SYNERGY
2018
DEVPARTNER
CONFERENCE
OCT 8-12 NEW ORLEANS, LA

EF Core

Presented by Jeff Greene

What Is It?

- “EF Core is an object-relational mapper (O/RM) that enables .NET developers to work with a database using .NET objects. It eliminates the need for most of the data-access code that developers usually need to write.”
- Open source from Microsoft
- Completely rewritten from EF 6
- EF Core 2.1 released May 30, 2018

Why did we make a provider for SDBMS?

- To support tooling that layers on top of EF Core
  - Specifically OData
- To simplify loading complex data from multiple ISAM files

What Does It Give You?

- Access to Synergy data
  - Including joins
- Optimistic concurrency
- Isolated transactions
- High performance and scalability

Where Does It Come From?

- GitHub
  - Part of the Harmony Core project
  - https://github.com/Synergex/HarmonyCore/tree/master/HarmonyCoreEF
- NuGet
  - https://www.nuget.org/packages/Harmony.Core
  - https://www.nuget.org/packages/Harmony.Core.EF
- CI/CD in Azure DevOps
  - http://harmonybuild.westus.cloudapp.azure.com:8624/feeds/HarmonyCore

Where Does It Run?

- Anywhere Synergy .NET runs
- .NET Standard
  - .NET Framework 4.6.2+
  - .NET Core 2.1+

How Does It Work?

- Synergy Select
  - Where
  - Join
  - OrderBy
- LINQ expressions
- Extensive CodeGen templates
- Your repository

How Do I Write Queries?

- System.Linq.Expression
  - Programmatically construct the operations to be performed
  - C#/VB compilers produce this
- System.Linq.Dynamic
  - Helper to turn textual queries into LINQ expressions
- Extension methods from HarmonyCoreEF
  - Helper to build **Compiled** LINQ expressions

Show Me the Queries (C# LINQ)

```
context.Customers.FirstOrDefault(customer => customer.CustomerNumber >= 8)

context.Customers
        .Where(customer => customer.CustomerNumber >= 8)
        .Include(customer => customer.REL_FavoriteItem);
```

Show Me the Queries (Dynamic)

```
db.Customers
&               .Where("CustomerNumber == @0 and City == @1", 10, “New Orleans")
&               .OrderBy("CustomerNumber")
```

Show Me the Queries (Helpers)

```
context.Customers.FirstOrDefaultIncluding(“REL_FavoriteItem", "CustomerNumber == @0", 8)


context.Customers.FirstOrDefault("CustomerNumber == @0", 8)


context.Customers.Where("CustomerNumber >= @0", new object[#] { 8 })


context.Customers.WhereIncluding("REL_FavoriteItem",
&          "CustomerNumber >= @0", new object[#] { 8 })

context.Customers.WhereIncluding("REL_FavoriteItem,REL_Orders",
&          "CustomerNumber >= @0", new object[#] { 8 })
```

You Said Something about Transactions…

- “Read committed” transaction isolation
- Optimistic concurrency based on GRFAs
- Nothing is written until SaveChanges gets called
- SaveChanges tracks all objects produced by the DBContext
- All tracked objects with changes persist
- SaveChanges either succeeds or fails in its entirety
  - no leftovers after a failure

Sounds Nice, but How?

Harmony Core transactions (under the hood)

0. Call to dbContext.SaveChanges()

Locks in effect

1. Execute user logic and read operations.
2. Lock records that will be updated or deleted.
3. Create and lock new records.
4. Update records.
5. Delete Records.
6. Unlock records.

Transaction complete!

But What Happens If There's an Error?

Harmony Core transactions (under the hood)

0. Call to dbContext.SaveChanges()

1. Execute user logic and read operations.
2. **Lock records that will be updated or deleted.**
3. Create and lock new records.
4. Update records.
5. Delete Records.
6. Unlock records.

Locks in effect

Points of failure

Unable to get a lock? (E.g., record already locked or deleted by another program.)
Response: Throw an exception.

Different GRFA?
Response: Throw an exception.

Hardware failure?
Response: Throw an exception.

But What Happens If There's an Error?

Harmony Core transactions (under the hood)

0. Call to dbContext.SaveChanges()

1. Execute user logic and read operations.
2. Lock records that will be updated or deleted.
3. **Create and lock new records.**
4. Update records.
5. Delete Records.
6. Unlock records.

Locks in effect

Points of failure

Unable to create a record? (E.g., duplicate key or disk full.)
Response: Roll back created records in reverse order and throw an exception.

Hardware failure?
Response: Roll back everything that can be rolled back in reverse order, and throw an exception. (Incomplete transaction.)

But What Happens If There's an Error?

Harmony Core transactions (under the hood)

0. Call to dbContext.SaveChanges()

1. Execute user logic and read operations.
2. Lock records that will be updated or deleted.
3. Create and lock new records.
4. **Update records.**
5. Delete Records.
6. Unlock records.

Locks in effect

Points of failure

Unable to update a record? (E.g., duplicate alternate key where duplicates are not allowed.)
Response: Delete all created records (in reverse order), and write original values to any updated records.

Hardware failure?
Response: Roll back everything that can be rolled back in reverse order, and throw an exception. (Incomplete transaction.)

But What Happens If There's an Error?

Harmony Core transactions (under the hood)

0. Call to dbContext.SaveChanges()

1. Execute user logic and read operations.
2. Lock records that will be updated or deleted.
3. Create and lock new records.
4. Update records.
5. **Delete Records.**
6. Unlock records.

Locks in effect

Points of failure

Hardware failure?
Response: Roll back everything that can be rolled back in reverse order, and throw an exception. (Incomplete transaction.)

But Won’t It Lock Itself out with Manual Locks?

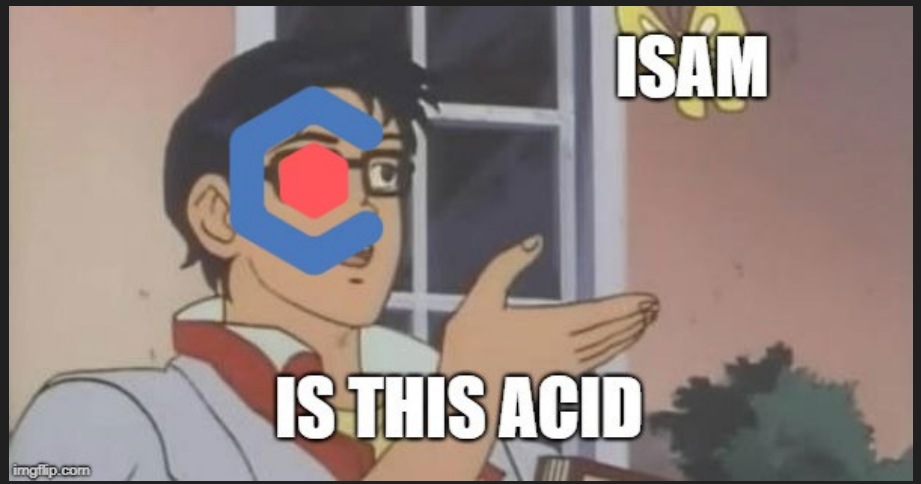
- No!
- The locks are short-lived.
- XCALL FREE is called on channel used by a transaction when it finishes, regardless of success or failure.
- EF Core coalesces changes made to a single object into one operation, so provider doesn’t attempt to acquire the same lock twice.

How Do I Avoid Reading Uncommitted Data?

- Data returned by Read is potentially uncommitted, regardless of lock.
- Otherwise, you only see committed records.
- Locks in this environment are very short-lived, so by default we wait up to a second to acquire them.
- If the lock can’t be acquired after a second, there is almost certainly a deadlock with another transaction.

Is This ACID?

- **Atomicity**
  - Almost (hardware failure and process termination aren't covered)
- **Consistency**
  - Yes, but with a limited set of rules
- **Isolation**
  - Yes
- **Durability**
  - Yes, with flushing



What About My I/O Routines?

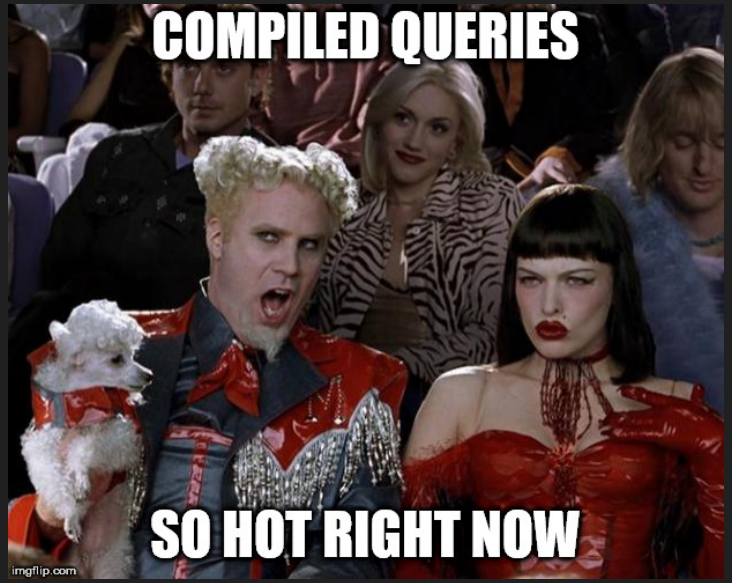
- We support custom I/O routines
  - Inherit from DataObjectIOBase and implement
    - Find
    - Read
    - Write
    - Delete
    - ISInfoCall
    - GetAlphaFileInfo
    - UnlockChannel
  - Support GRFAs and explicit locking

Why Is It so Fast?

- IFileChannelManager
  - Don’t open and close channels, especially *xf*Server
- DataObjectMetadataBase
  - Pooled memory allocation and Fast construction of complex objects
- AddDbContextPool
  - Less construction == (lower latency && higher throughput)

Why is it so Fast (cont.)

- PreparedQueryPlan
  - Perform the heavy lifting only once
  - Next time the query is executed, it runs directly
- Automatic Query Cache in EF Core
  - All queries are checked against the cache before compilation
- Pre-Compiled Queries
  - Converts Linq Expression into a runnable delegate



But My Files Are Strange

- Tag fields to identify the record type
- Multi-segment keys
- Foreign keys with literals
- Allow dups

Is This a Replacement for *xf*ODBC?

- Probably not
  - If this interests you, find me and help me understand your environment.

Does This Work with *xf*Server?

- Yes
- Special considerations
  - OpenVMS generates the GRFAs instead of storing them in files
  - Tuning your query is more important
  - Use the logs
- ThreadedContextBase
  - Provided to allow multiple *xf*Server connections
  - Important in high-scalability scenarios

I Use IOHooks. Do They Still Work?

- Yes
  - Inherit from HookableFileChannelManager.
  - Register your implementation instead of ours by using dependency injection.
  - Implement HookChannel to return an instance of your IOHook.

```
protected abstract method HookChannel, @Synergex.SynergyDE.IOExtensions.IOHooks
        channel, int
        fileName, @string
        openMode, FileOpenMode
proc
endmethod
```

Can I Have It Now?

- Yes!

Other Questions?