FSC105

Type Providers

- Lecture will begin shortly
- Download class materials from university.xamarin.com

Xamarin University

Objectives

1. Connect to data sources with type providers
2. Query and transform data from type providers

Connect to data sources with type providers

Xamarin University

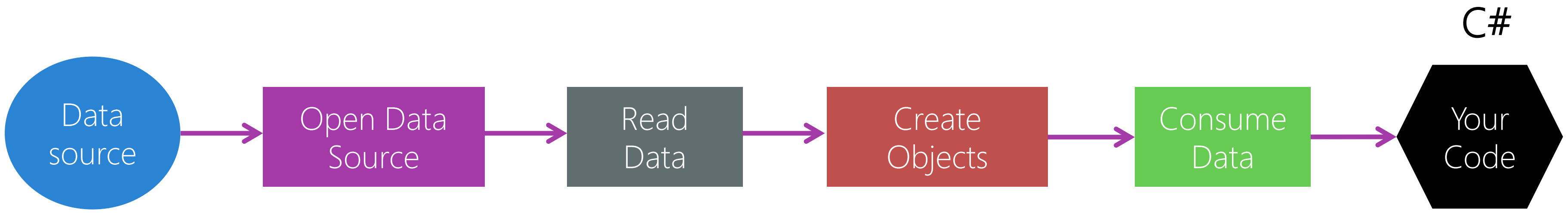
Tasks

1. Define type providers
2. Explore the benefits of type providers
3. Describe how to connect to type providers

Xamarin University

Consuming data

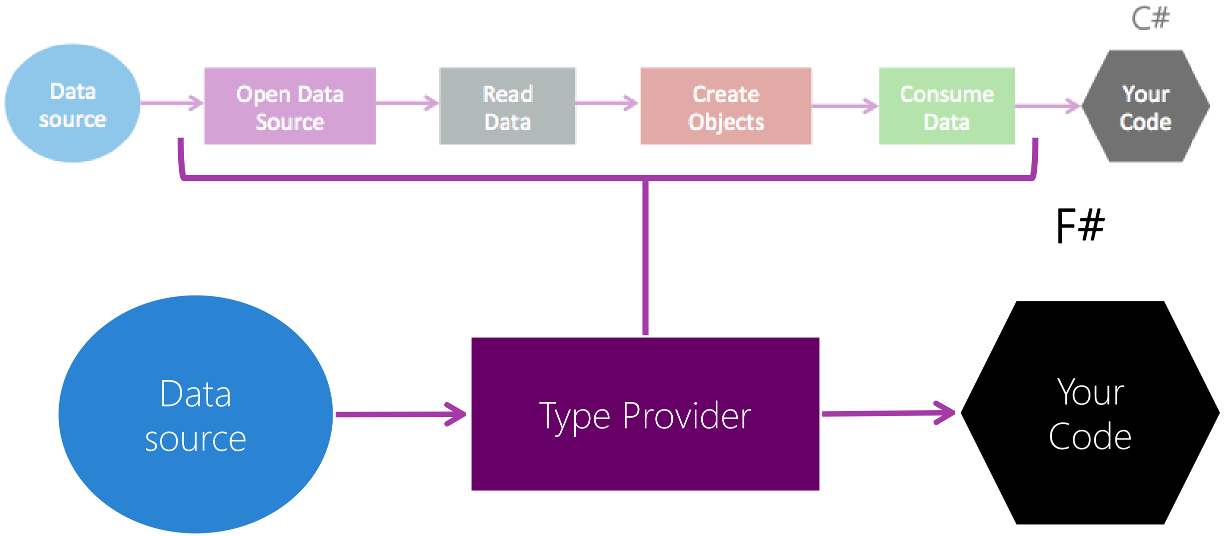
- Consuming external data in a program requires several programmed steps in C#



Often must use *code generation* tools such as **`svcutil.exe`**, **`wsdl.exe`**, **`edmgen.exe`**, etc. to create "code" representation of data

