MATHEMA

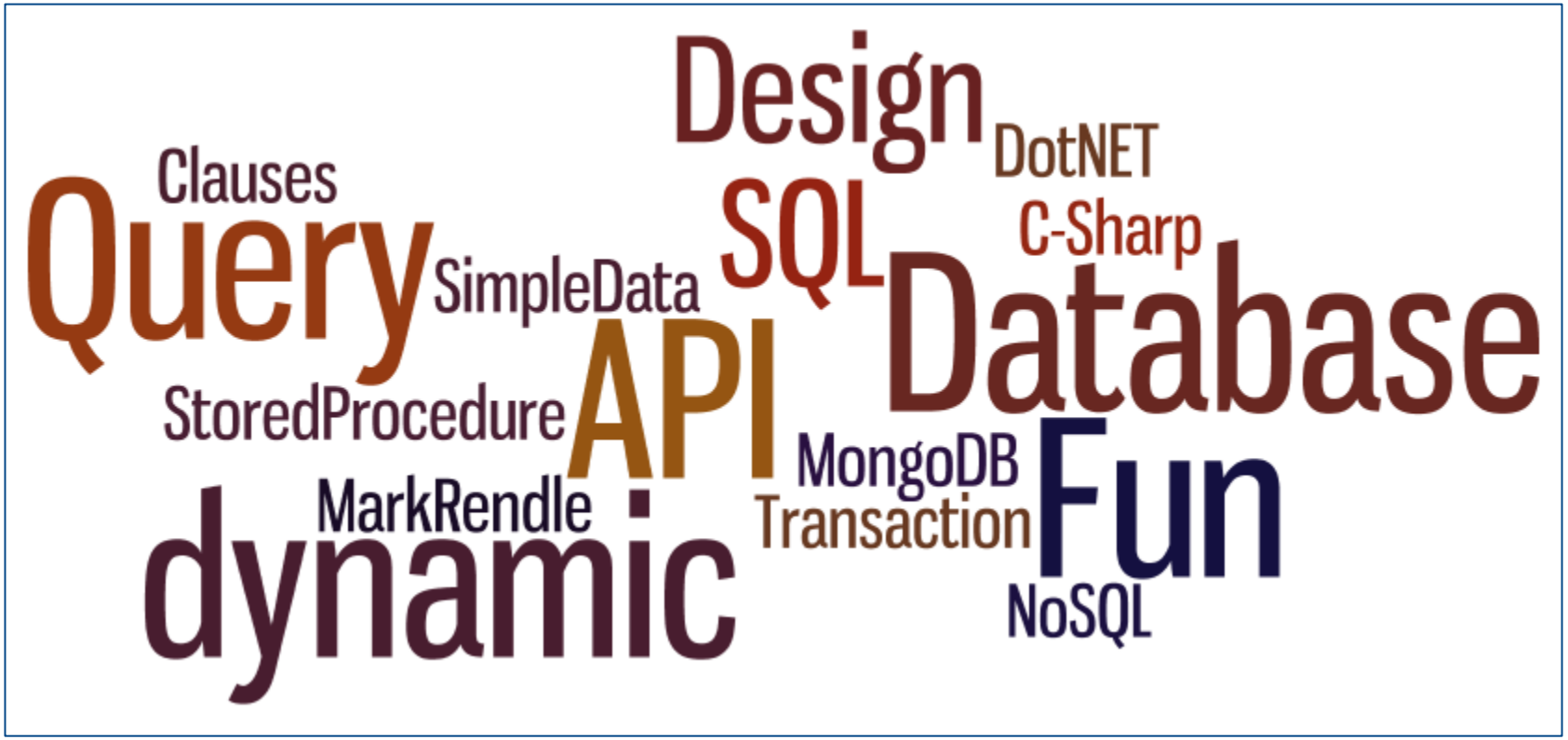
# Simple.Data

… an O/RM without O, R or M

Timothée Bourguignon

# What's on the menu?





# What is Simple.Data?



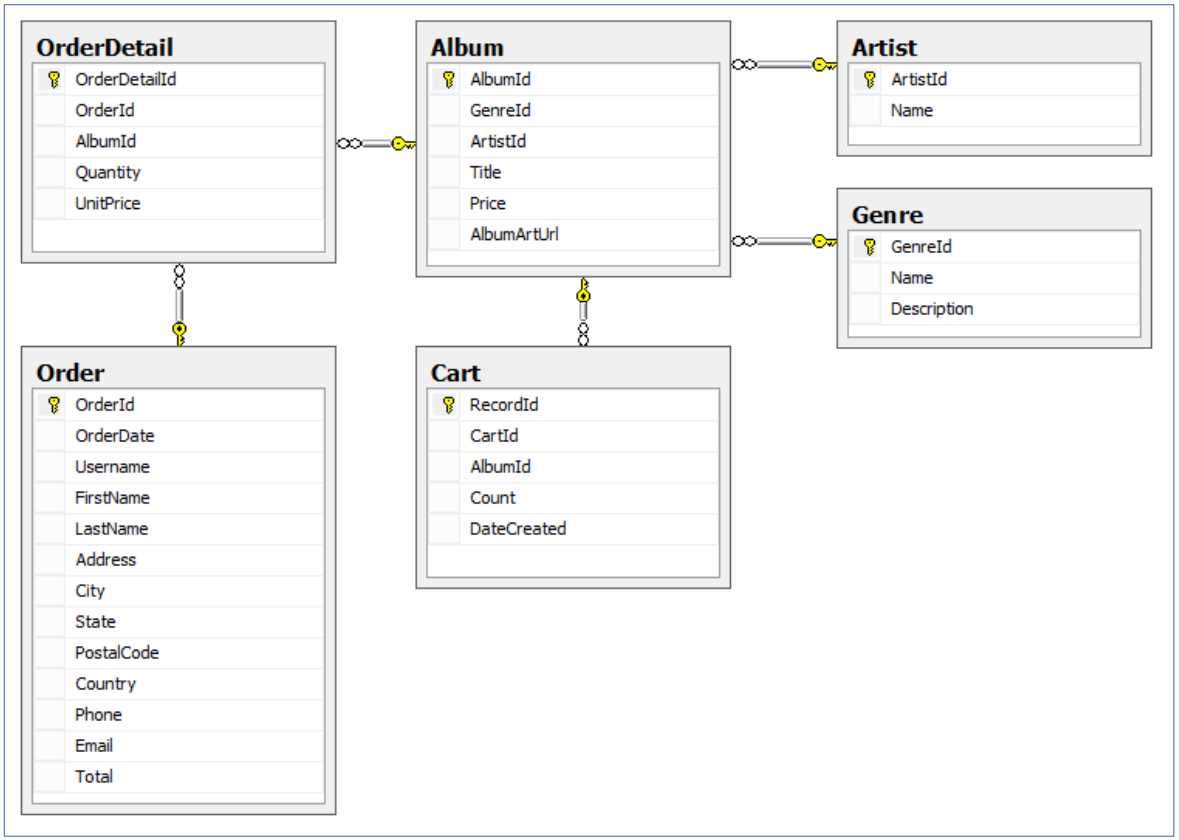
An O/RM without O, R or M!

# Hands-on!



## SQL Server + MvcMusicStore DB

- http://mvcmusicstore.codeplex.com/



# What is Simple.Data?



- Lightweight way of manipulating data
  - Based on .NET 4.0's „dynamic“ keyword
  - Interprets method and property names
  - Maps them to your underlying data-store
  - Prevents SQL Injection
  - Inspired by Ruby’s ActiveRecord and DataMappers
  - Open Source & runs on Mono
  - V1.0 rc3 released in Nov. 2012

