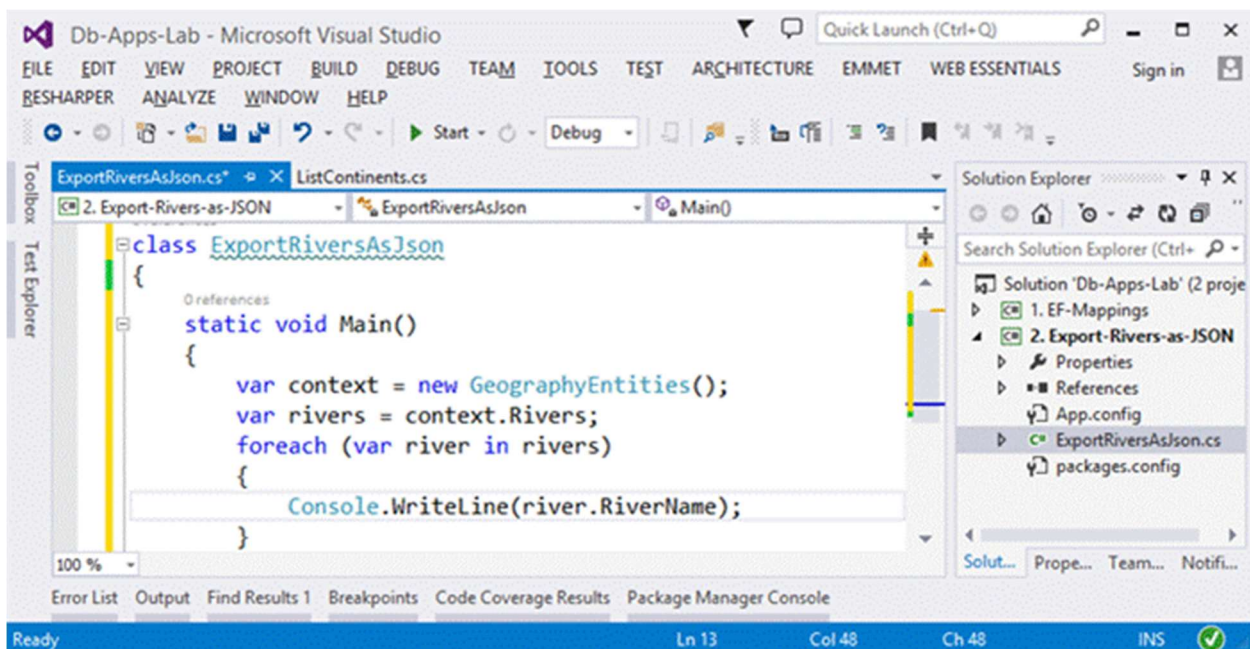


Step 5. To reuse the EF data model from the previous exercise, **add a reference** to the project holding the previous problem “**1. EF-Mappings**”:

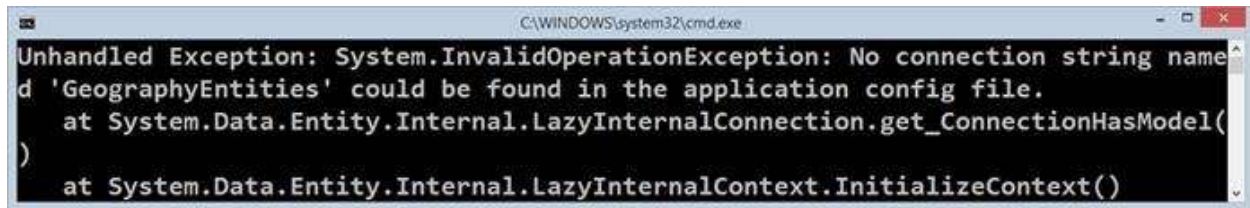


Step 6. Rename the class “**Program**” to “**ExportRiversAsJson**”.

Step 7. Write some C# code to **list all rivers**:

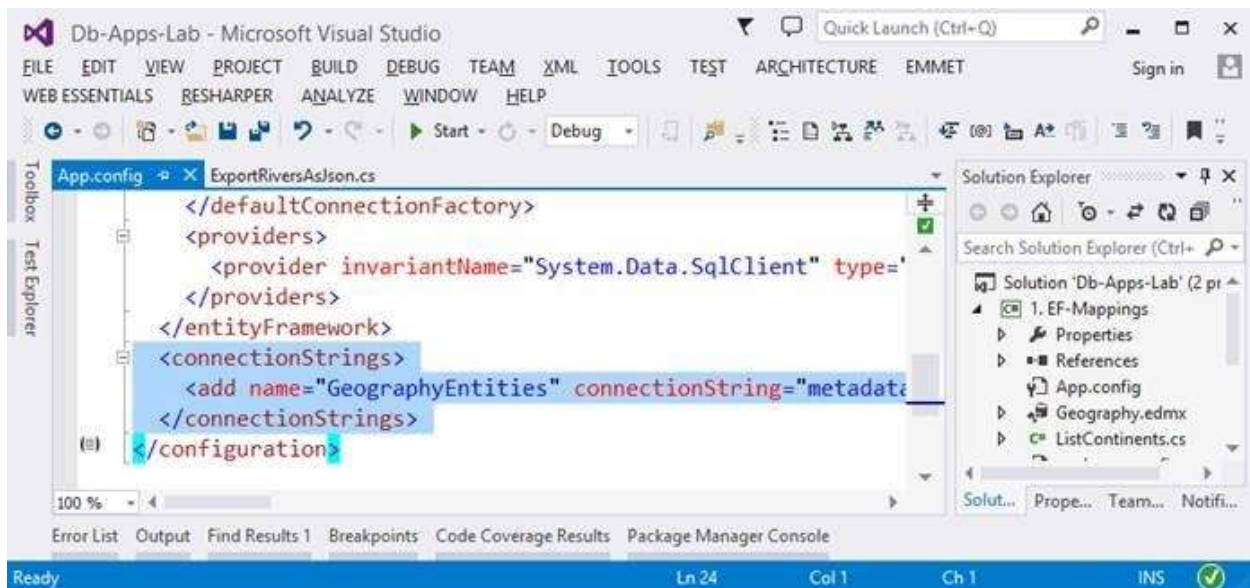


Step 8. Run the program to see the “No connection string” exception:

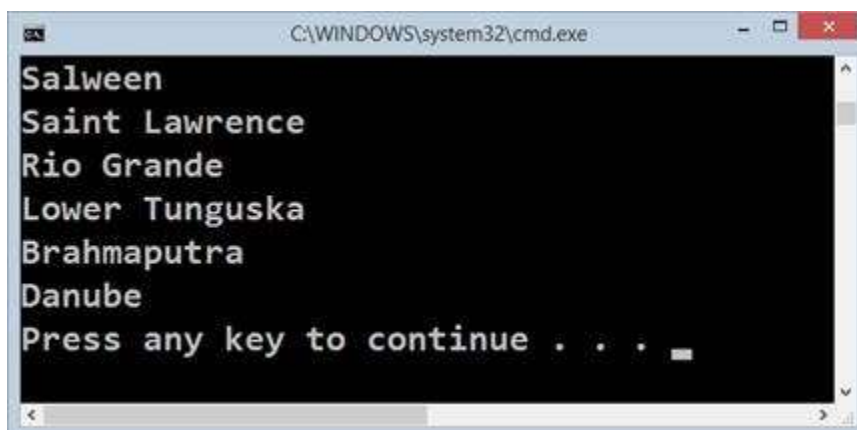


```
C:\WINDOWS\system32\cmd.exe
Unhandled Exception: System.InvalidOperationException: No connection string name
d 'GeographyEntities' could be found in the application config file.
   at System.Data.Entity.Internal.LazyInternalConnection.get_ConnectionHasModel(
)
   at System.Data.Entity.Internal.LazyInternalContext.InitializeContext()
```

Step 9. Add the **connection string** settings to your App.config file or copy the entire **App.config** file from the previous example:

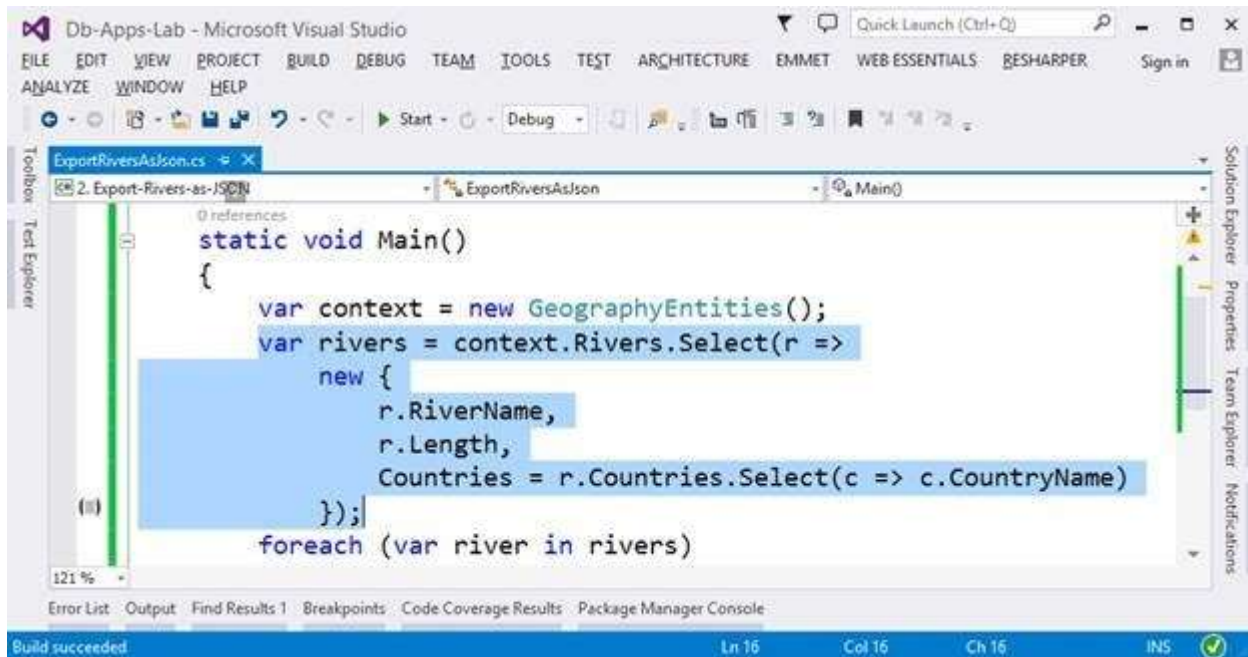


Step 10. Now run your program again. It should **list all rivers**:



```
C:\WINDOWS\system32\cmd.exe
Salween
Saint Lawrence
Rio Grande
Lower Tunguska
Brahmaputra
Danube
Press any key to continue . . .
```

Step 11. Write a **LINQ query** to select all rivers, their name, length and countries:



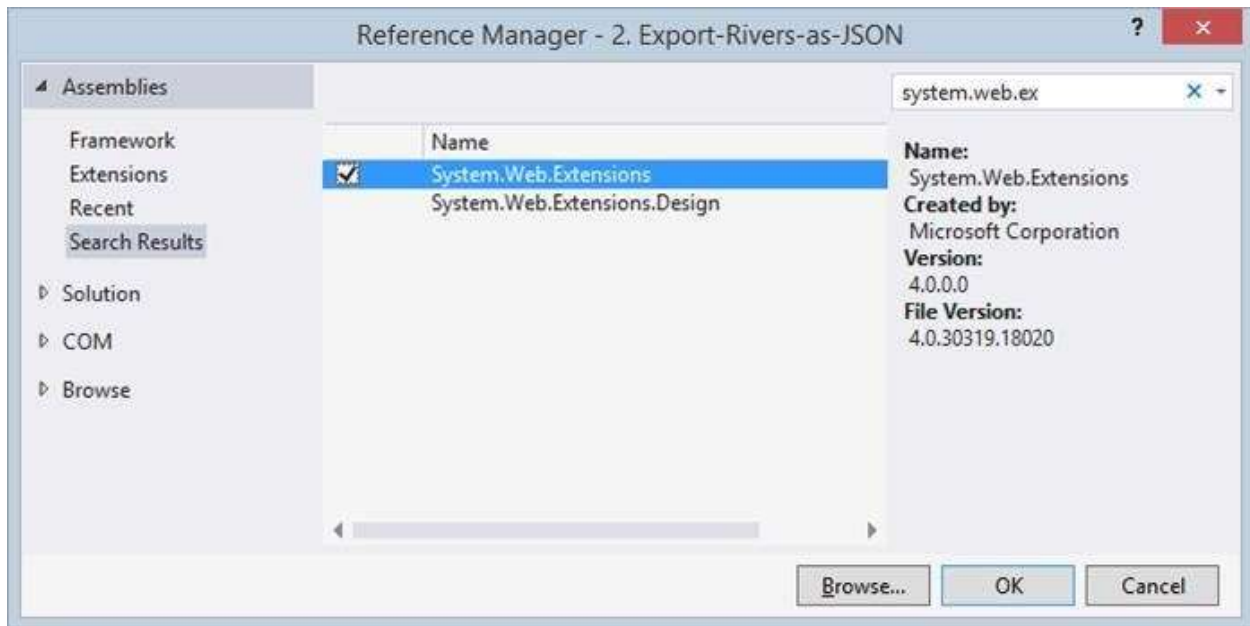
Step 12. Test your query by running the program and checking the output.

Step 13. Add the requested **sorting by river length** (from longest) and **sorting by country name** alphabetically:

```
var context = new GeographyEntities();
var riversQuery = context.Rivers
    .OrderByDescending(r => r.Length)
    .Select(r => new {
        r.RiverName,
        r.Length,
        Countries = r.Countries
            .OrderBy(c => c.CountryName)
            .Select(c => c.CountryName)
    });
```

Step 14. Print the results to the console with a foreach loop.

Step 15. Finally, you have to **convert the results to JSON**. You can use the **JavaScriptSerializer**. First add a reference to the .NET assembly **System.Web.Extensions.dll**:

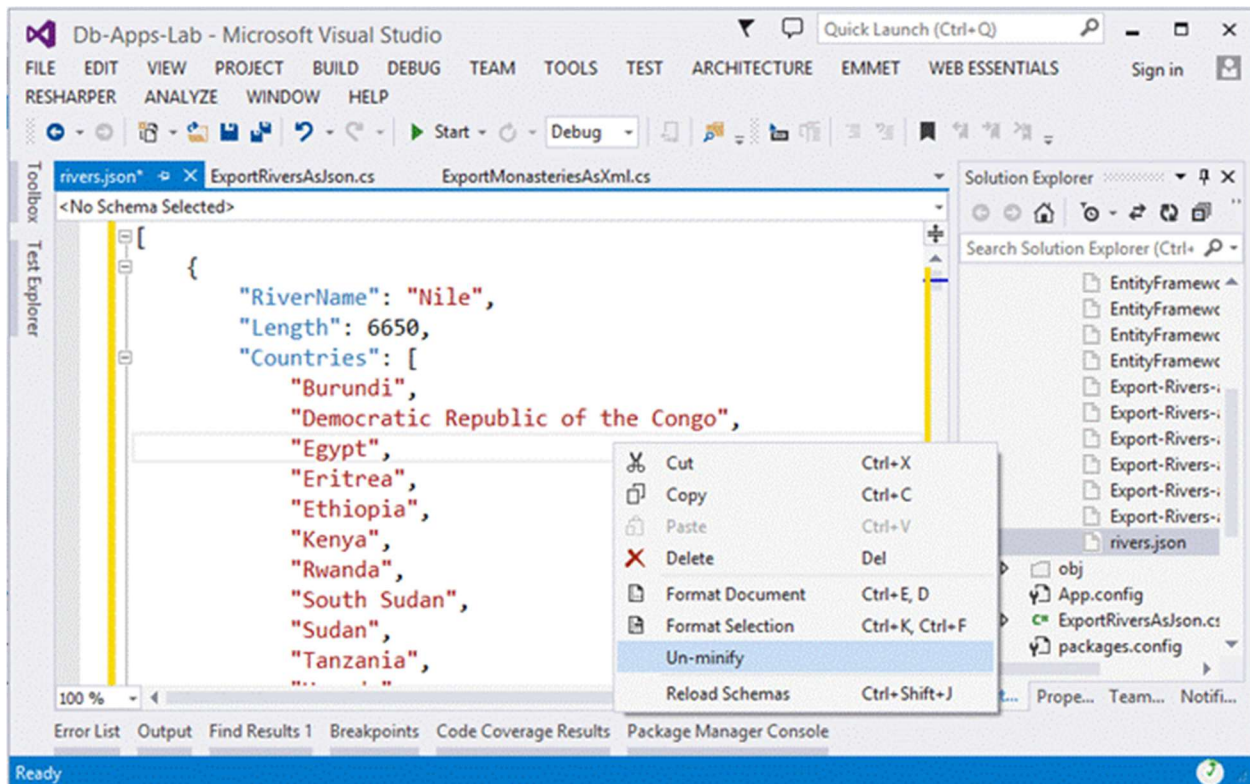


Step 16. Use the following code to **serialize the results as JSON** string:

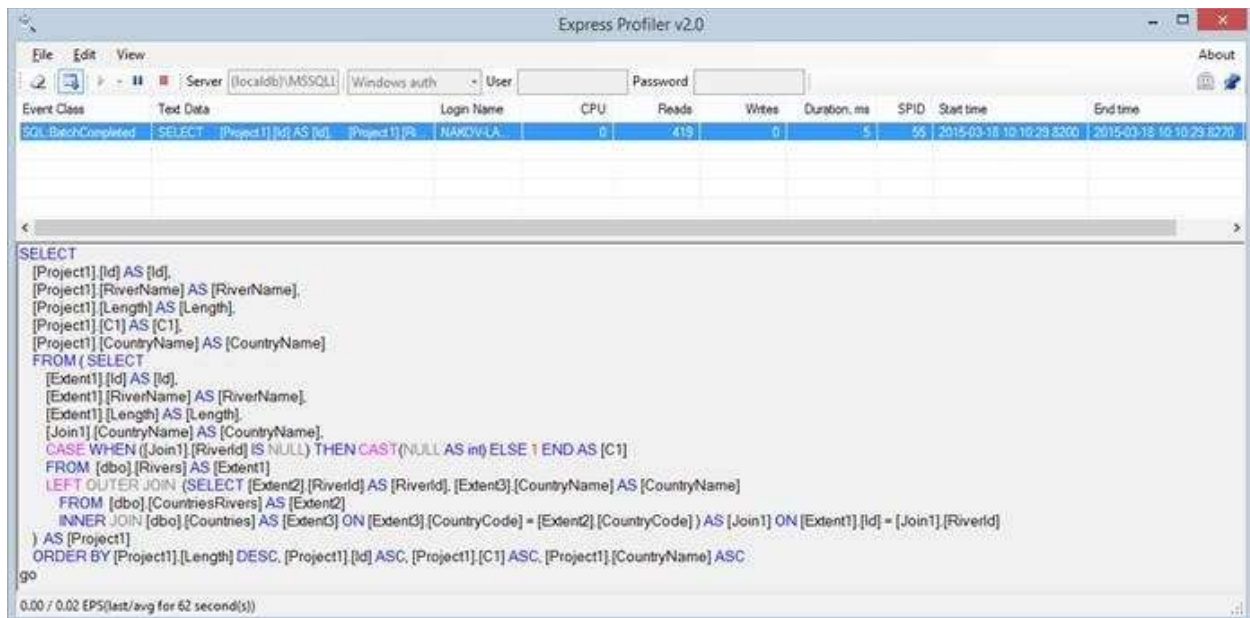
```
var jsSerializer = new JavaScriptSerializer();  
var riversJson = jsSerializer.Serialize(riversQuery.ToList());  
Console.WriteLine(riversJson);
```

Step 17. Finally, save the result JSON string to a text file named **“rivers.json”**. Find in Internet how to *“write text to file in C#”*. You may use **File.WriteAllText(...)**.

Step 18. Run and **test your program**. Open the output file **bin\Debug\rivers.json** in Visual Studio and check whether it holds the correct results as expected. **Un-minify the JSON** to check it for correctness. If the **“Un-minify”** option is unavailable, update your Visual Studio to the latest version or install Web Essentials for Visual Studio (<http://vswebessentials.com>).



Step 19. Check for **performance problems** and ensure your program executes a **single SQL query**. You might use the **SQL Express Profiler** to see all executed queries:



Problem 3. Export Monasteries by Country as XML

Write a **C# application** based on your EF data model for **exporting all monasteries by country** to a XML file named `monasteries.xml` in the following XML format:

```
1
2    <?xml version="1.0" encoding="utf-8"?>
3    <monasteries>
4        <country name="Bhutan">
5            <monastery>Taktsang Palphug Monastery</monastery>
6        </country>
7        <country name="Bulgaria">
8            <monastery>Bachkovo Monastery "Virgin Mary"</monastery>
9            <monastery>Rila Monastery "St. Ivan of Rila"</monastery>
10           <monastery>Troyan Monastery "Holy Mother's Assumption"</monastery>
11        </country>
12    ...
13 </monasteries>
```

Exclude all countries with no monasteries. Use an XML parser by choice. Order the **countries alphabetically** and the **monasteries** in each country also **alphabetically**.

For better performance, ensure your program executes a **single DB query** and retrieves from the database only the required data (without unneeded rows and columns).

Step 1. Create a **new Console Application** called **“Export-Monasteries-as-XML”** in your VS solution.

Step 2. Rename the project from **“Export-Monasteries-as-XML”** to **“3. Export-Monasteries-as-XML”**.

Step 3. Add a **NuGet reference** to the package **“Entity Framework”**.

Step 4. Add a reference to the project holding EF data model **“1. EF-Mappings”**.

Step 5. Rename the class **“Program”** to **“ExportMonasteriesAsXml”**.