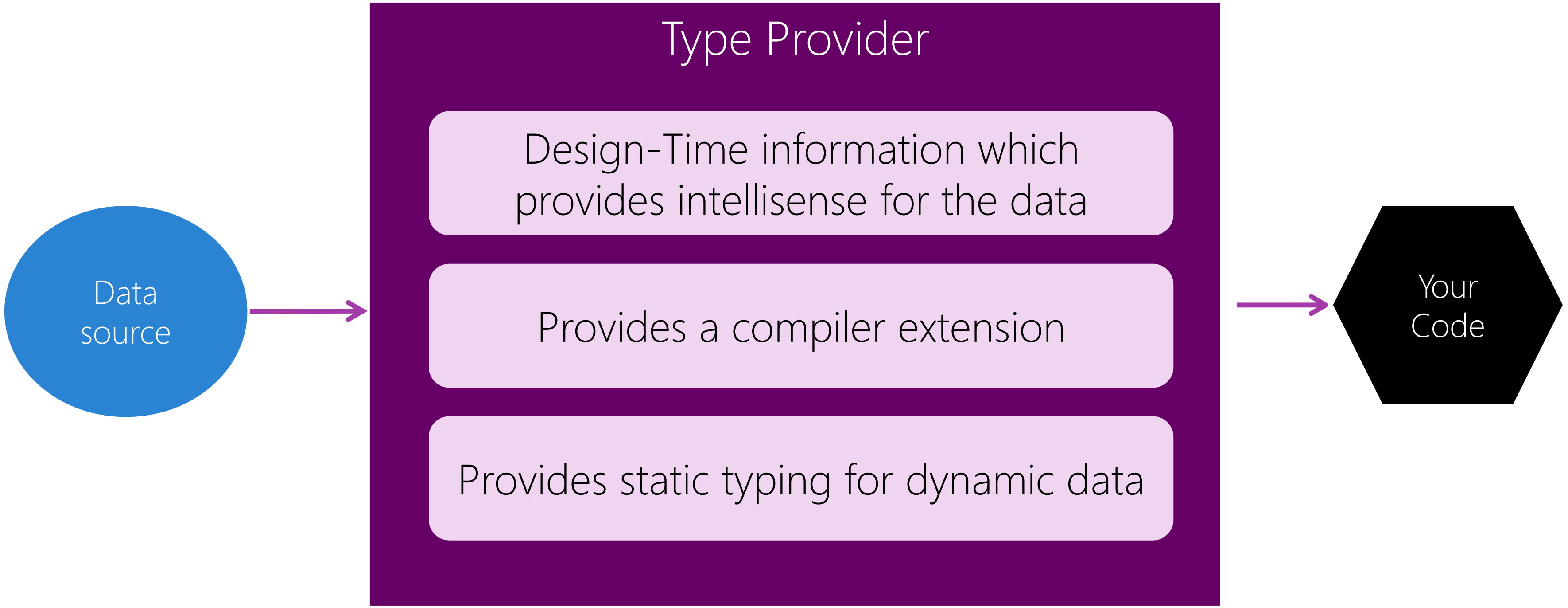
What is a type provider?

❖ F# Type Providers are an intelligent mechanism to bring in types from an external data source to your code in a standardized fashion



Xamarin University

What is a type provider?


Data source
Type Provider
Design-Time information which provides intellisense for the data
Provides a compiler extension
Provides static typing for dynamic data
Your Code

Xamarin University

Example type providers

- There are available type providers for all kinds of data – it's a thriving extensibility point for F#

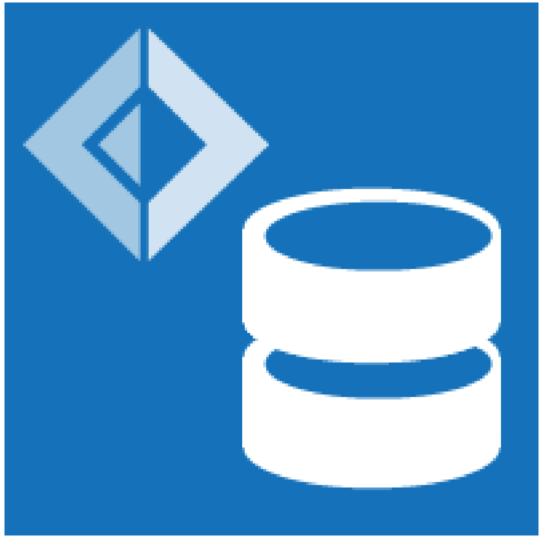
powershell	Azure	Choose your own adventure	Matlab	RSS	DBML	EDMX
SignalR	FunScript	R	Python	MS Dynamics CRM	World Bank	Regex
Twitter	CSV	JSON	XML	LINQ	IKVM	SQL Server
SQL Server with EF	XAML	Hadoop	WSDL	OData	Apiary	Facebook

Xamarin University

How does it work?