# The real menu

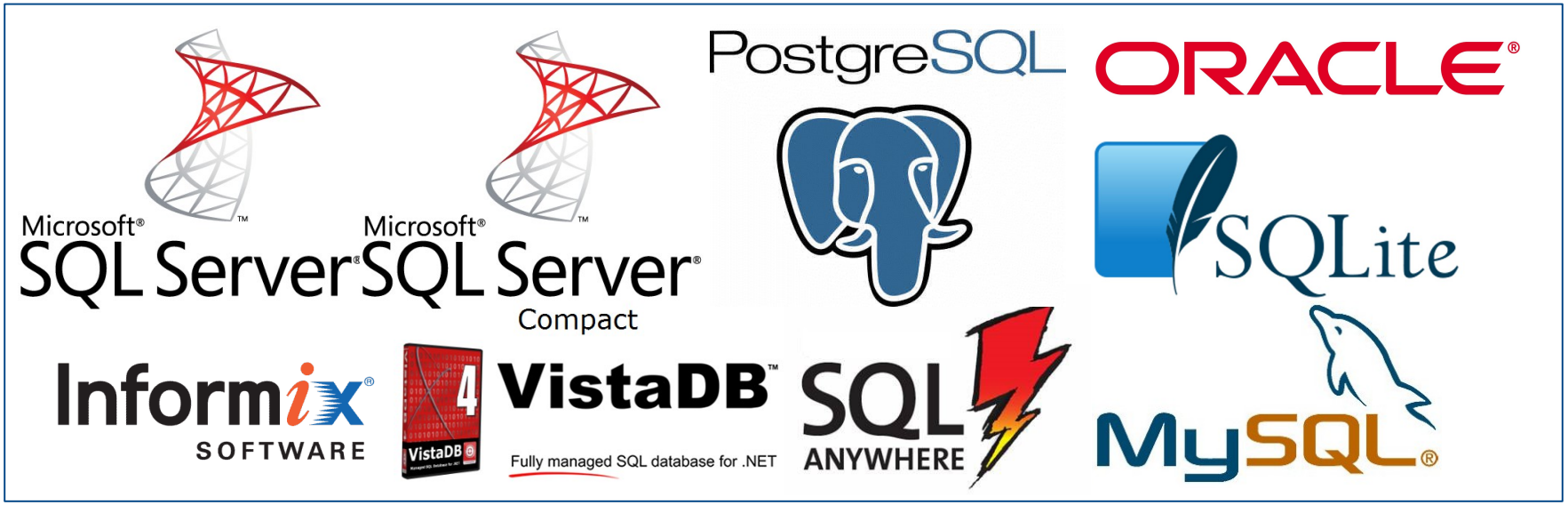


- Generalities
- Conventions
- CRUD Operations
- Objects Returned
- Joins & Evaluation Strategies
- Various Functions
- Tool & Testing
- Wrap-Up