Step 6. Copy the **App.config** from the project **“1. EF-Mappings”**. Thus, you will copy the DB connection string.

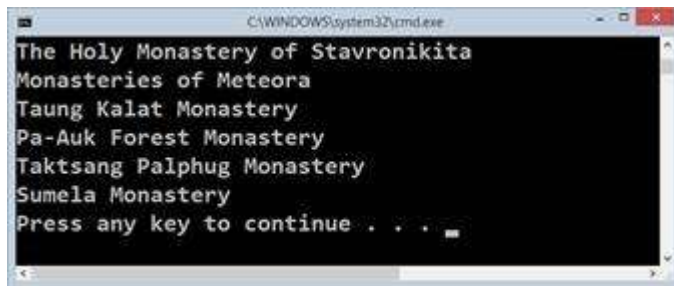
Step 7. Write some code to **list all monasteries**, for example:


```

static void Main()
{
    var context = new GeographyEntities();
    foreach (var monastery in context.Monasteries)
    {
        Console.WriteLine(monastery.Name);
    }
}

```

Step 8. Run and **test your code**. Ensure the monasteries are listed correctly:



Step 9. Write a query to select all the **monasteries alphabetically** along with all their **countries alphabetically**.

To **test your query**, print the monasteries and their countries to the console. Your code might look similar to this:

```

var countriesQuery = context.Countries
    .OrderBy(c => c.CountryName)
    .Select(c => new {
        c.CountryName,
        Monasteries = c.Monasteries
            .OrderBy(m => m.Name)
            .Select(m => m.Name)
    });
foreach (var country in countriesQuery)
{
    Console.WriteLine(country.CountryName + " " + string.Join(", ", country.Monasteries));
}

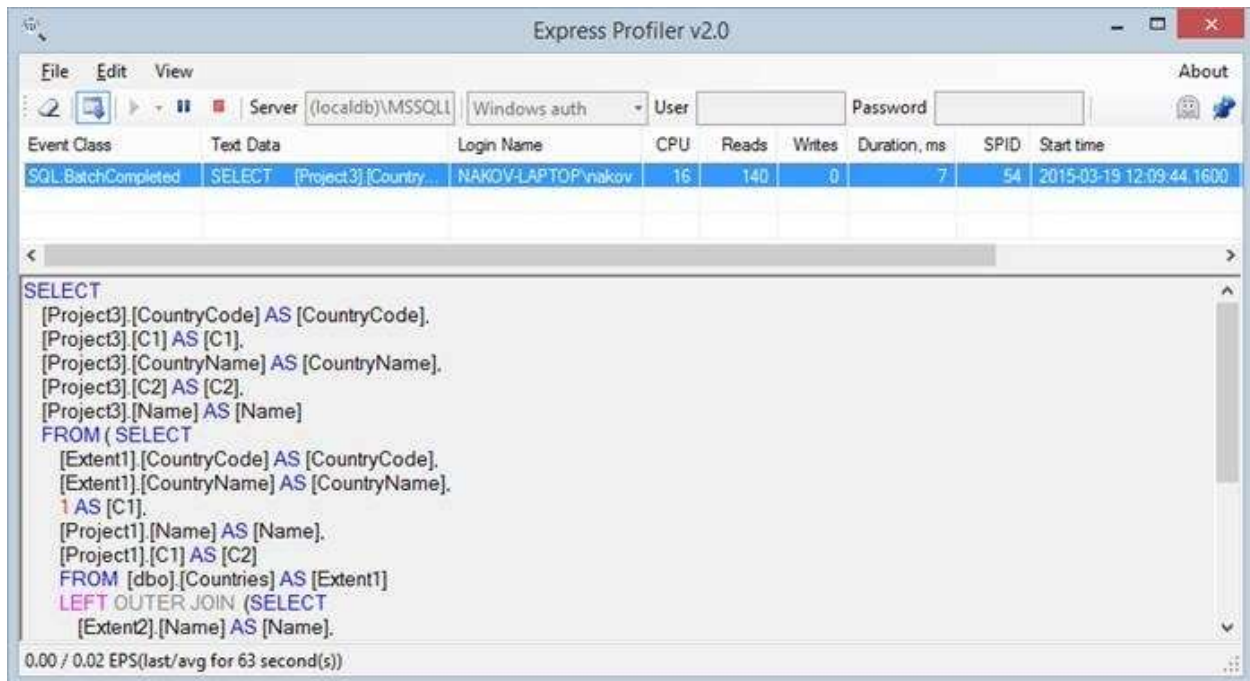
```

Step 10. Exclude from the query the countries with no monasteries. You may use a **.Where(...)** filter along with **monasteries.Any()**.

Note that you should first filter the data, then sort and finally select. If you put the **.Where(...)** filter last, it will cause runtime error during the translation of the LINQ query to SQL.

Step 11. Test again to ensure all countries with their monasteries are listed correctly.

Step 12. Check for **performance problems** and ensure your program executes a **single SQL query** (and the N+1 query problem is avoided). You might use the **SQL Express Profiler** to see all executed queries:



Step 13. Now you have to **build the output XML**. The easiest way is to use the **XElement** class (the LINQ to XML parser). You can start with the root element:

```
var xmlMonasteries = new XElement("monasteries");
```

Step 14. Then you can **iterate through all countries** and append each country as child XElement:

```
foreach (var country in countriesQuery)
{
    var xmlCountry = new XElement("country");
    xmlCountry.Add(new XAttribute("name", country.CountryName));
    xmlMonasteries.Add(xmlCountry);
}
```

Step 15. As next step you could **iterate though all monasteries** in each country and attach the current monastery as child XElement:

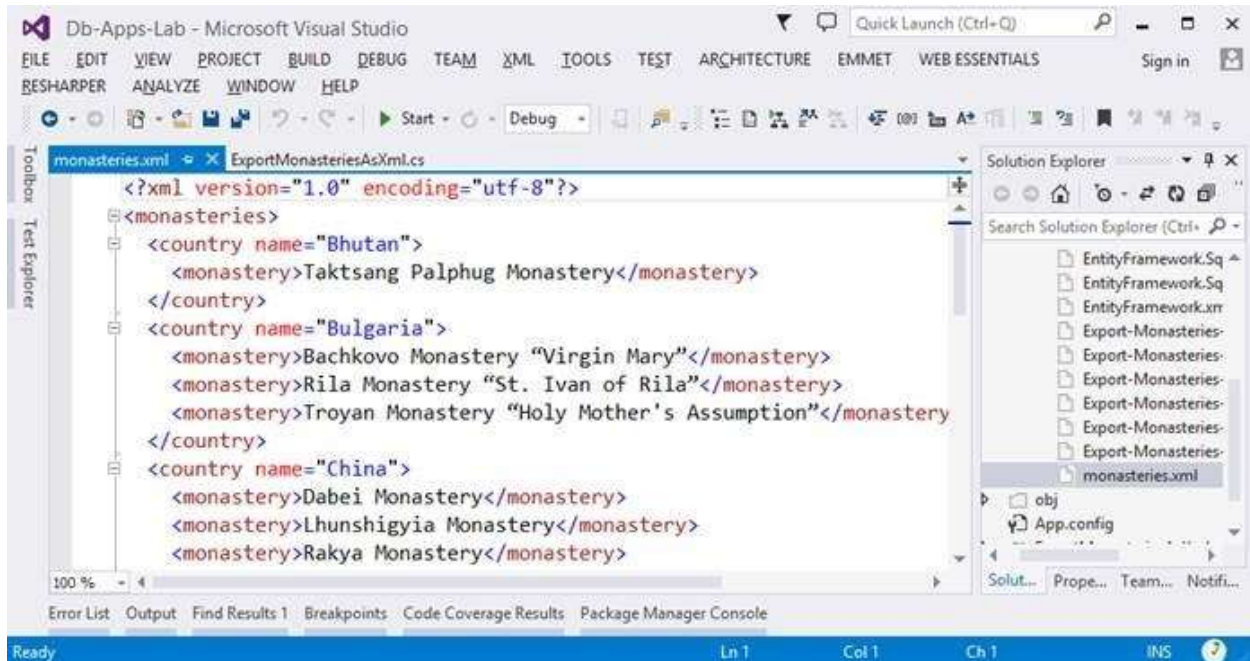
```
foreach (var monastery in country.Monasteries)
{
    xmlCountry.Add(new XElement("monastery", monastery));
}
```

Step 16. Print the output XML on the console to ensure it is correct.

Step 17. Finally save the XML to file:

```
var xmlDoc = new XDocument(xmlMonasteries);  
xmlDoc.Save("monasteries.xml");
```

Step 18. Check the output file `bin\Debug\monasteries.xml`:



Problem 4. Import Rivers from XML

Write a **C# application** based on your EF data model for **importing into the DB a set of rivers** given in the XML file `rivers.xml`. The rivers come in the following XML format:

rivers.xml

```
1    <?xml version="1.0" ?>  
2    <rivers>  
3      <river>  
4        <name>Maritsa</name>  
5        <length>480</length>  
6        <outflow>Aegean Sea</outflow>  
7        <countries>  
8          <country>Bulgaria</country>  
9        </countries>  
10     </river>  
11     <river>  
12       <name>Madre de Dios</name>  
13       <length>1130</length>  
14       <drainage-area>125000</drainage-area>  
15       <average-discharge>4915</average-discharge>  
16       <outflow>Beni River</outflow>
```

```
14         <countries>
15             <country>Peru</country>
16             <country>Bolivia</country>
17         </countries>
18     </river>
19     ...
20 </rivers>
21
22
23
```

The name, length and outflow elements are **mandatory**. The drainage-area, average-discharge and countries elements are **optional**.

You should **parse the XML** and throw an **exception** in case of incorrect data, e.g. when a required element is missing or an invalid value is given. The size of the XML file will be less than **10 MB**. Use an XML parser by choice.

Step 1. Create a **new Console application** named **“Import-Rivers-From-XML”** in your VS solution.

Step 2. Rename the application to **“4. Import-Rivers-From-XML”**.

Step 3. Add a **NuGet reference** to the package **“Entity Framework”**.

Step 4. To reuse the EF data model, **add a reference** to the project **“1. EF-Mappings”**.