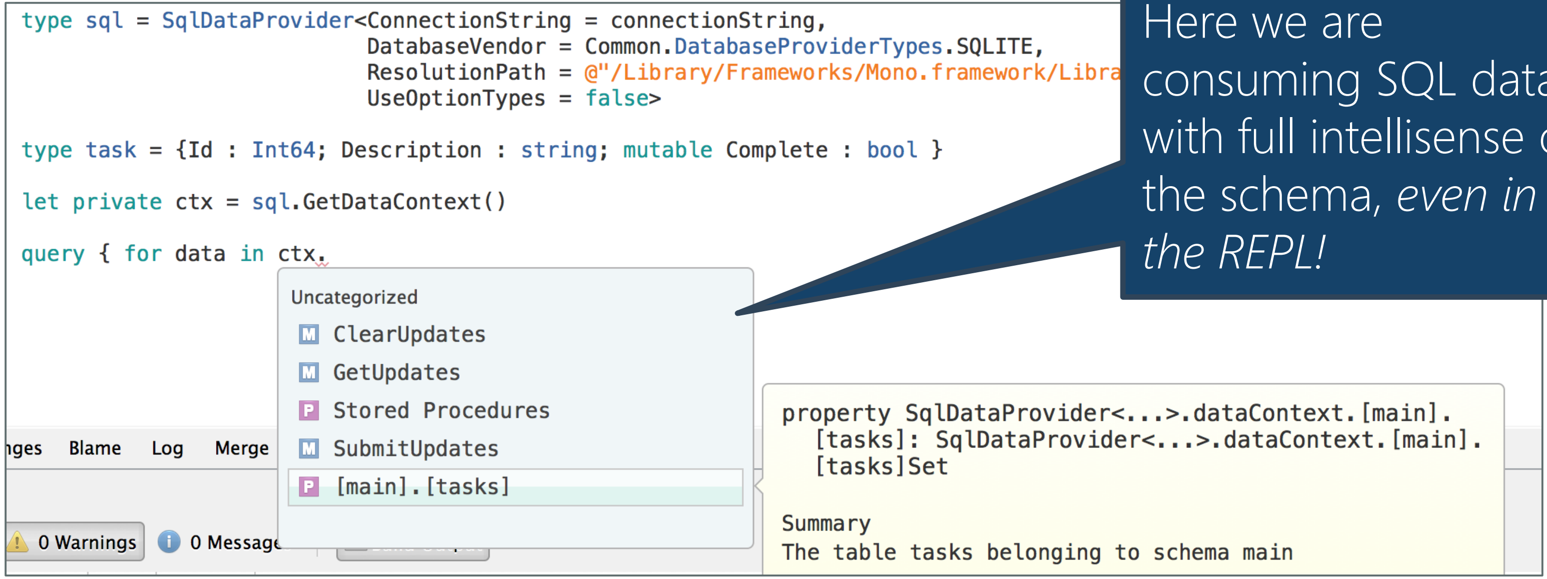
- Type providers are DLLs which extend the name resolution of the compiler and map data sources into the .NET type system using schema / metadata or provided sample information

- Supports both synthetic and generative providers



Tooling

❖ Type providers enable intellisense and tool tips on the data the type provider accesses



```
type sql = SqlDataProvider<ConnectionString = connectionString,
                           DatabaseVendor = Common.DatabaseProviderTypes.SQLITE,
                           ResolutionPath = @"/Library/Frameworks/Mono.framework/Libra
                           UseOptionTypes = false>

type task = {Id : Int64; Description : string; mutable Complete : bool }

let private ctx = sql.GetDataContext()

query { for data in ctx.
```

```
property SqlDataProvider<...>.dataContext.[main].
   [tasks]: SqlDataProvider<...>.dataContext.[main].
   [tasks]Set

Summary
The table tasks belonging to schema main
```