- Hands-ons along the way

# Database agnostic





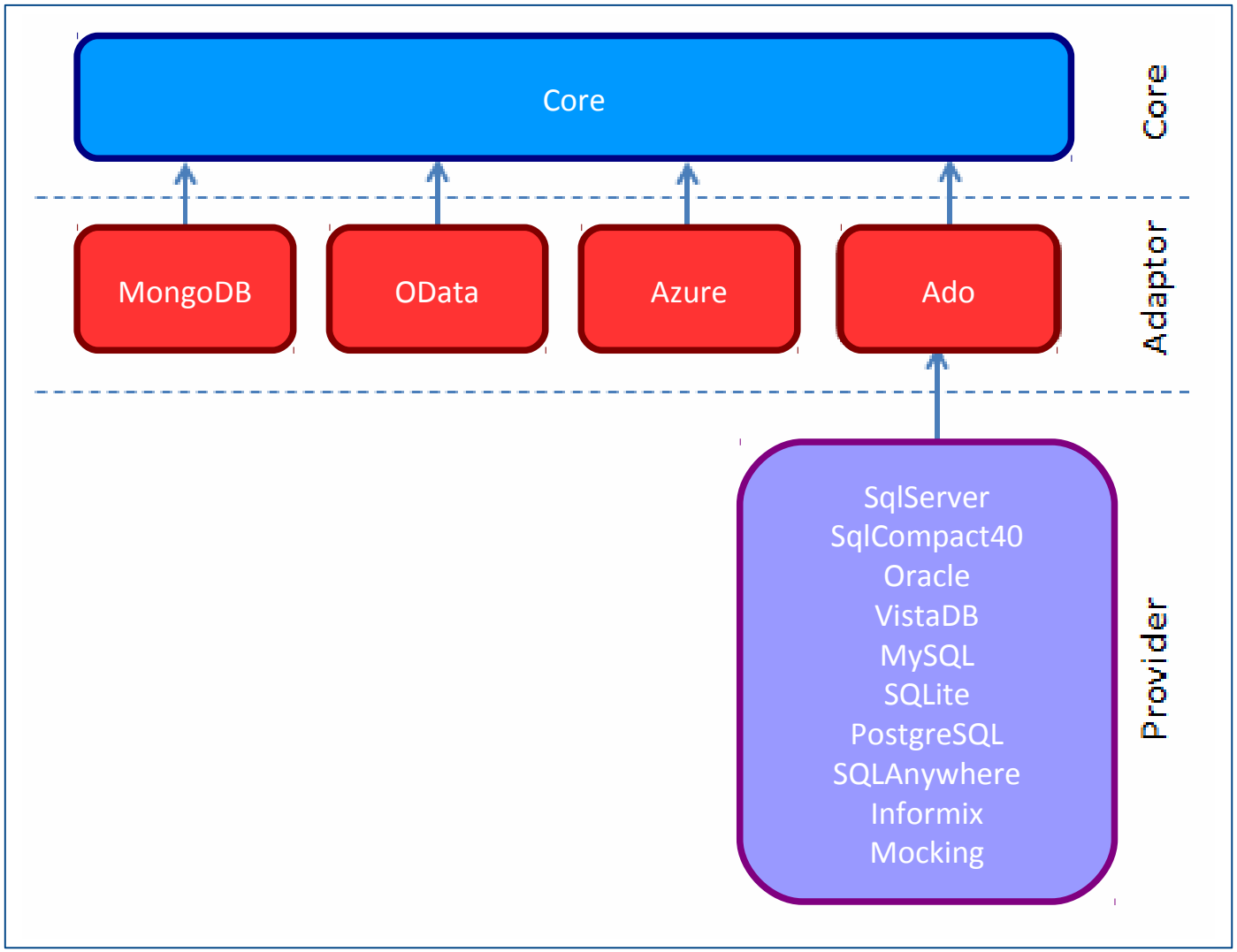

# Hands-on!



- MongoDB
  - + MvcMusicStore DB

# Package Architecture





# Nuget Packages



```
PM> Install-Package
   Simple.Data.SqlServer
   Simple.Data.Ado
   Simple.Data.SqlCompact40
   Simple.Data.Sqlite
   Simple.Data.MongoDB
   ...
```

# Conventions

# Opening a Connection



```
var magicDb = Database
                .OpenConnection("ConnectString");

var fileDb = Database
                .OpenConnection("MyDB.sdf");
```

- Per default connections are aggressively opened and closed for each query
- Supports shared connections

