



# “Housekeeping”

- Welcome to today’s ACM Webinar. The presentation starts at the top of the hour.
- If you are experiencing any problems/issues, refresh your console by pressing the **F5** key on your keyboard in **Windows**, **Command + R** if on a **Mac**, or refresh your browser if you’re on a mobile device; or close and re-launch the presentation. You can also view the Webcast Help Guide, by clicking on the “Help” widget in the bottom dock.
- To control volume, adjust the master volume on your computer.
- If you think of a question during the presentation, please type it into the **Q&A** box and click on the submit button. You do not need to wait until the end of the presentation to begin submitting questions.
- At the end of the presentation, you’ll see a **survey** open in your browser. Please take a minute to fill it out to help us improve your next webinar experience.
- You can download a copy of these slides by clicking on the **Resources** widget in the bottom dock.
- This presentation is being recorded and will be available for on-demand viewing in the next 1-2 days. You will receive an **automatic e-mail notification** when the recording is ready.



# Data Access Using Entity Framework

Terry Coatta  
CTO Marine Learning Systems  
December 3, 2014



Association for  
Computing Machinery

*Advancing Computing as a Science & Profession*



# ACM Learning Center

<http://learning.acm.org>

- 1,400+ trusted technical books and videos by leading publishers including O' Reilly, Morgan Kaufmann, others
- Online courses with assessments and certification-track mentoring, member discounts on tuition at partner institutions
- Learning Webinars on big topics (Cloud/Mobile Development, Cybersecurity, Big Data, Recommender Systems, SaaS, Agile, Machine Learning, NLP, Hadoop, Parallel Programming, etc.)
- ACM Tech Packs on top current computing topics: Annotated Bibliographies compiled by subject experts
- Popular video tutorials/keynotes from ACM Digital Library, A.M. Turing Centenary talks/panels
- Podcasts with industry leaders/award winners



Association for  
Computing Machinery

*Advancing Computing as a Science & Profession*



# “Housekeeping”

- If you are experiencing any problems/issues, refresh your console by pressing the **F5** key on your keyboard in **Windows**, **Command + R** if on a **Mac**, or refresh your browser if you’re on a mobile device; or close and re-launch the presentation. You can also view the Webcast Help Guide, by clicking on the “Help” widget in the bottom dock.
- To control volume, adjust the master volume on your computer.
- If you think of a question during the presentation, please type it into the **Q&A** box and click on the submit button. You do not need to wait until the end of the presentation to begin submitting questions.
- At the end of the presentation, you’ll see a **survey** open in your browser. Please take a minute to fill it out to help us improve your next webinar experience.
- You can download a copy of these slides by clicking on the **Resources** widget in the bottom dock.
- This presentation is being recorded and will be available for on-demand viewing in the next 1-2 days. You will receive an **automatic e-mail notification** when the recording is ready.





# Talk Back

- Use the **Facebook** widget in the bottom panel to share this presentation with friends and colleagues
- Use **Twitter** widget to Tweet your favorite quotes from today's presentation with hashtag **#ACMWebinarEF**
- Submit questions and comments via Twitter to **@acmeducation** – we're reading them!



# Overview

Two major components for today's presentation:

- When is it appropriate to use Entity Framework?
  - Data comes in lots of sizes, shapes, and flavours
  - Need to choose the right tools for the job
- Best practices for using Entity Framework
  - Usage patterns (unit of work, repository)
  - Testing
  - Performance

# What is Entity Framework?

	Id	UserName	FirstName	LastName	EmailAddress
1	FF38DA07-F66C-11E3-80BD-0CC47A025D51	NTest	New User1	Test	newuser123@gmail.com
2	28883384-585A-11E3-8354-9651563C3E58	TTest1	Existing User	test	1@12.com
3	CE7BB6EB-588B-11E3-8354-9651563C3E58	TTest2	test	test	NULL
4	C9C9BDE8-588D-11E3-8354-9651563C3E58	TTest11	test	test1	NULL
5	6AFB7BFB-4890-11E3-8354-9651563C3E58	TTest	test	test	NULL

```
public class User
{
    public Guid Id;
    public string UserName;
    public string FirstName;
    public string LastName;
    public string EmailAddress;
}
```



# Background Info

- There are a number of ORMs available, but the most popular are:
  - Hibernate (Java)
  - NHibernate (Windows/C#)
  - Entity Framework (Windows/C#)
- <http://hibernate.org/orm/documentation/>
- <http://msdn.microsoft.com/en-us/data/ee712907.aspx>
- <http://blog.oneunicorn.com/category/entity-framework/>