Here we are consuming SQL data, with full intellisense of the schema, *even in the REPL!*

Two major benefits

- Type providers offer two primary benefits, which distinguish F# from other programming languages

Simplicity

Scalability

Simplicity

- Type providers do not rely on code generation to create a data layer which results in fewer source files

- This provides a simpler file structure because we have less code, which makes it easier to understand and work with

Less Parsing

Inferred Types

Fewer files

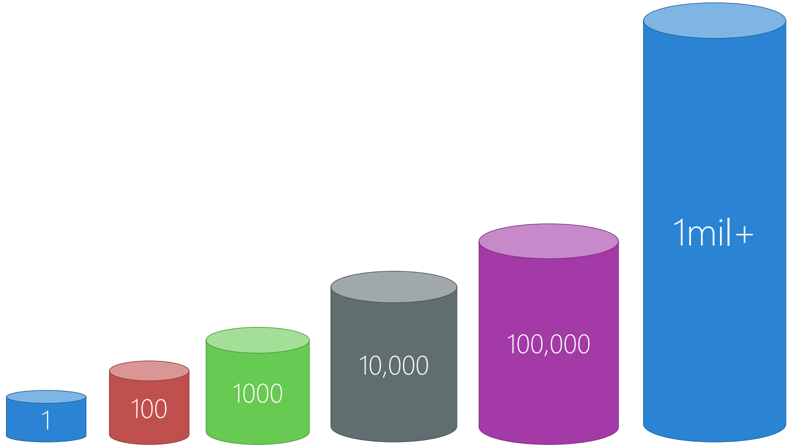
Less Coding

Xamarin University

Scalability

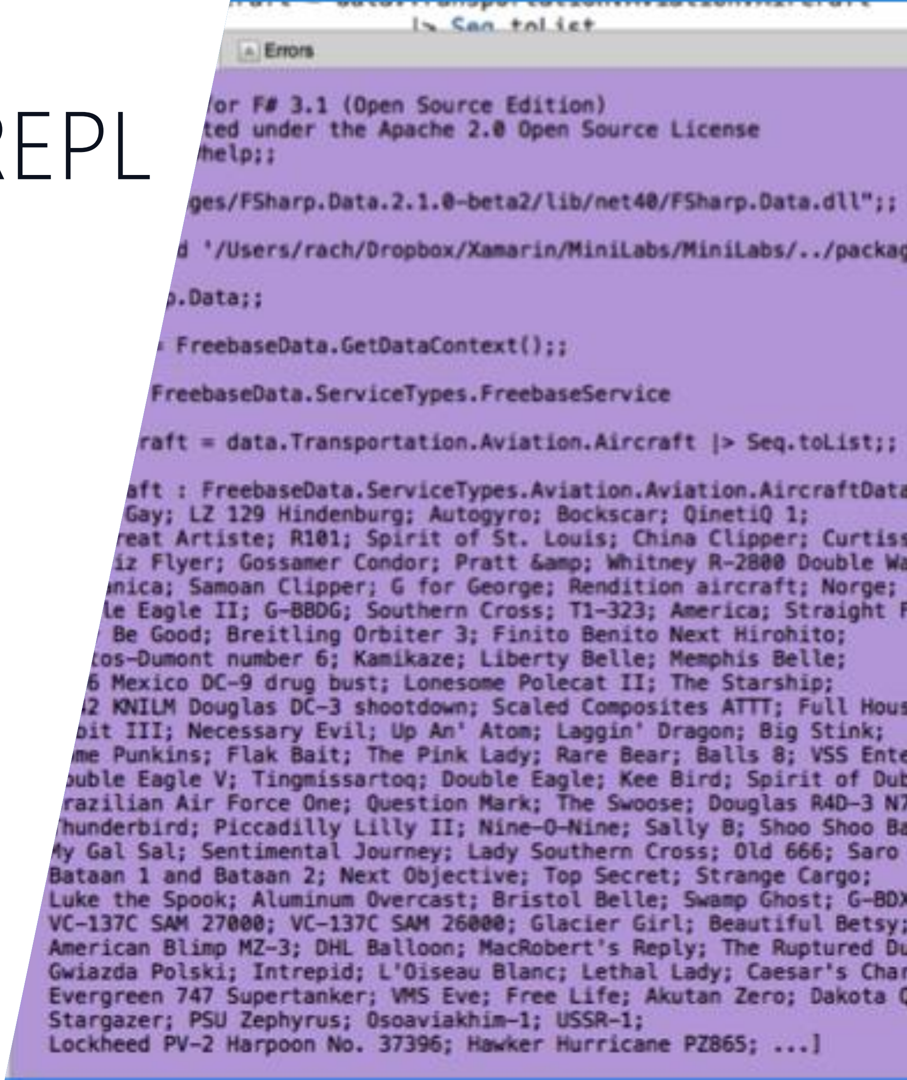
- Type providers return sequences – that means they do not create all objects at once, which increases scalability and performance