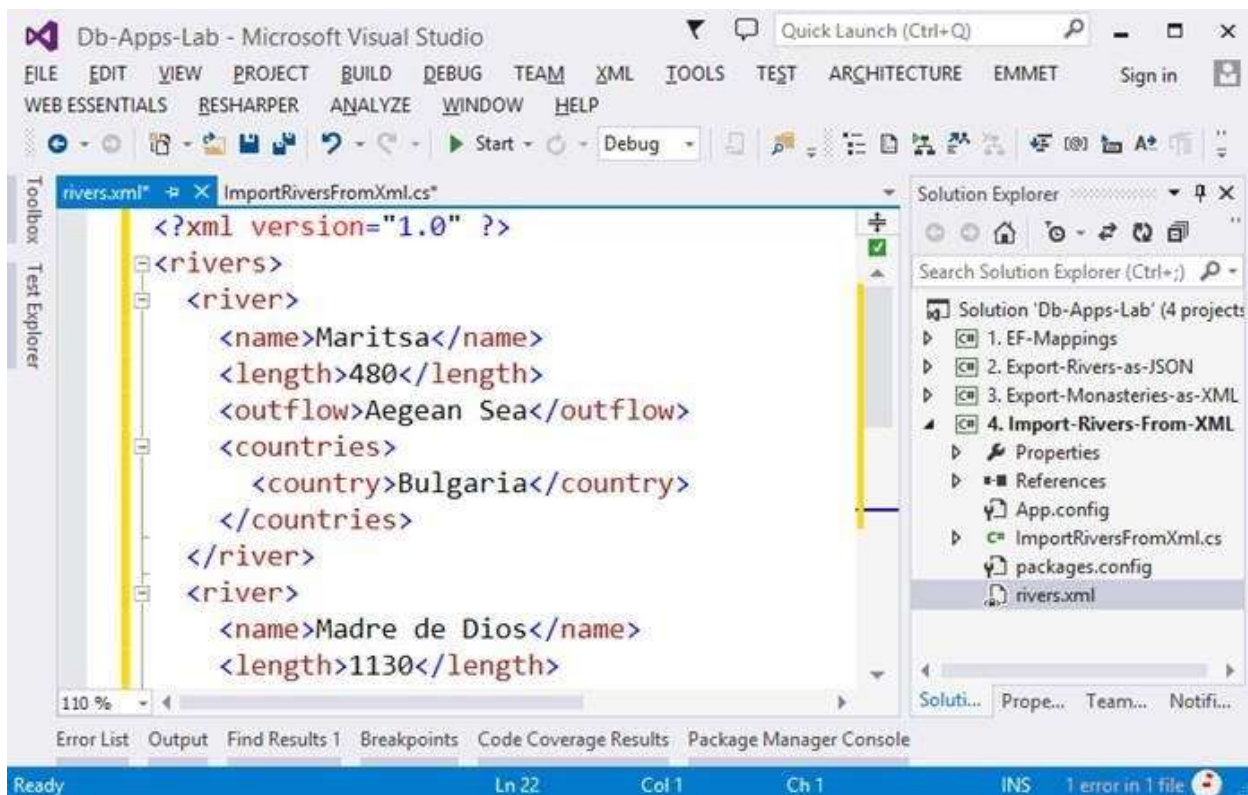
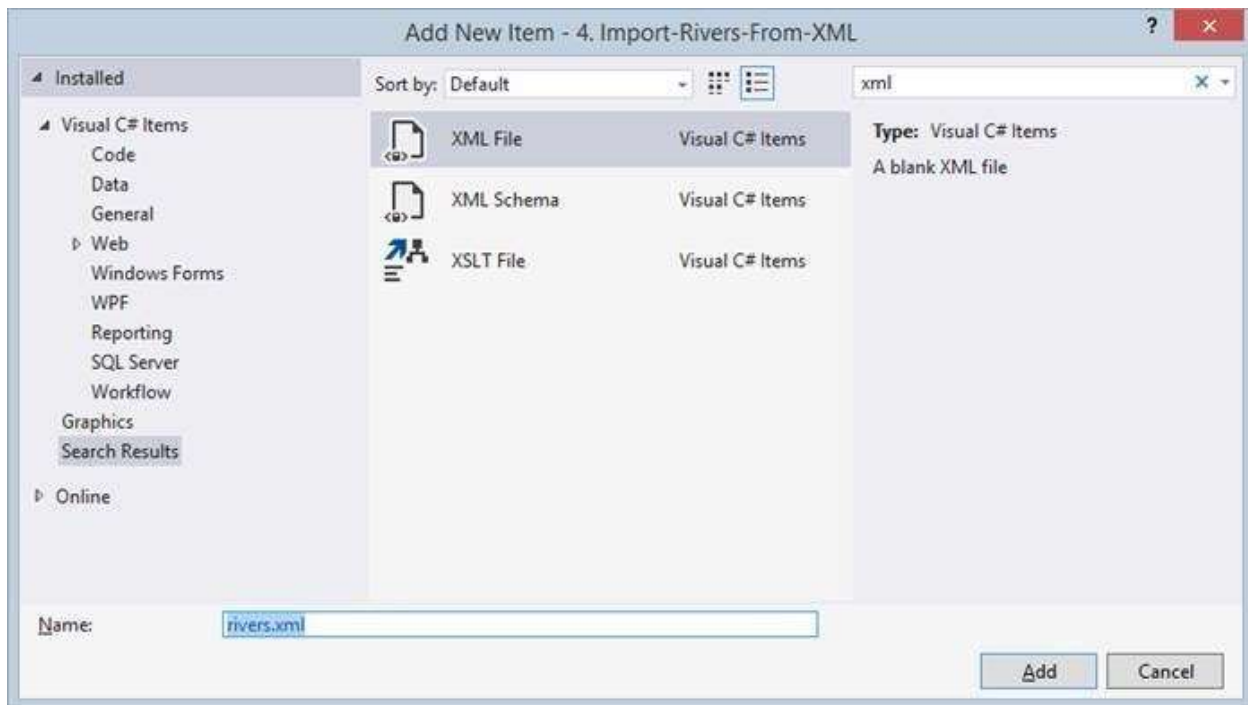
Step 5. Rename the class **“Program”** to **“ImportRiversFromXml”**. Rename also the file **“Program.cs”** to **“ImportRiversFromXml.cs”**.

Step 6. Replace your local **App.config** file with the **App.config** from the project **“1. EF-Mappings”**. This will copy transfer your connection string settings.

Step 7. To **test whether you have correctly configured EF**, the DB connection string and you have database access, you can query for the number of rivers from the database and print the result on the console:

```
Console.WriteLine(new GeographyEntities().Rivers.Count());
```

Step 8. The next step is to **parse the XML** input file. First, create a sample file **rivers.xml** and put inside the rivers from the example:



Step 9. As a next step, **load the XML** and print it on the console. You might use the **XDocument** parser:

```
var xmlDoc = XDocument.Load(@"..\..\rivers.xml");
Console.WriteLine(xmlDoc);
```

Step 10. Test your program to ensure you have loaded the XML correctly.

Your compiled project (.exe file) is located in **bin\Debug** folder, so if you want to load the rivers.xml from the root of your C# project, it should be accessed through the path **“..\..\rivers.xml”**.

If the XML **fails to parse**, check whether it is correct. In the problem description, the sample XML is unfinished (it holds “...” at the last line), so you need to finish it.

Step 11. Now you have loaded the XML in the memory as XDocument. The next step is to iterate through all rivers. You may select them with an XPath selector **“/rivers/river”**. Then, you can print the selected nodes:

```
var xmlDoc = XDocument.Load(@"..\..\rivers.xml");
var riverNodes = xmlDoc.XPathSelectElements("/rivers/river");
foreach (var riverNode in riverNodes)
{
    Console.WriteLine(riverNode);
}
```

If the **XPathSelectElements(...)** extension method is unavailable, add **“using System.Xml.XPath”**.

Step 12. Test your code to ensure it selects and prints all rivers correctly.

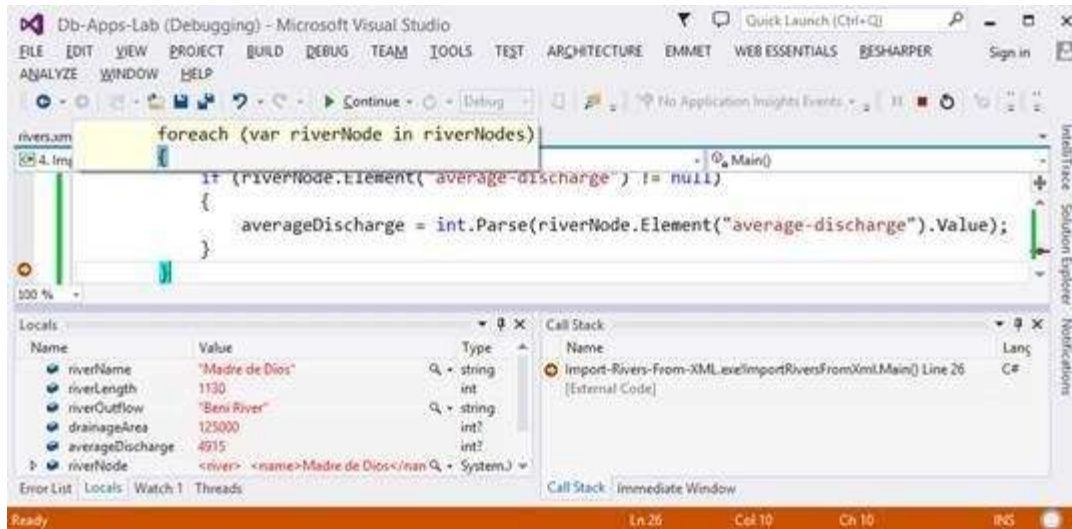
Step 13. Now you should extract the data from each river. First, **extract the mandatory fields**:

```
var xmlDoc = XDocument.Load(@"..\..\rivers.xml");
var riverNodes = xmlDoc.XPathSelectElements("/rivers/river");
foreach (var riverNode in riverNodes)
{
    string riverName = riverNode.Element("name").Value;
    int riverLength = int.Parse(riverNode.Element("length").Value);
    string riverOutflow = riverNode.Element("outflow").Value;
    ...
}
```

Step 14. Next, **extract the optional fields**: drainage area and average discharge. Each optional field should be explicitly checked for **null**:

```
int? drainageArea = null;
if (riverNode.Element("drainage-area") != null)
{
    drainageArea = int.Parse(riverNode.Element("drainage-area").Value);
}
...
```

Step 15. Test your code. The easiest way is to put a breakpoint and ensure the river data is correctly parsed:



Alternatively, you can print the parsed river data on the console:

```
Console.WriteLine("{0} {1} {2} {3} {4}", riverName, riverLength,
    riverOutflow, drainageArea, averageDischarge);
```

Step 16. Now **parse the countries for each river** in the same way, with an XPath selector **"countries/country"**, then print the rivers with its countries on the console:

```
var countryNodes = riverNode.XPathSelectElements("countries/country");
var countries = countryNodes.Select(c => c.Value);

Console.WriteLine("{0} -> {1}", riverName, string.Join(", ", countries));
```