# No Silver Bullet

- EF/SQL are a useful choice in some scenarios
- **Important:** Must be able to identify those scenarios!
- Considerations:
  - Volume of data
  - Arrival rate
  - Write once vs update
  - Structure (flat vs relational)
  - Consistency requirements
  - Scaling requirements
  - Time to market



# When To Use Entity Framework

- EF can be applied in a variety of circumstances, but the sweet spot is probably:
  - Data with significant relational structure
  - Strong consistency needed
  - Arrival rate low to moderate
  - Data can be partitioned into reasonably independent groups whose size is small to medium



# Data Structure (tightly coupled)

```
public class Course
{
    public List<Instructor> Instructors;
    public List<Student> Students;
    public List<ExamDefinition> Exams;
    public List<ExamResult> ExamResults;
}
```

```
public class ExamDefinition
{
    public Course BelongsTo;
    public List<Question> Questions;
}
```

```
public class ExamResult
{
    public Student TakenBy;
    public Course BelongsTo;
    public ExamDefinition DefinedBy;
    public List<Answer> Answers;
}
```



# Data Structure (loosely coupled)

```
public class User
{
    public Guid Id;
    public string Name;
    public int Age;
}

public class UserAudit
{
    public Guid Id;
    public Guid AuditedId; // Not a foreign key
    public List<UserAuditState> Changes;
}

public class UserAuditState
{
    public DateTime ChangedAt;
    public string Name;
    public int Age;
}
```



# Time To Market Considerations

Using a third party framework like Entity Framework will usually speed up delivery of an application if:

- You would end up implementing similar functionality yourself (e.g. query caching, in memory representation of data, memory-based navigation between objects)
- The learning curve for the framework is not too steep (i.e. you are using it in a way that its designers anticipated)
- You don't run into correctness/performance issues that are hard to understand (i.e. all frameworks hide things)



# Is EF Scalable?

- Sort of... you can build a big system out of smaller pieces but you won't define it as a single unified data model in EF
- At some point you will need to scale to multiple DB's
  - 2-phase commits are slow
  - CAP theorem limits what can be achieved
  - So you can't really have tightly coupled data spread over multiple databases
- Carefully choose where you want to deal with inconsistency
  - Location/customer/function based sharding
  - CDN for read-only or low consistency data



# Enterprise Applications - A Good Fit?

There is a large class of enterprise applications with:

- Rich data structure
- More limited scaling requirements

A viable architecture is functional sharding combined with geo/horizontal sharding

- Divide objects into clusters based on relationship cohesion / consistency
- Build services around each
- Use location/customer/functional sharding to scale
- Still able to use transactions within each service



# Using Entity Framework

- The remainder of the talk will look at:
  - Building a data access layer with EF
    - Unit of work
    - Repository
    - Testing/Dependency Injection
  - Performance Issues
    - Change tracking
    - Bulk inserts
    - Bulk retrieval (for reporting)
    - Cache tables (i.e. views)

# Building a Data Access Layer

Microsoft examples are not good guidance 😊

So, next we'll cover:

- Need a 'unit of work' abstraction
- Need a repository abstraction
- Working with LINQ
- Interaction of 'dependency injection' and EF



# Data Access Layer

A unit of work is essentially a transaction.

```
using (var uow = new UOW(UnitOfWorkTypes.ReadOnly))
{
    /* ... work with objects */
}
```

A repository provides access to one type of object:

```
var userRepo = uow.GetRepo<User>();
var myUser = userRepo.All().FirstOrDefault(u => u.Name == "Terry");
if (myUser == null) { /* ... didn't find it ... */ }
else { /* ... access MyUser ... */ }
```



# The Joy of LINQ

Queries that are: type-safe and composable

```
IQueryable<User> ActiveUsers()  
{  
    ... get repo ...  
    var users = UserRepo.All();  
    var activeUsers = user.Where(u => u.Active);  
    return activeUsers;  
}
```

```
void SomeMethod()  
{  
    var activeUsers = ActiveUsers();  
    var youngActiveUsers = activeUsers.Where(u => u.Age < 10);  
}
```



# Entity Framework and Interfaces

- Entity Framework doesn't like interfaces
  - Objects are mapped to the DB via a concrete class
  - LINQ queries must use this class
  - Makes it hard to hide objects behind an interface
  - Makes unit testing challenging (can't easily stub objects)
  - But you can stub the unit of work and repositories
- 
- Entity framework also doesn't like dependency injection
  - It allocates objects directly when they are realized
  - Possible to fake with `ObjectMaterialized` handler