- World Bank has > 8000 types which can be processed



Exploring data in the REPL

- Since there is no code generation or compilation requirement, type providers can be used in the REPL and scripting environments



Errors

```
            |> Seq.toList
or F# 3.1 (Open Source Edition)
ted under the Apache 2.0 Open Source License
help;;

ges/FSharp.Data.2.1.0-beta2/lib/net40/FSharp.Data.dll";;

d '/Users/rach/Dropbox/Xamarin/MiniLabs/MiniLabs/../packag

.Data;;

 FreebaseData.GetDataContext();;

FreebaseData.ServiceTypes.FreebaseService

raft = data.Transportation.Aviation.Aircraft |> Seq.toList;;

aft : FreebaseData.ServiceTypes.Aviation.Aviation.AircraftData
Gay; LZ 129 Hindenburg; Autogyro; Bockscar; QinetiQ 1;
reat Artiste; R101; Spirit of St. Louis; China Clipper; Curtiss
iz Flyer; Gossamer Condor; Pratt &amp; Whitney R-2800 Double Wa
nica; Samoan Clipper; G for George; Rendition aircraft; Norge;
le Eagle II; G-BBDG; Southern Cross; T1-323; America; Straight F
 Be Good; Breitling Orbiter 3; Finito Benito Next Hirohito;
tos-Dumont number 6; Kamikaze; Liberty Belle; Memphis Belle;
6 Mexico DC-9 drug bust; Lonesome Polecat II; The Starship;
2 KNILM Douglas DC-3 shootdown; Scaled Composites ATTT; Full Hous
bit III; Necessary Evil; Up An' Atom; Laggin' Dragon; Big Stink;
me Punkins; Flak Bait; The Pink Lady; Rare Bear; Balls 8; VSS Ente
ouble Eagle V; Tingmissartoq; Double Eagle; Kee Bird; Spirit of Dub
razilian Air Force One; Question Mark; The Swoose; Douglas R4D-3 N7
hunderbird; Piccadilly Lilly II; Nine-O-Nine; Sally B; Shoo Shoo Ba
My Gal Sal; Sentimental Journey; Lady Southern Cross; Old 666; Saro
Bataan 1 and Bataan 2; Next Objective; Top Secret; Strange Cargo;
Luke the Spook; Aluminum Overcast; Bristol Belle; Swamp Ghost; G-BDX
VC-137C SAM 27000; VC-137C SAM 26000; Glacier Girl; Beautiful Betsy;
American Blimp MZ-3; DHL Balloon; MacRobert's Reply; The Ruptured Du
Gwiazda Polski; Intrepid; L'Oiseau Blanc; Lethal Lady; Caesar's Char
Evergreen 747 Supertanker; VMS Eve; Free Life; Akutan Zero; Dakota C
Stargazer; PSU Zephyrus; Osoaviakhim-1; USSR-1;
Lockheed PV-2 Harpoon No. 37396; Hawker Hurricane PZ865; ...]
```