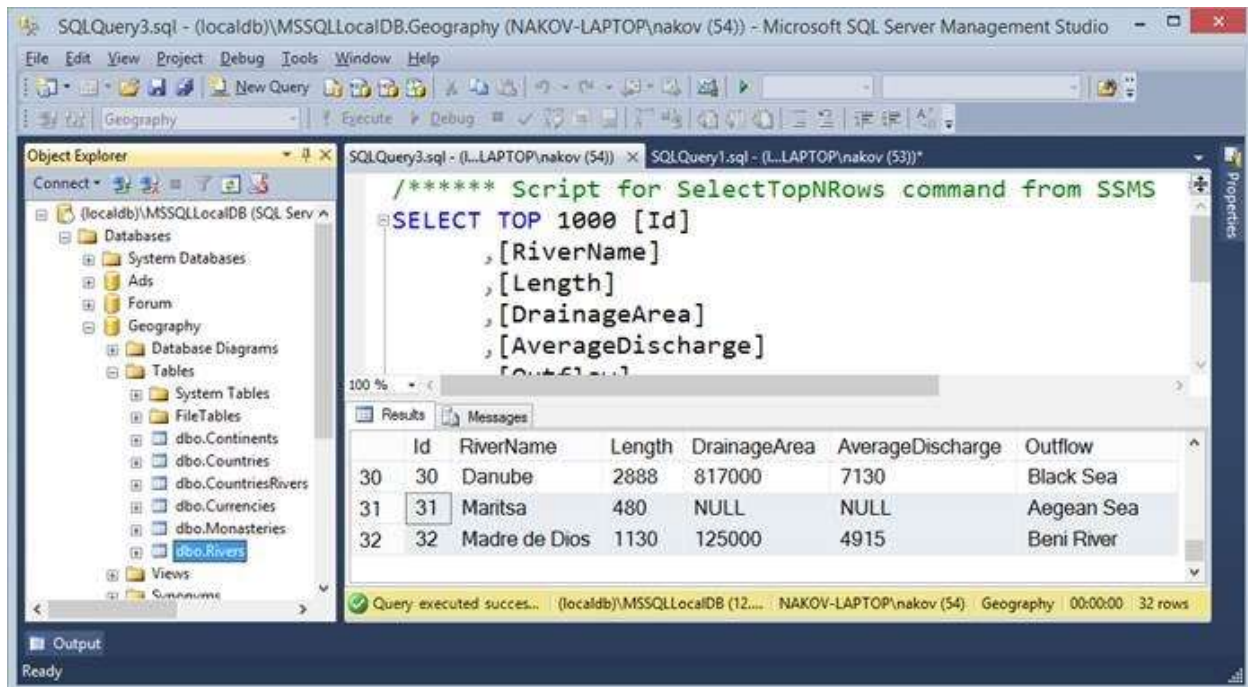
Note: You should use relative XPath selector **"countries/country"**, not full. If you use XPath selector **"/countries/country"** (with leading **"/"**), it will select nothing. If you use **"/rivers/river/countries/country"**, it will select all countries for all rivers.

Step 17. Test your project to ensure the rivers data is correctly parsed.

Step 18. The next step is to **import the parsed rivers into the database**. First create a **River** entity, fill its properties and save it to DB:

```
var context = new GeographyEntities();
var river = new River() { RiverName = riverName, Length = riverLength,
    Outflow = riverOutflow, DrainageArea = drainageArea, AverageDischarge =
    averageDischarge};
context.Rivers.Add(river);
context.SaveChanges();
```

Step 19. Check whether the **rivers are imported** in the database with SQL Server Management Studio:



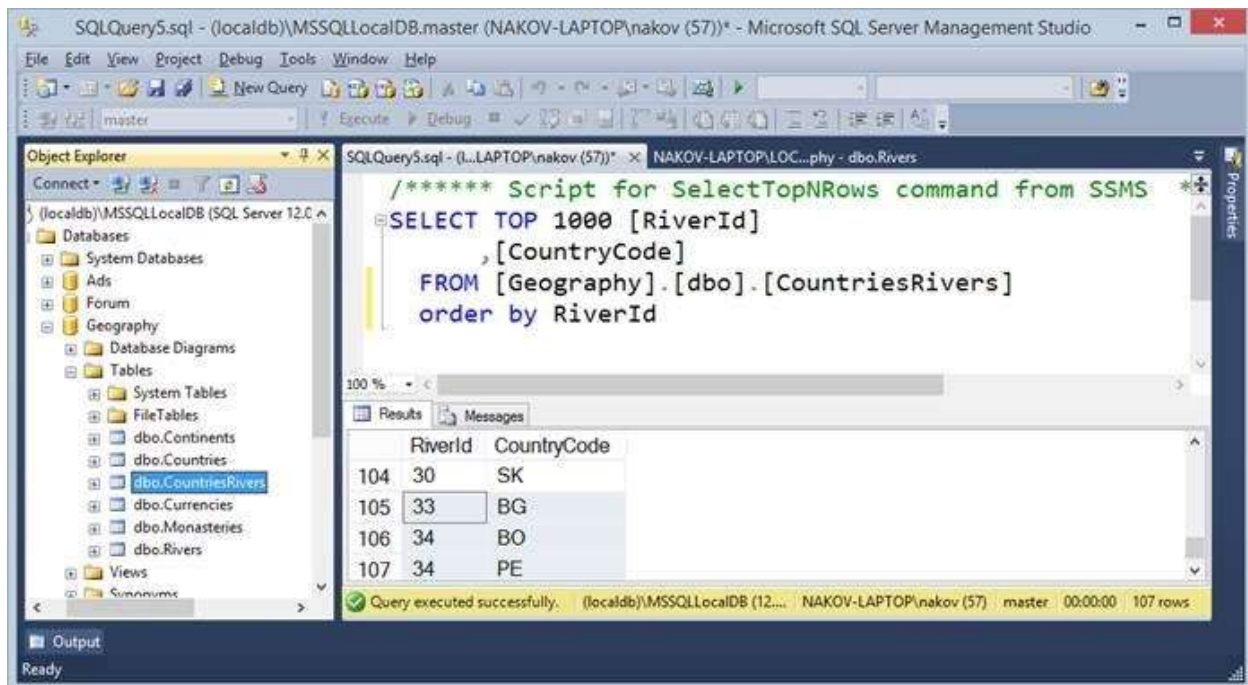
Step 20. Now write the code to **load the countries for each river** in the database. You need to find each country by its name and then add it to the current river:

```

var countryNodes = riverNode.XPathSelectElements("countries/country");
var countryNames = countryNodes.Select(c => c.Value);
foreach (var countryName in countryNames)
{
    var country = context.Countries
        .FirstOrDefault(c => c.CountryName == countryName);
    river.Countries.Add(country);
}

```

Step 21. Finally, run the program and check in the database whether the countries are correctly added to the imported rivers:



Problem 5. EF Code First: Countries, Mountains and Peaks

Create an **Entity Framework (EF) code first data model** for keeping countries, mountains and peaks.

- **Countries** have **country code** (2 Latin letters) and **country name**.
- **Mountains** have a **name** and belong to **multiple countries**.
- **Peaks** have a **name**, **elevation** and **mountain**.

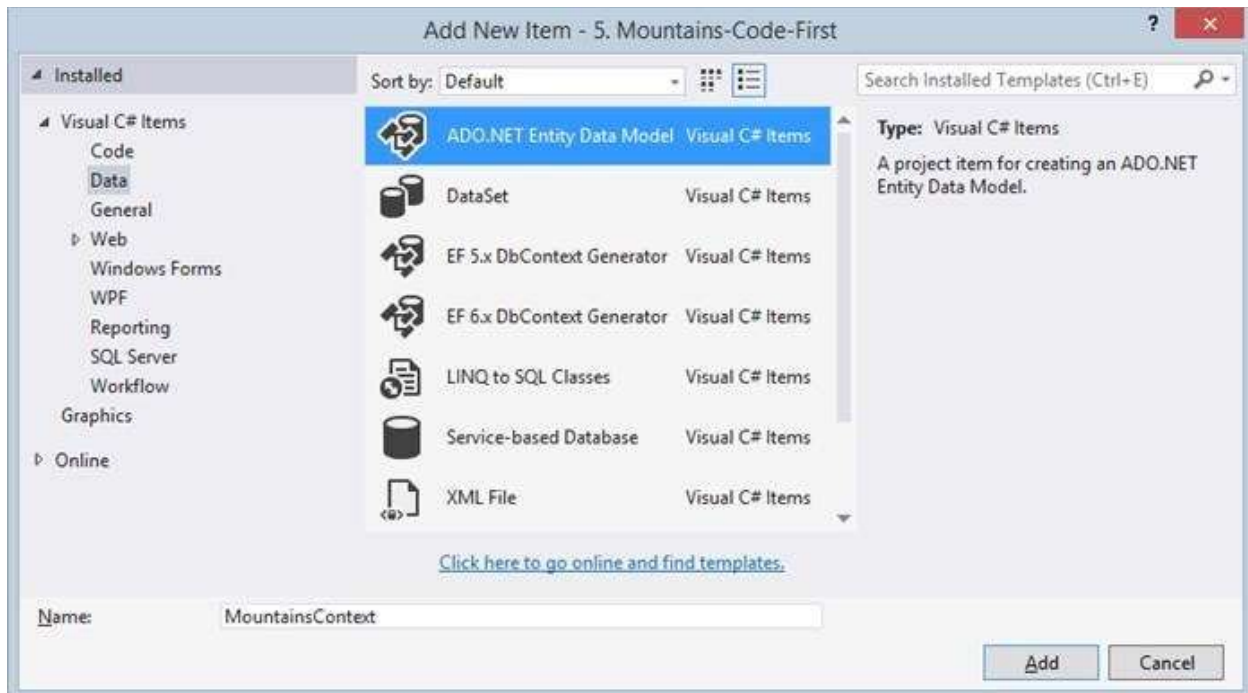
Step 1. Create a new **Console application** named **"Mountains-Code-First"** in your VS solution.

Step 2. Rename the application to **"5. Mountains-Code-First"**.

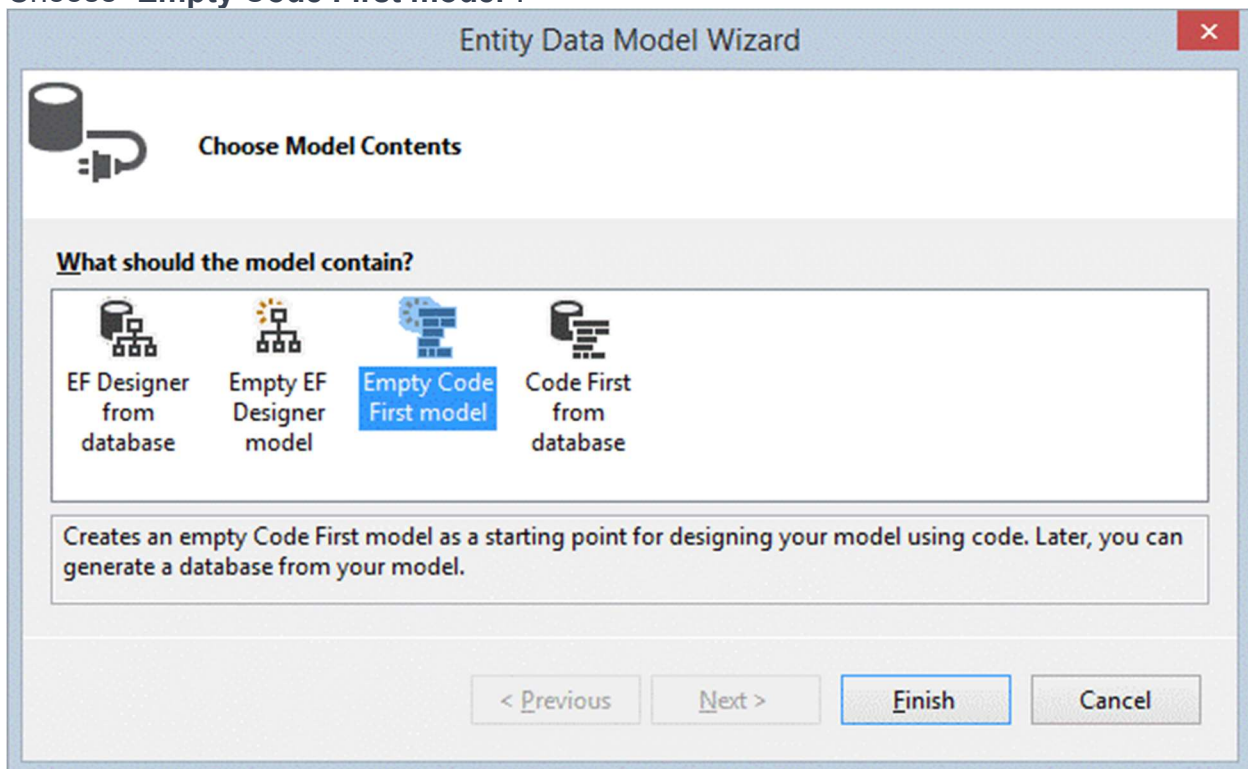
Step 3. Add a **NuGet reference** to the package **"Entity Framework"**.