# Queries for Modified Objects

- Queries are always executed against the DB
- So, if you modify objects in a unit of work and then query for them with those modified values, you won't find them
- This can be fixed by explicitly looking in EF's cache of objects



# FindObject()

```
T FindObject<T>(Expression<Func<T, bool>> predicate) where T : class
{
    var addedObjs = GetChangedEntitiesByState<T>(EntityState.Added);
    var modifiedObjs = GetChangedEntitiesByState<T>(EntityState.Modified);
    var unmodifiedObjs = GetChangedEntitiesByState<T>(EntityState.Unchanged);

    Func<T, bool> predCompiled = predicate.Compile();

    IEnumerable<T> foundObjects = addedObjs.Where(predCompiled);
    int itemCount = foundObjects.Count();
    if (itemCount == 1) { return foundObjects.First(); }

    // ... now check in modified objects
    // ... now check in unmodified objects (in DB)
}
```



# Performance Issues (Change Tracking)

- What is change tracking?
  - EF needs to know what objects to write back to the DB
- Change tracking via API calls (auto detect changes)
  - $O(n^2)$  performance if you create new objects
  - Problematic for bulk inserts
- Change tracking via proxies
  - Proxy knows when object changed
  - Carries out 'fix-up' when navigation properties change
  - Expense happens on every access to navigation properties
- Which is better?
  - Depends on whether you touch many objects a little or a few objects a lot



# Performance Issues (Bulk Inserts)

For bulk inserts two techniques can make orders of magnitude differences:

1. Turn off autodetectchanges
2. Assign navigation properties via foreign key

Item (2) is odd. In EF, we can assign a related object:

- Directly via a navigation property (`user.Owner = me`)
- Via a foreign key (`user.OwnerId = me.Id`)

The latter is significantly faster!



# Performance Issues (Reporting)

It can be challenging to get EF to query efficiently for bulk data sets that aggregate data across tables.

- Materialization costs
- Deeply nested data structures and aggregates
- Using views/caches to improve performance

The goal is to try and ensure that the data set is computed in the DB rather than in memory. This is particularly significant when returning 'paged' result sets.

# Performance Issues (materialization)

```
Dictionary<Student, float> grades = new Dictionary<Student, float>();

foreach (var s in allStudents)
{
    grades[s] = s.ExamResults.Average(e =>
        (float) e.Answers.Sum(a => a.Score) /
        e.Answers.Sum(a => a.AnswerTo.MaxScore));
}

return grades.OrderBy(kvp => kvp.Value)
    .Skip(50)
    .Take(25)
    .ToDictionary(kvp => kvp.Key);
```



# Performance Issues (materialization)

```
var grades = allStudents.Select(s => new
{
    student = s,
    grade = s.ExamResults.Average(e =>
        (float) e.Answers.Sum(a => a.Score) /
        e.Answers.Sum(a => a.AnswerTo.MaxScore))
});

return grades.OrderBy(g => g.grade)
    .Skip(50)
    .Take(25)
    .ToDictionary(g => g.student, g => g.grade);
```



# Performance Issues (join)

```
var studentResults = Students.Join(ExamResults,  
    s => s.Id, e => e.fkTakenBy,  
    (s, e) => new { Student = s, ExamResult = e });
```

```
var groupedResults = studentResults.GroupBy(sr =>  
    new { sid = sr.Student.Id, eid = sr.ExamResult.Id });
```

```
var studentsWithCount = groupedResults.Select(g => new  
    {  
        student = g.FirstOrDefault().Student,  
        examCount = g.Count()  
    });
```



# Performance Issues (other)

1. Map EF objects to SQL views
2. Use a secondary EF type that contains 'pre-flattened' results. This is effectively like using a SQL view, except that there are no limitations on the 'shape' of the view (e.g. not being able to use Min, Max, etc)
3. Don't use Any(), use Count() > 0



# Questions?

You can also follow up with me by email:

[coatta@acm.org](mailto:coatta@acm.org)



Association for  
Computing Machinery

*Advancing Computing as a Science & Profession*



# ACM: The Learning Continues...

- Questions about this webcast? [learning@acm.org](mailto:learning@acm.org)
- ACM Learning Webinars (on-demand archive): <http://learning.acm.org/webinar>
- ACM Learning Center: <http://learning.acm.org>
- ACM Queue: <http://queue.acm.org>