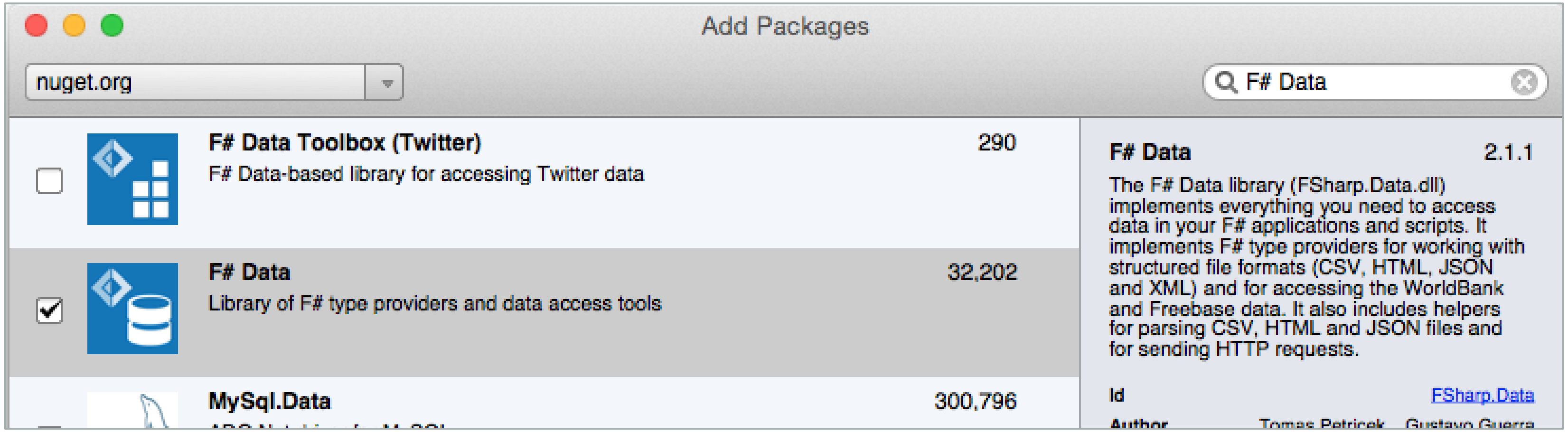
F# Data Library

- The F# data library contains the type providers for CSV, HTML, JSON and XML file formats. It also includes a World Bank provider.

Xamarin University

Using the F# Data type providers

- In order to use a type provider you will need a reference to the DLL – this can often be obtained through Nuget



F# Data Toolbox (Twitter) 290
F# Data-based library for accessing Twitter data

F# Data 32,202
Library of F# type providers and data access tools

MySql.Data 300,796

F# Data 2.1.1

The F# Data library (FSharp.Data.dll) implements everything you need to access data in your F# applications and scripts. It implements F# type providers for working with structured file formats (CSV, HTML, JSON and XML) and for accessing the WorldBank and Freebase data. It also includes helpers for parsing CSV, HTML and JSON files and for sending HTTP requests.

Id	FSharp.Data
Author	Tomas Petricek, Gustavo Guerra

Type providers in script files

- You can connect to type providers in script files using the #r directive, then must use the open directive to make the types accessible

```
#r "../packages/FSharp.Data.2.1.0/lib/net40/FSharp.Data.dll"

open FSharp.Data
open Fsharp.Net
```

Xamarin University

Initializing type providers

- Syntax for initializing all type providers follows the same general format.

```
type name = providerName<optionalParameters>
```

```
#r "../../FSharp.Data.SqlProvider.dll"
open FSharp.Data.Sql
type sql = SqlDataProvider<ConnectionString = "...",
        DatabaseVendor = Common.DatabaseProviderTypes.SQLITE,
        ResolutionPath = @"/Library/Frameworks/.../mono/4.5/",
        UseOptionTypes = false>
```

Parameters vary based on the providers needs and ability to infer the data shape

Xamarin University

Example: loading CSV files

- In many cases, the schema of the data is known, but for some such as the CSV provider, it must be inferred or supplied

```
#r "../packages/FSharp.Data.2.1.1/lib/net40/FSharp.Data.dll"
open FSharp.Data

type Stocks = CsvProvider<Sample = "data/sampleStockData.csv",
                         InferRows = 100,
                         Separators = ";">
```