Step 4. Rename the class **"Program"** to **"MountainsCodeFirst"**. Rename also the file **"Program.cs"** to **"MountainsCodeFirst.cs"**.

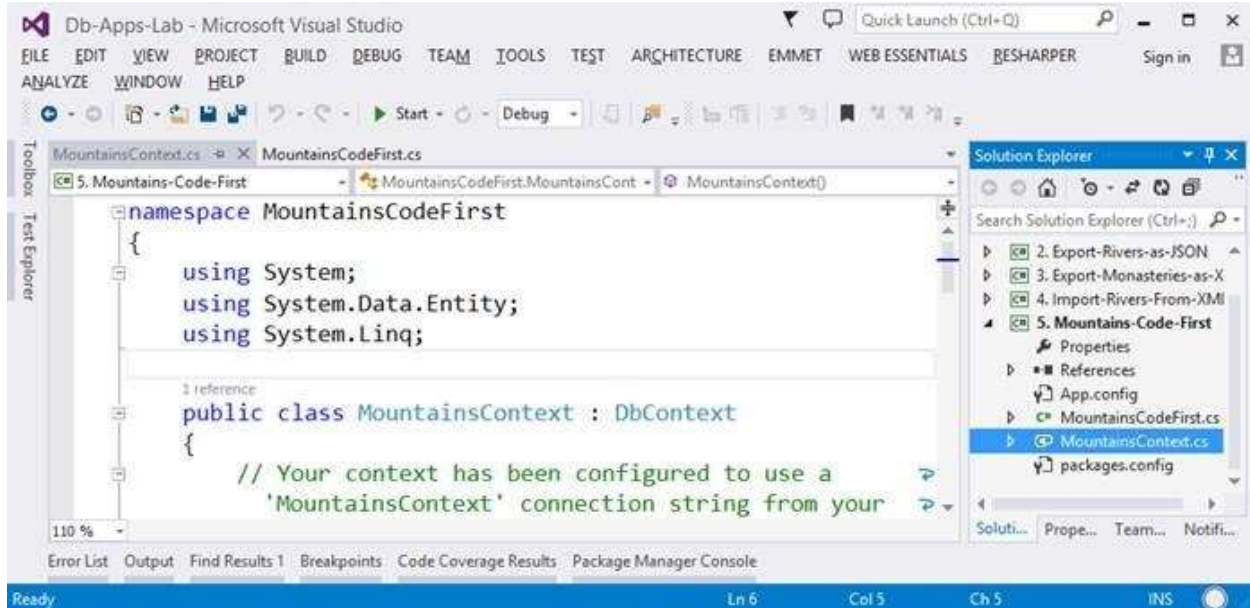
Step 5. Create a new **Entity Framework Code First Data Model**. Click on the VS project, select [Add] -> [New Item...] -> [Data] -> [ADO.NET Entity Data Model]. Name it **"MountainsContext"**.



Choose “**Empty Code First model**”:



Visual Studio will generate for you a skeleton for your **MountainsContext**:



Step 6. Now create your first entity class **Country**. It should hold **country code** (two characters, primary key) and **country name** (string, required). Entity classes should be public. The **Country** class might look like this:

```
public class Country
{
    [Key]
    [StringLength(2, MinimumLength = 2)]
    public string Code { get; set; }

    [Required]
    public string Name { get; set; }
}
```

Step 7. Add the Country entity in your DB context. First clean-up the **MountainsContext** class, then add a **DbSet** of countries:

```
public class MountainsContext : DbContext
{
    public MountainsContext()
        : base("name=MountainsContext")
    {
    }

    public virtual DbSet<Country> Countries { get; set; }
}
```

Step 8. The best way to continue is to **test the project**. First, configure the database **connection string** in App.config. It should hold the server name (e.g. “(localdb)\MSSQLLocalDB” or “.\SQLEXPRESS”) and the database name (e.g. “Mountains”):

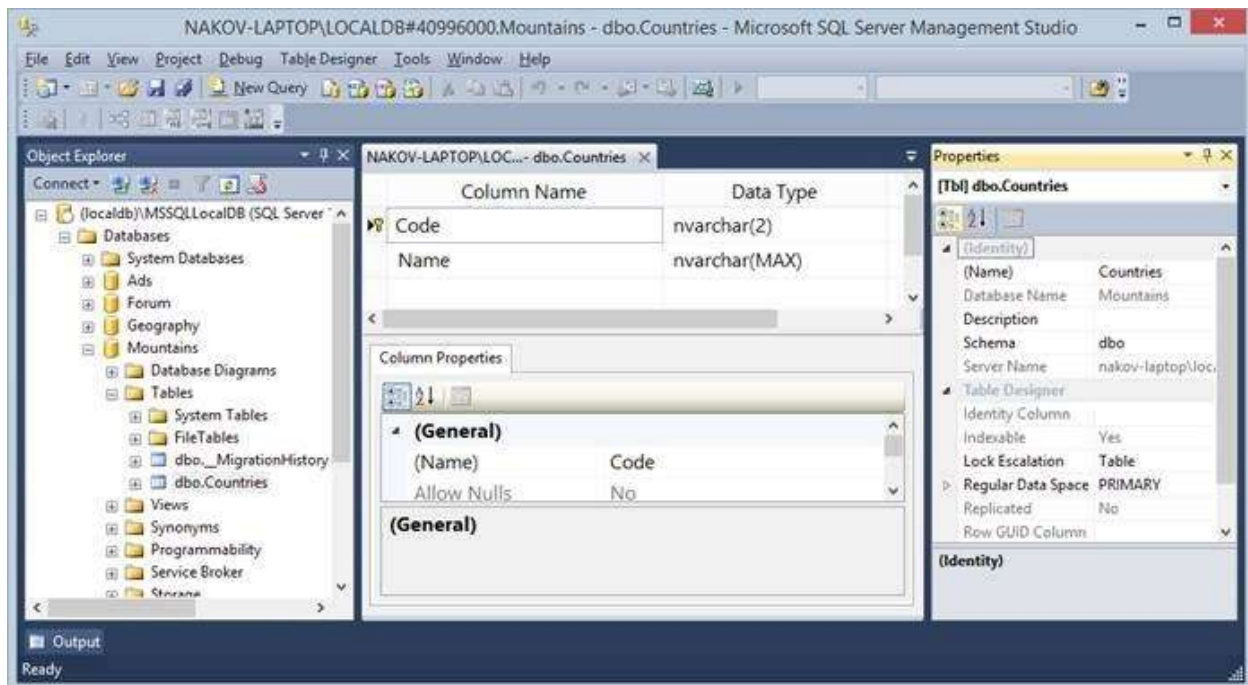
```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>...</configSections>
  <entityFramework>...</entityFramework>
  <connectionStrings>
    <add name="MountainsContext" connectionString="data source=(localdb)\MSSQLLocalDB;initial catalog=Mountains;" />
  </connectionStrings>
</configuration>
```

Step 9. To **test the EF code first data model**, try to find the countries count from the database:

```
class MountainsCodeFirst
{
    0 references
    private static void Main()
    {
        var context = new MountainsContext();
        Console.WriteLine(context.Countries.Count());
    }
}
```

Step 10. Run your application. It should print “0” on the console, without any error messages.

- If your application **runs without errors**, it should have created the database “Mountains” (see your connection string). Try to open the database and check whether the “Countries” tables is correctly defined. It should have columns “Code” (primary key) and “Name” of correct data type.



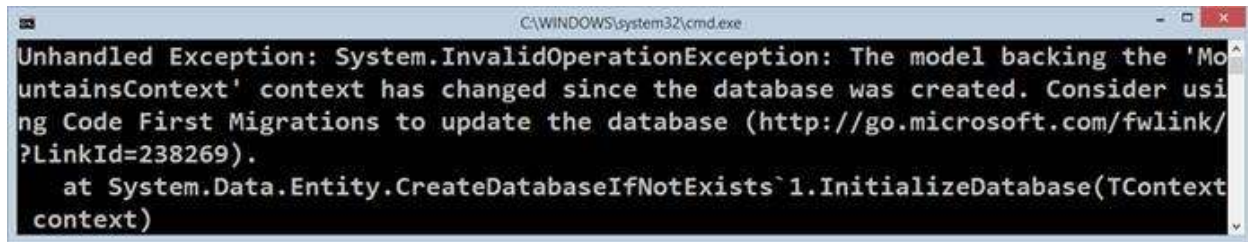
- If your application **fails with an exception**, this is most probably due to incorrect connection string. You can fix your connection string in your **App.config** file.

Step 11. It looks like EF has generated the column **Country.Code** of type **nvarchar(2)** instead of **char(2)**. This can be fixed by the configuration attribute **[Column]**:

```
public class Country
{
    [Key]
    [StringLength(2, MinimumLength = 2)]
    [Column(TypeName = "char")]
    public string Code { get; set; }

    [Required]
    public string Name { get; set; }
}
```

Step 12. Now run the code after the fix. It will fail with **"InvalidOperationException: The model has changed since the database was created."**. This is completely normal, because the project has not been configured to use **"Code First Migrations"**:



```
Unhandled Exception: System.InvalidOperationException: The model backing the 'MountainsContext' context has changed since the database was created. Consider using Code First Migrations to update the database (http://go.microsoft.com/fwlink/?LinkId=238269).
   at System.Data.Entity.CreateDatabaseIfNotExists`1.InitializeDatabase(TContext context)
```

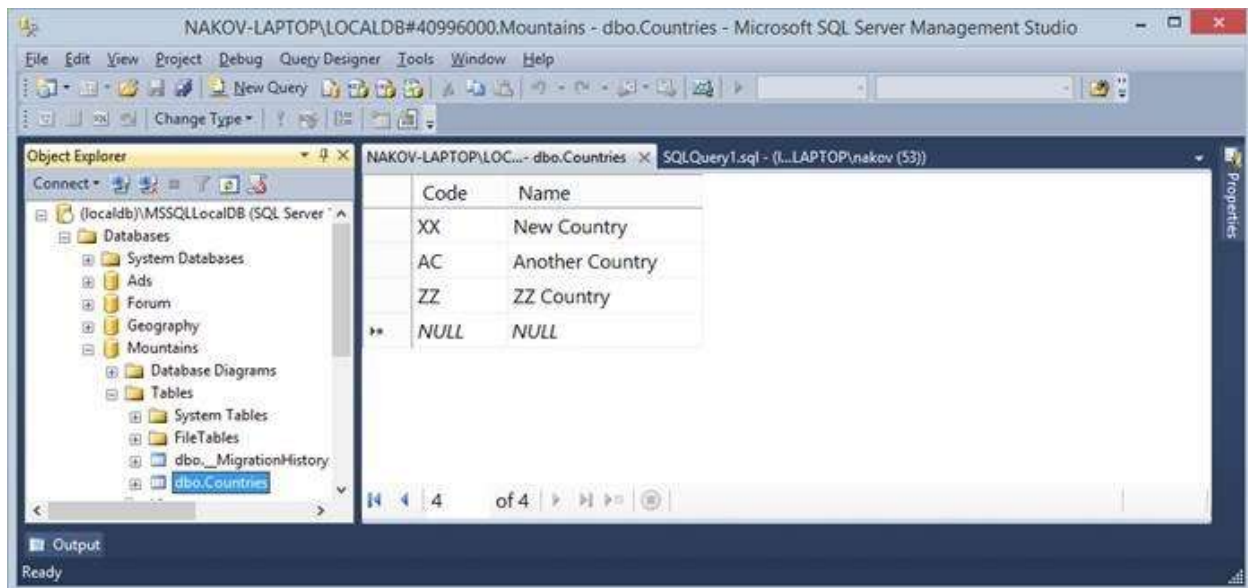
Step 13. By default the migration strategy in EF code first projects is **CreateDatabaseIfNotExists**. You can change this strategy with another: **DropCreateDatabaseIfModelChanges**. This EF database migration strategy will work well in our case, but avoid it in production systems, because it **drops the database with all data in all tables** each time when you modify the EF data model. To change the default DB migration strategy, put this code at the application start:

```
static void Main()
{
    Database.SetInitializer(
        new DropCreateDatabaseIfModelChanges<MountainsContext>());

    ...
}
```

Step 14. Now **run again your project**. It should **drop the existing database and re-create it** again to apply the changes in the EF code first data model. Sometimes, you could get an exception: **System.Data.SqlClient.SqlException: Cannot drop database "Mountains" because it is currently in use**. If this happens, ensure you have closed SQL Server Management Studio and any other applications that use the database. In the typical case, your program will run and will display "0" countries in the database, because it will be empty after the drop and create.

Step 15. Now open the SQL Server Management Studio and **insert a few new records** in the Countries table:



Step 16. Run your program again. It should show how many countries are available in the database.

Step 17. The next step is to define the **Mountain** and **Peak** entity classes:

```

public class Mountain
{
    0 references
    public int Id { get; set; }

    [Required]
    0 references
    public string Name { get; set; }
}

public class Peak
{
    0 references
    public int Id { get; set; }

    [Required]
    0 references
    public string Name { get; set; }

    0 references
    public int Elevation { get; set; }
}

```

Step 18. Modify your EF database context to **register the new entity classes**:


```

public class MountainsContext : DbContext
{
    1 reference
    public MountainsContext()
        : base("name=MountainsContext")
    {
    }

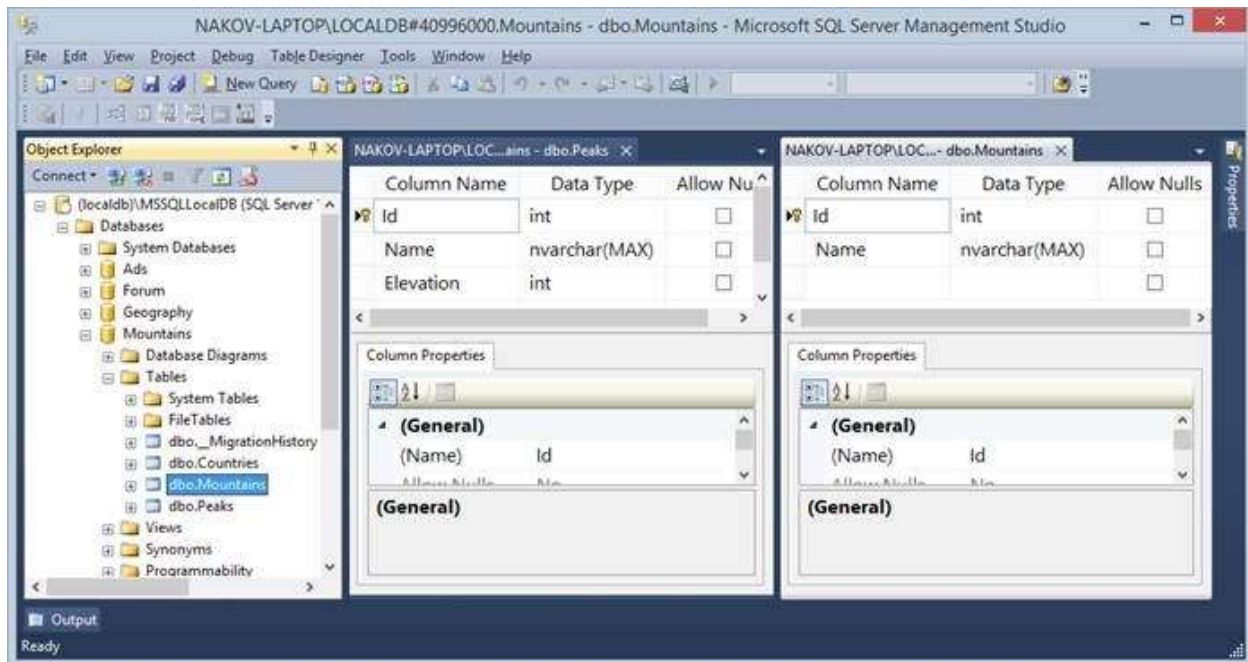
    1 reference
    public virtual DbSet<Country> Countries { get; set; }

    0 references
    public virtual DbSet<Mountain> Mountains { get; set; }

    0 references
    public virtual DbSet<Peak> Peaks { get; set; }
}

```

Step 19. Run your program to test the modified EF data model. The database should be automatically dropped and re-created with tables for the new entity classes. Check the structure of the **tables** in the database:



Step 20. Now we have the **Countries**, **Mountains** and **Peaks** tables in the DB, but no relations are defined between them. Let's **define the relationships**:

- Countries have many mountains (many-to-many):

```

public class Country
{
    0 references
    public Country()
    {
        this.Mountains = new HashSet<Mountain>();
    }

    ...

    2 references
    public virtual ICollection<Mountain> Mountains { get; set; }
}

```

- Mountains have many peaks (one-to-many) and belong to many countries (many-to-many):

```

public class Mountain
{
    0 references
    public Mountain()
    {
        this.Countries = new HashSet<Country>();
        this.Peaks = new HashSet<Peak>();
    }

    ...

    3 references
    public virtual ICollection<Peak> Peaks { get; set; }
}

```

- Peaks belong to certain mountain (many-to-one):

```

public class Peak
{
    ...
    0 references
    public virtual Mountain Mountain { get; set; }
}

```

Step 21. Now test your EF code first data model by **creating a few countries, mountains and peaks**:

```

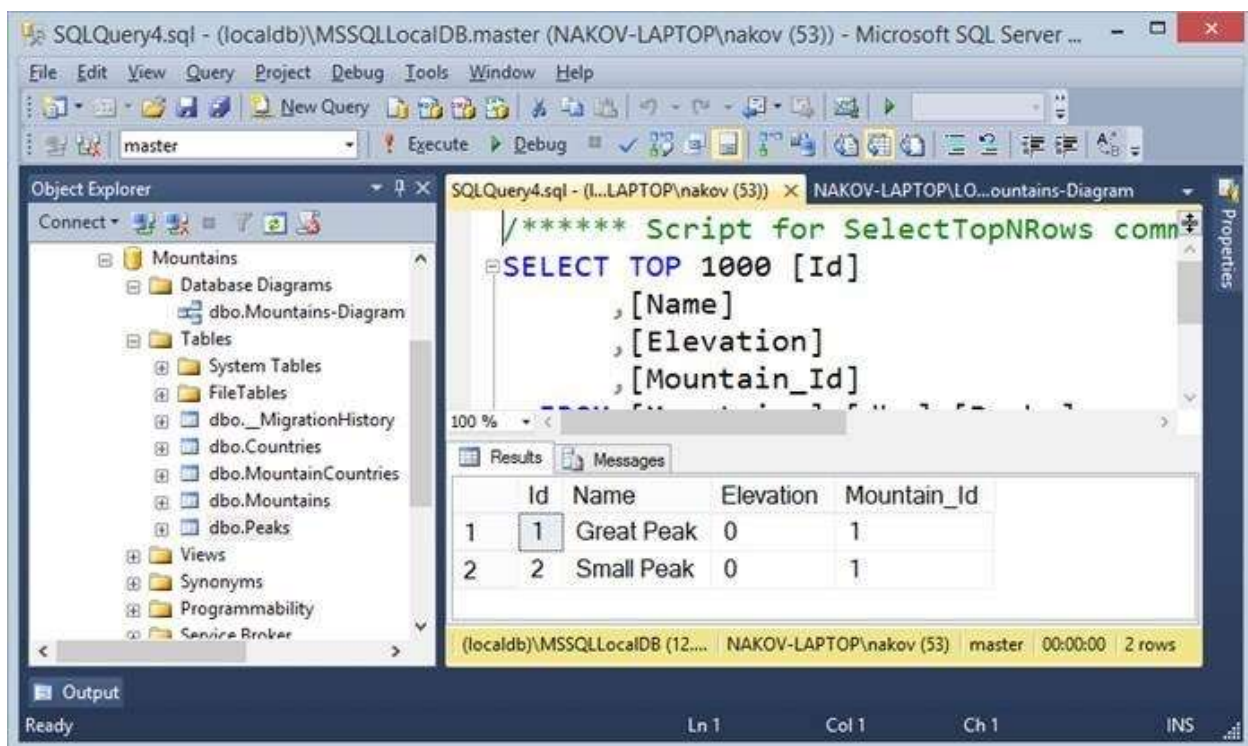
static void Main()
{
    Database.SetInitializer(
        new DropCreateDatabaseIfModelChanges<MountainsContext>());

    Country c = new Country() { Code = "AB", Name = "Absurdistan" };
    Mountain m = new Mountain() { Name = "Absurdistan Hills" };
    m.Peaks.Add(new Peak() { Name = "Great Peak", Mountain = m });
    m.Peaks.Add(new Peak() { Name = "Small Peak", Mountain = m });
    c.Mountains.Add(m);

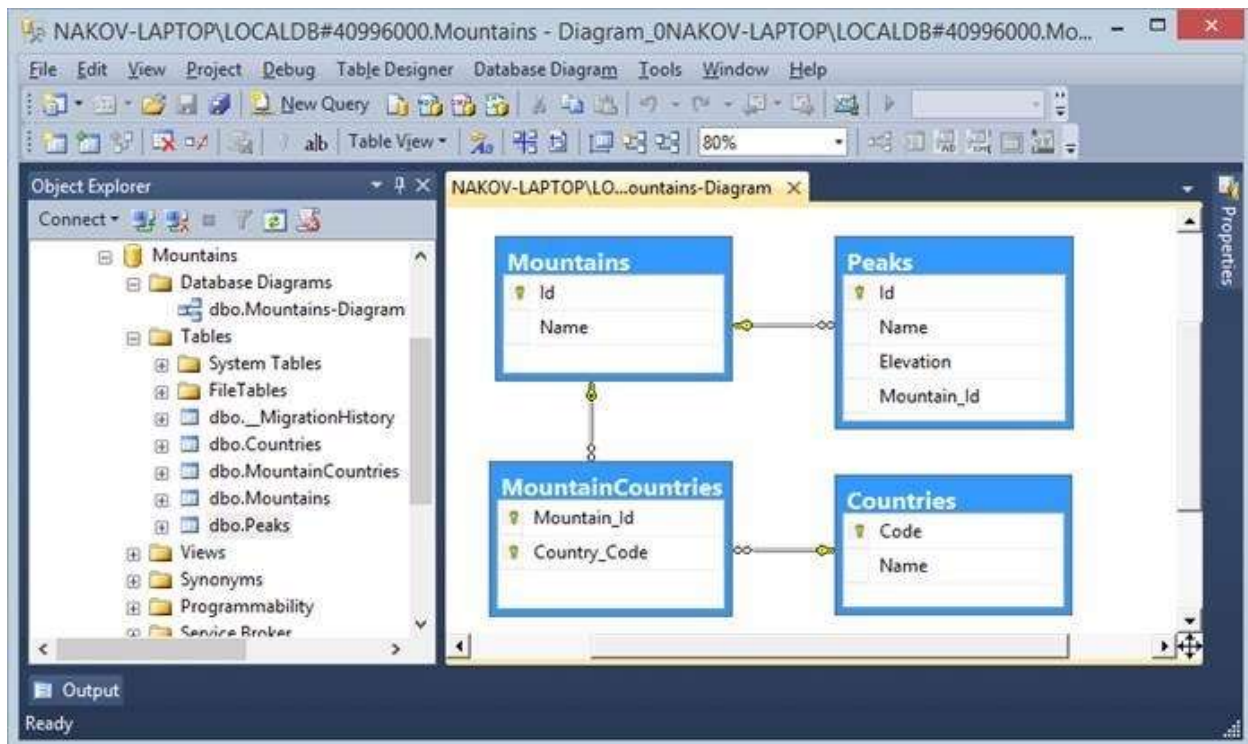
    var context = new MountainsContext();
    context.Countries.Add(c);
    context.SaveChanges();
}

```

Step 22. Run the code. Your database should have a few countries, mountains and peaks:



Step 23. To check whether the tables have correct relationships, create a database diagram in SQL Server Management Studio. It should hold the tables **Countries**, **Mountains**, **Peaks** and the many-to-many table **MountainsCountries** and correct relationships:



Problem 6. EF Code First: Seed the Database

Seed your database with a few countries, mountains and peaks using the EF migrations framework. It is OK to drop the database in case of model changes or use any other migration strategy.

- **Countries:** *Bulgaria* and *Germany*.
- **Mountains:** *Rila*, *Pirin* and *Rhodopes*, all in country *Bulgaria*.
- **Peaks:** *Musala* (elevation 2925, *Rila*), *Malyovitsa* (elevation 2729, *Rila*) and *Vihren* (elevation 2914, *Pirin*).

To test your data model, **list all mountains along with their countries and peaks**.

Step 1. Seeding the database means to initially load some data after the database is created or modified. To seed the database, we could create our own database migration class by inheriting some of the existing EF migration classes (e.g. **DropCreateDatabaseIfModelChanges**) and override their **Seed(...)** method:


```

public class MountainsMigrationStrategy :
    DropCreateDatabaseIfModelChanges<MountainsContext>
{
    0 references
    protected override void Seed(MountainsContext context)
    {
        // Seed data if not already seeded
        var bulgaria = new Country { Code = "BG", Name = "Bulgaria" };
        context.Countries.Add(bulgaria);
        var germany = new Country { Code = "DE", Name = "Germany" };
        context.Countries.Add(germany);

        var rila = new Mountain { Name = "Rila", Countries = { bulgaria } };
        context.Mountains.Add(rila);
        var pirin = new Mountain { Name = "Pirin", Countries = { bulgaria } };
        context.Mountains.Add(pirin);
        var rhodopes = new Mountain { Name = "Rhodopes", Countries = { bulgaria } };
        context.Mountains.Add(rhodopes);

        var musala = new Peak { Name = "Musala", Elevation = 2925, Mountain = rila };
        context.Peaks.Add(musala);
        var malyovitsa = new Peak { Name = "Malyovitsa", Elevation = 2729, Mountain = rila };
        context.Peaks.Add(malyovitsa);
        var vihren = new Peak { Name = "Vihren", Elevation = 2914, Mountain = pirin };
        context.Peaks.Add(vihren);
    }
}

```

The **Seed(...)** method will be invoked after the database is dropped and re-created. You need to configure the application to use your new migration strategy class:

```

class MountainsCodeFirst
{
    0 references
    static void Main()
    {
        Database.SetInitializer(new MountainsMigrationStrategy());

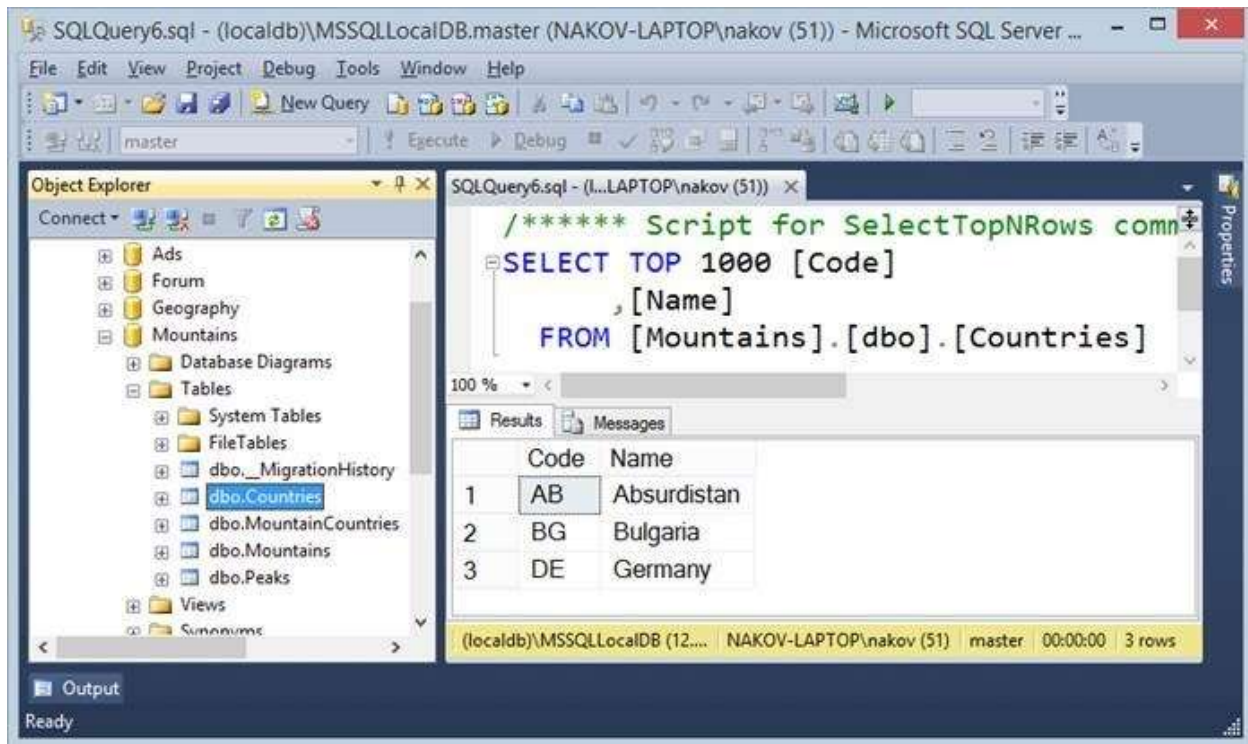
        Country c = new Country() { Code = "AB", Name = "Absurdistan" };
        Mountain m = new Mountain() { Name = "Absurdistan Hills" };
        m.Peaks.Add(new Peak() { Name = "Great Peak", Mountain = m });
        m.Peaks.Add(new Peak() { Name = "Small Peak", Mountain = m });
        c.Mountains.Add(m);

        var context = new MountainsContext();
        context.Countries.Add(c);
        context.SaveChanges();
    }
}

```

Step 2. Now **drop the database** and run the program. It should invoke the **Seed(...)** method. Check the results in the database. You should have 3 countries

(“**Bulgaria**” and “**Germany**” from the seed method and “**Absurdistan**” from the main program):



Step 3. Now list all mountains along with their countries and peaks in order to test the seed:


```

static void Main()
{
    Database.SetInitializer(new MountainsMigrationStrategy());

    var context = new MountainsContext();
    var countriesQuery = context.Countries.Select(c => new {
        CountryName = c.Name,
        Mountains = c.Mountains.Select(m => new {
            m.Name,
            m.Peaks
        })
    });
    foreach (var country in countriesQuery)
    {
        Console.WriteLine("Country: " + country.CountryName);
        foreach (var mountain in country.Mountains)
        {
            Console.WriteLine("    Mountain: " + mountain.Name);
            foreach (var peak in mountain.Peaks)
            {
                Console.WriteLine("        \t{0} ({1})", peak.Name, peak.Elevation);
            }
        }
    }
}

```

Step 4. Run the program to test whether it works correctly. The output should be like this:



```

C:\WINDOWS\system32\cmd.exe
Country: Absurdistan
    Mountain: Absurdistan Hills
        Great Peak (0)
        Small Peak (0)
Country: Bulgaria
    Mountain: Rila
        Musala (2925)
        Malyovitsa (2729)
    Mountain: Pirin
        Vihren (2914)
    Mountain: Rhodopes
Country: Germany
Press any key to continue . . .

```

~~~~~ The End ~~~~~