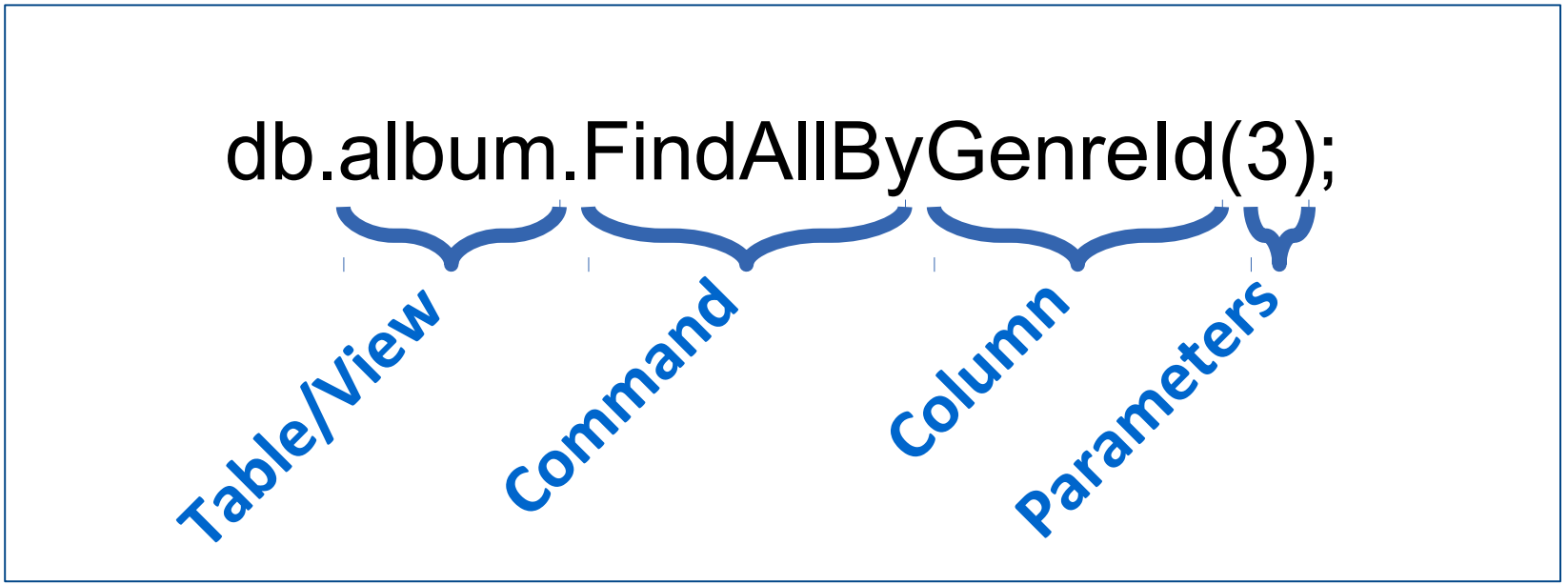
# Choose your weapon



- The „Fluid“ Way
  - Methods & properties convention-based mapping
- The „Indexer“ Way
  - Identification via an indexer syntax

# The Fluid Way





# The Indexer Way



- The problem

```
//Find by „Candy“ or find by „C and Y“?
db.sweets.FindAllByCAndY("Oreo");
```

- The solution

```
//Full indexer way
db["sweets"].FindAllBy(Candy: "Oreo");
//Hybrid fluid + indexer
db.sweets.FindAllBy(Candy: "Oreo");
db["city"].FindAllByCityName("Paris");
```

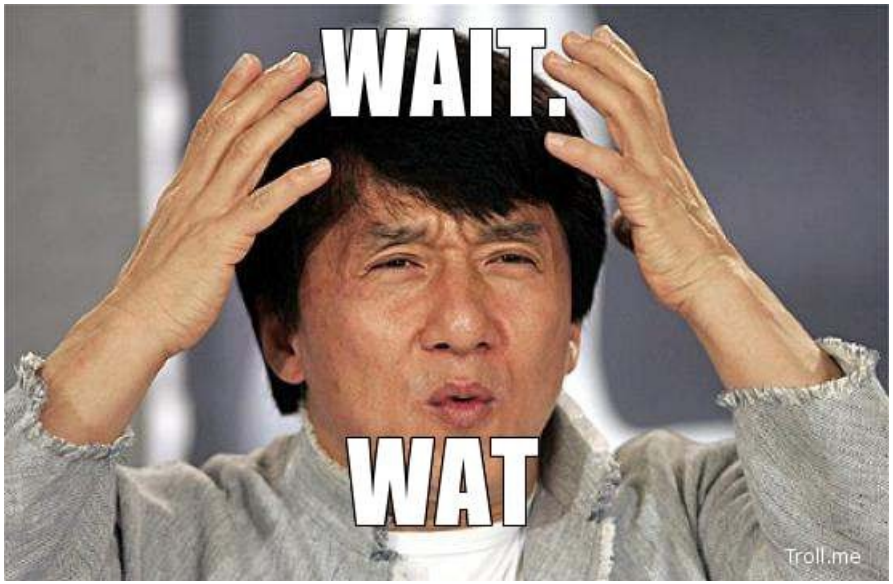
# Target Matching



- Sequence
  - Exact match
  - Case-insensitive - non-alphanumeric chars
  - Pluralized/Singularized version
- The following are thus all identical
  - Albums.GenreId
  - Album.GenreId
  - ALBUMS.GENREID
  - ALBUM.GENREID
  - [ALBUMS].[GENREID]
  - [ALBUM].[GENREID]
  - AlBuM.geNReId
  - Al__*bum.Genr-eId
  - ...

# No IntelliSence






- Dynamics => no member / function inferrence
- Schema analyse planned for Simple.Data v2
- Tool: Simple.Data.Pad
- Still easy to get used to

# CRUD OPERATIONS

# Create



- Insert(object or named parameters)

# Read



- Read
  - All()
  - Find(simple expressions)
  - Get(primary key)
  - FindAll(optional condition)
  - FindAllByXXX(parameter)

# Update



- Update(object or named parameters)
  - Update
  - UpdateByXXX
  - UpdateAll + optional condition
- Upsert e.g. Update or Insert
- Some kind of optimistic locking
  - Update(modifiedObject, originalObject)
  - Fails if the column(s) you are modifying changed
    - Nota: does not work with Upsert

# Delete



- Delete
  - Delete(object or named parameters)
  - DeleteByXXX(parameters)
  - DeleteAll(optional conditions)

# Hands-on!



- CRUD Operations
  - Insert
  - Update
  - Delete
  - Read

# Objects Returned

# SimpleRecord



- Dynamic object
- Contains a property for each of the columns requested whose values are those of the single row retrieved from the data store
- „Cast-able“ to a concrete implementation

# SimpleQuery



- Dynamic object
- Similar to LINQ structure
- Executes when enumerated
- Contains a SimpleRecord object for each row returned

# Casting



- Casting to objects
  - Implicit
  - Explicit: Cast<T>, ToList, ToList<T>, ToArray, ToArray<T>

- Hands-On
  - Implicit casting
  - Explicit casting

# Joins
# & Evaluation Strategies

# Hands-on!

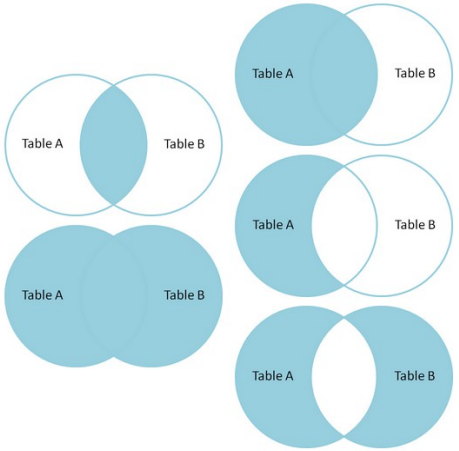


- Lazy natural evaluation
- Casting + Lazy?
- Eager evaluation

# Joins



- Lazy loading
  - „Natural Joins“ e.g. „Table chaining“




- Eager Loading
  - „With Joins“
    - With/WithXXX } Foreign-Key relationship present
    - WithOne
    - WithMany } No Foreign-Key relationship necessary (no referential integrity)
  - „Explicit Joins“
    - Join
    - LeftJoin
    - OuterJoin } Natural joins can be used as part of an explicit join, the join is then eager loaded

# Hands-on!



- Eager Joins
  - Select + Natural Joins + As
  - With

# Various Functions

# Ordering Results



- OrderBy, OrderByDescending
- ThenBy, ThenByDescending

```
db.Albums.All().OrderByGenreId()
                .ThenByArtistIdDescending();
```

# Scalar Queries



- GetCount
- GetCountBy
- Exists, Any
- ExistsBy, AnyBy

```
int albumCount = db.Albums.GetCount(
                              db.Albums.GenreId == 2);
```

# Query Modifiers



- Select
  - Star & AllColumns

```
db.Albums.All().Select(db.Albums.Title,
                       db.Albums.ArtistId);
```

# Query Modifiers



- Column Aliasing: As(string)

```
var albums = db.Albums.All().Select(
                        db.Albums.AlbumId,
                        db.Albums.Title.As("AlbumName"));
```

# Query Modifiers



- Where clauses
  - Operators (+, -, *, /, %)
  - IN, BETWEEN, LIKE, IS NULL

```
var albums = db.Albums.FindAllByGenreId(1)
        .Select(db.Albums.Title)
        .Where(db.Albums.Price < 8);

db.Albums.All().Where(
        db.Albums.Title.Like("%Side Of The%"));
```

# Aggregate Functions



- Grouping and Aggregates
  - Having → Group By / Having
  - Min, Max, Avg, Sum

```
var cheapAlbums = db.Albums.All()
        .Having(db.Albums.Price < 9).ToList();

var totalCost = db.Albums.All().Select(
        db.Albums.Price.Sum().As("TotalCost"));
```

- Like a function...

```
CREATE PROCEDURE ProcedureWithParameters
@One VARCHAR(MAX),
@Two VARCHAR(MAX)
AS

SELECT *  FROM Customers
WHERE Firstname = @One and Lastname like @Two
```

```
db.ProcedureWithParameters(1, 2);
```

- Wrap up the calls

```
using (var transaction = db.BeginTransaction())
{
        transaction.albums.Insert(GenreId: 1...);
        transaction.Commit();
}
```

# Tool & Testing

# Tool: Simple.Data.Pad



- Similar to LINQ-Pad... kind of...
  - https://github.com/markrendle/Simple.Data.Pad


Simple.Data.Pad 0.16.2

Connection

OpenConnection ( Data Source=.\SQLEXPRESS;Initial Catalog=world;Integrated Security=True )

Run

```
db.city.FindAllByCountryCode("USA").Where(db.city.Population > 1000000);
```

| ID | Name | CountryCode | District | Population |
|---|---|---|---|---|
| 3793 | New York | USA | New York | 8008278 |
| 3794 | Los Angeles | USA | California | 3694820 |
| 3795 | Chicago | USA | Illinois | 2896016 |
| 3796 | Houston | USA | Texas | 1953631 |
| 3797 | Philadelphia | USA | Pennsylvania | 1517550 |
| 3798 | Phoenix | USA | Arizona | 1321045 |
| 3799 | San Diego | USA | California | 1223400 |
| 3800 | Dallas | USA | Texas | 1188580 |
| 3801 | San Antonio | USA | Texas | 1144646 |

Results Trace

```
Simple.Data.Ado:
Text
select [dbo].[city].[ID],[dbo].[city].
[Name],[dbo].[city].[CountryCode],[dbo].
[city].[District],[dbo].[city].
[Population] from [dbo].[city] WHERE
([dbo].[city].[CountryCode] = @p1 AND
[dbo].[city].[Population] > @p2)
@p1 (AnsiStringFixedLength) = USA
@p2 (Int32) = 1000000
```

Results Trace

# Testing: InMemoryAdapter



```
[Test]
public void Should_do_something()
{
        var adapter = new InMemoryAdapter();
        Database.UseMockAdapter(adapter);
        var db = Database.Open();
        db.Test.Insert(Id: 1, Name: "Alice");
        //...
}
```

- The InMemoryAdapter supports
  - Joins, Transactions, Stored procedures...

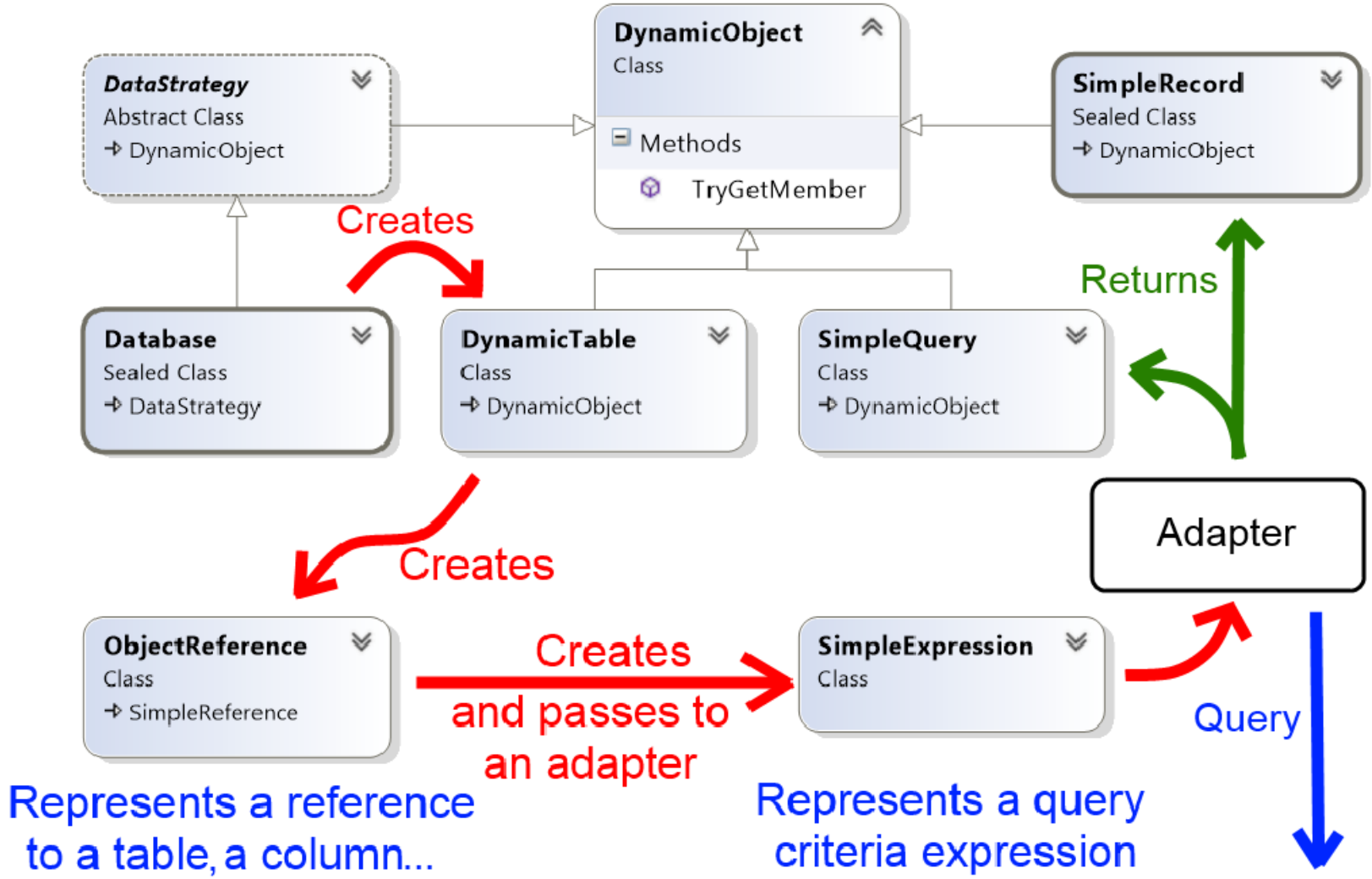
# InMemoryAdapter Configuration



- Tweaking functions
  - SetKeyColumn
  - SetAutoIncrementColumn
  - AddFunction (stored procedure)
  - ConfigureJoin
  - ...

# Design




DynamicObject
Class
Methods
TryGetMember
DataStrategy
Abstract Class
DynamicObject
SimpleRecord
Sealed Class
DynamicObject
Creates
Returns
Database
Sealed Class
DataStrategy
DynamicTable
Class
DynamicObject
SimpleQuery
Class
DynamicObject
Adapter
Creates
ObjectReference
Class
SimpleReference
Creates
and passes to
an adapter
SimpleExpression
Class
Query
Represents a reference
to a table, a column...
Represents a query
criteria expression

# Wrap-Up

# Wrap up



- OpenSource, Mono
- Everything is dynamic
- Fluid-, Indexer Way
- CRUD
  - FindXXX, DeleteXXX, UpdateXXX etc.
- Dynamics Objects Returned
- Joins, lazy, eager
  - Natural, WithXXX, Join
- Various Functions
  - Group, Order, Scalar, Modifiers etc.
- Tool & Testing
- Design

# Simple.Data in Short



- Lightweight
- Readable
- Compelling
- Fun to use
- Interesing design

- Dynamics extensive testing
- Good understanding upfront

- My Recommendation
  - Try it and study it
  - Take it for a spin for some tooling and/or prototyping
  - ...and some projects?

# Further Reading



- Github
  - https://github.com/markrendle/Simple.Data
- Nuget
  - http://nuget.org/packages?q=simple.data
- GoogleGroup
  - https://groups.google.com/forum/?fromgroups#!forum/simpledata
- Framework Author: Mark Rendle
  - @MarkRendle
  - http://blog.markrendle.net/

# Fragen?

tim.bourguignon@mathema.de
about.me/timbourguignon

# MATHEMA


Eine kleine Stärkung gibt es jetzt am MATHEMA-Stand!