Must pass in a file or URL which is then read to figure out the schema by reading the first 100 rows (can be altered with the **`InferRows`** property)

Xamarin University

Example: loading CSV files

- In many cases, the schema of the data is known, but for some such as the CSV provider, it must be inferred or supplied

```
type Stocks = CsvProvider<HasHeaders = false,
                    Schema = "Date(string),Open(float),
                             High,Close(float),Volume,Adj,
                             Close (float)">
```

Can also supply the schema directly in the form **`Name(Type)`** or **`Name`**, or just **`Type`** in which case it uses **`Column1`**…**`n`**

XML and JSON provide work much the same way

Xamarin University

Loading the data

- Once the type has been created, you must assign a value to the data representation – this is often done through a **specific load method**

```
type SqlConn = SqlDataProvider<ConnectionString, ...>
let ctx = SqlConn.GetDataContext()
```

```
[<Literal>]
let Url = "http://ichart.finance.yahoo.com/table.csv?s=APL"
type Stocks = CsvProvider<Url, InferRows=10>
let apple = Stocks.Load(Url)
```

Xamarin University

Working with the data

❖ The loaded data can then be accessed through typed properties, with full intellisense discovery and compile-time checking

```
let stocks = Stocks.Load(Url)

for s in stocks.Rows do
    printfn "%s Change: %f" s.Date (s.Close - s.Open)
```

Properties exposed match the metadata shape of the CSV file

Working with the data

- The loaded data can then be accessed through typed properties, with full intellisense discovery and compile-time checking

```
let ctx = SqlConn.GetDataContext()

for row in ctx.``[MAIN].[TASKS]`` do
   printfn "%s: %b" row.Title row.IsCompleted
```

Properties exposed match the schema of the table being queried, initiating the loop causes a **`SELECT`** to be issued

Flash Quiz

Flash Quiz

① Which phrase best describes a type provider?

a) An intelligent mechanism to bring in types from an external data source to your code

b) An expression which is evaluated on the final line of the function

c) Non-scalable database access

Flash Quiz

① Which phrase best describes a type provider?

a) An intelligent mechanism to bring in types from an external data source to your code

b) An expression which is evaluated on the final line of the function

c) Non-scalable database access

Flash Quiz

② Which command do you use to get access to a type provider in a script file?

a) #o

b) #r

c) #t

Flash Quiz

② Which command do you use to get access to a type provider in a script file?

a) #o

b) [✓ #r]

c) #t

Xamarin University

Flash Quiz

③ Type providers tend to be scalable because they ________

a) Are written in native, machine code

b) Are based on IEnumerable and designed to do lazy evaluation

c) Use parallel processing techniques and multiple cores

d) Are written in C#

Flash Quiz

③ Type providers tend to be scalable because they ________

a) Are written in native, machine code

b) Are based on IEnumerable and designed to do lazy evaluation

c) Use parallel processing techniques and multiple cores

d) Are written in C#

Individual Exercise

Accessing data from the JSON type provider

Xamarin University

Summary

1. Define type providers
2. Explore the benefits of type providers
3. Describe how to connect to type providers

Query and transform data from type providers

Xamarin University

Tasks

1. Query data sources
2. Explore query operators
3. Run queries

Query a data source

- Type providers return *sequences* which allow for mapping, filtering and pipelining to query and organize the data being returned

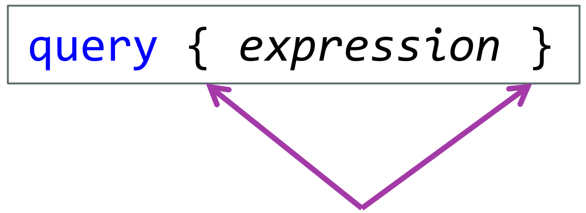
```
let wb = WorldBankData.GetDataContext()

let bigCountries =
    wb.Countries
    |> Seq.map (fun c -
> (c, c.Indicators.``Population, total``.[2012]))
    |> Seq.filter (fun (c,p) ->  p > 1000000.)
    |> Seq.toList
```

Get all the countries which had a population > 1M in 2012

Query Expressions

- Query expressions are a formalized F# language feature that allow you to filter, group and transform data from a sequence

- Provide support for LINQ in F#, almost identical features and syntax



```
query { expression }
```

The expression is contained within curly brackets{}

The query keyword

- The **query** keyword tells the compiler that you want to filter the data from the type provider, starts a LINQ query

```
let filteredIncomeList =
        query { for c in wb.Countries do ... }
```

Query expressions are one of the cases where loops are useful in F#, because the data is being pulled in as needed

The select keyword

- We use the **`select`** operator to identify what data will be returned in a query expression (projection)

```
let incomeList =
    query {
        for c in wb.Countries do
        select (c.Name, c.Indicators.``Income share held by lowest 10%``.[2010])
    }
```

return a tuple with the country name and income for the lowest 10% of the population in 2010

The where keyword

- We use the **where** operator to provide a filter expression on the data

```
let filteredIncomeList =
    query {
        for c in wb.Countries do
            where (not
                <| System.Double.IsNaN(
                     c.Indicators.``Income share held by lowest 10%``.[2010]))
            select (c.Name, c.Indicators.``Income share held by lowest 10%``.[2010])
    }
```

Only return countries which have the data we need – notice the **not** function being used here to reverse the condition

Xamarin University

Query operators

❖ Query operators allow you to identify what type of data you want, and how it should be returned

Operator	Description	Operator	Description
`contains`	Results include specific parameter	`groupBy`	Groups elements based on defined indicators
`count`	Returns the number of selected elements	`join`	Joins elements
`last`	Selects the last element	`averageBy`	Creates an average value for elements
`where`	Returns element based on specific criteria	`sortBy`	Sorts elements in ascending order

This is a partial list – check the MSDN documentation for a complete reference

Xamarin University

Using query operators

- Can combine operators together to generate the final result you want

```
let filteredCountries =
    query {
        for c in wb.Countries do
            where (c.Name.StartsWith("C") && c.Region.Contains("Europe"))
            sortByDescending c.Name
            select c
            skip 1
            take 5
    }
```

Running the query

- Once the data has been selected, we can pipe it into a sequencing operation to execute the query or provide further transformation

```
let filteredIncomeList =
    query {
        ...
    } |> Seq.toList

let MinIncomeShare = filteredIncomeList |> Seq.minBy snd
let MaxIncomeShare = filteredIncomeList |> Seq.maxBy snd
```

```
val MinIncomeShare : string * float = ("Lesotho", 0.98)
val MaxIncomeShare : string * float = ("Ukraine", 4.42)
```

Individual Exercise

Connecting to the World Bank type provider

Xamarin University

Flash Quiz

Flash Quiz

① Query expressions enable _________ in F#

a) SQL expressions

b) File I/O

c) Type providers

d) LINQ

Flash Quiz

① Query expressions enable ________ in F#

a) SQL expressions

b) File I/O

c) Type providers

d) <u>LINQ</u>

Xamarin University

Flash Quiz

② Which of the following is the correct way to write the function below:

```
A.|> Seq.filter (fun (x,y) -> not (System.Double.IsNaN(y)))
B.|> Seq.filter (fun (x,y) -> not <| System.Double.IsNaN(y))
```

a) A is the proper way to write the function

b) B is the proper way to write the function

c) Neither A or B is correct

d) Both are correct

Flash Quiz

② Which of the following is the correct way to write the function below:

```
A.|> Seq.filter (fun (x,y) -> not (System.Double.IsNaN(y)))
B.|> Seq.filter (fun (x,y) -> not <| System.Double.IsNaN(y))
```

a) A is the proper way to write the function

b) B is the proper way to write the function

c) Neither A or B is correct

d) <u>Both are correct</u>

Xamarin University

Flash Quiz

③ Which statement is true about query expressions?

a) Are triggered using the select keyword
b) Are an example of when loops are useful in F#
c) Allow you to filter but not transform data

Flash Quiz

③ Which statement is true about query expressions?

a) Are triggered using the select keyword

b) Are an example of when loops are useful in F#

c) Allow you to filter but not transform data

Summary

1. Query data sources
2. Explore query operators
3. Run queries

Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile