Understanding Capabilities of EF Core's Database Connectivity

Julie Lerman

Most Trusted Authority on Entity Framework Core

@JulieLerman   www.thedatafarm.com

Module Overview

High level overview of each topic

So many other DBs you can work with

A look at the provider for Azure Cosmos DB

Database transaction support

Dynamic connection strings

Connection and DBContext pooling

How EF Core handles failing connections

Recognizing the Many Database Providers Available for EF Core 6

Microsoft Created Providers

All start with Microsoft.EntityFrameworkCore

SqlServer
v2012 onwards

Sqlite
v3.7 onwards

InMemory

Cosmos
SQL API only

SQL Server Provider Connects to All SKUs

SQL Server (Enterprise, Standard or Developer)

SQL Server Express / LocalDb

Azure SQL Server

DB Providers from 3rd Party and Open-Source

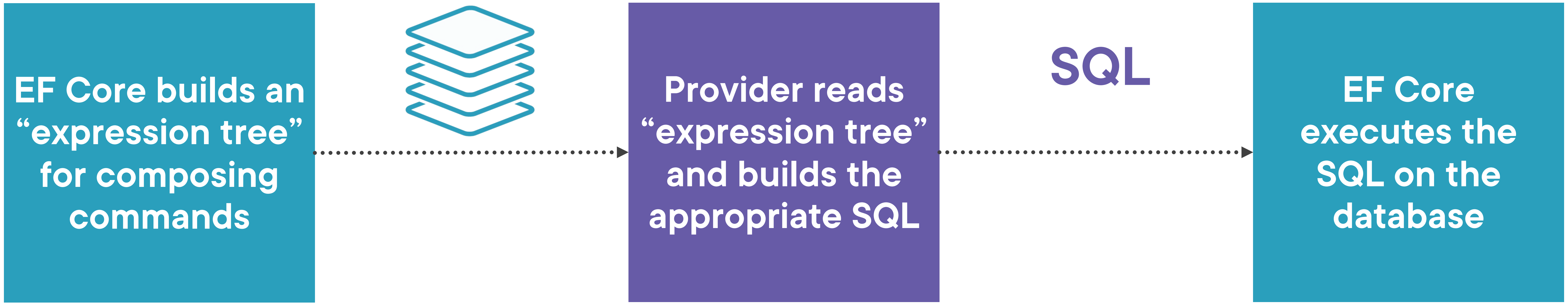
Some are free, some require paid licenses

- ✓ MySQL
- ✓ Oracle DB
- ✓ PostgreSQL
- ✓ SQL Server
- ✓ SQLite
- ✓ Firebird
- ✓ Db2 & Informix
- ✓ MS Access
- ✓ Google Cloud Spanner
- ✓ SQL Server Compact
- ✓ Progress OpenEdge

Keep up to date at docs.microsoft.com/ef/core/providers/

EF Core supports database features and data types common to database servers.

Coordinating to Create and Execute Commands


EF Core builds an “expression tree” for composing commands

Provider reads “expression tree” and builds the appropriate SQL

EF Core executes the SQL on the database

Microsoft works with provider writers, so you can be confident that those listed in the docs are trustworthy.

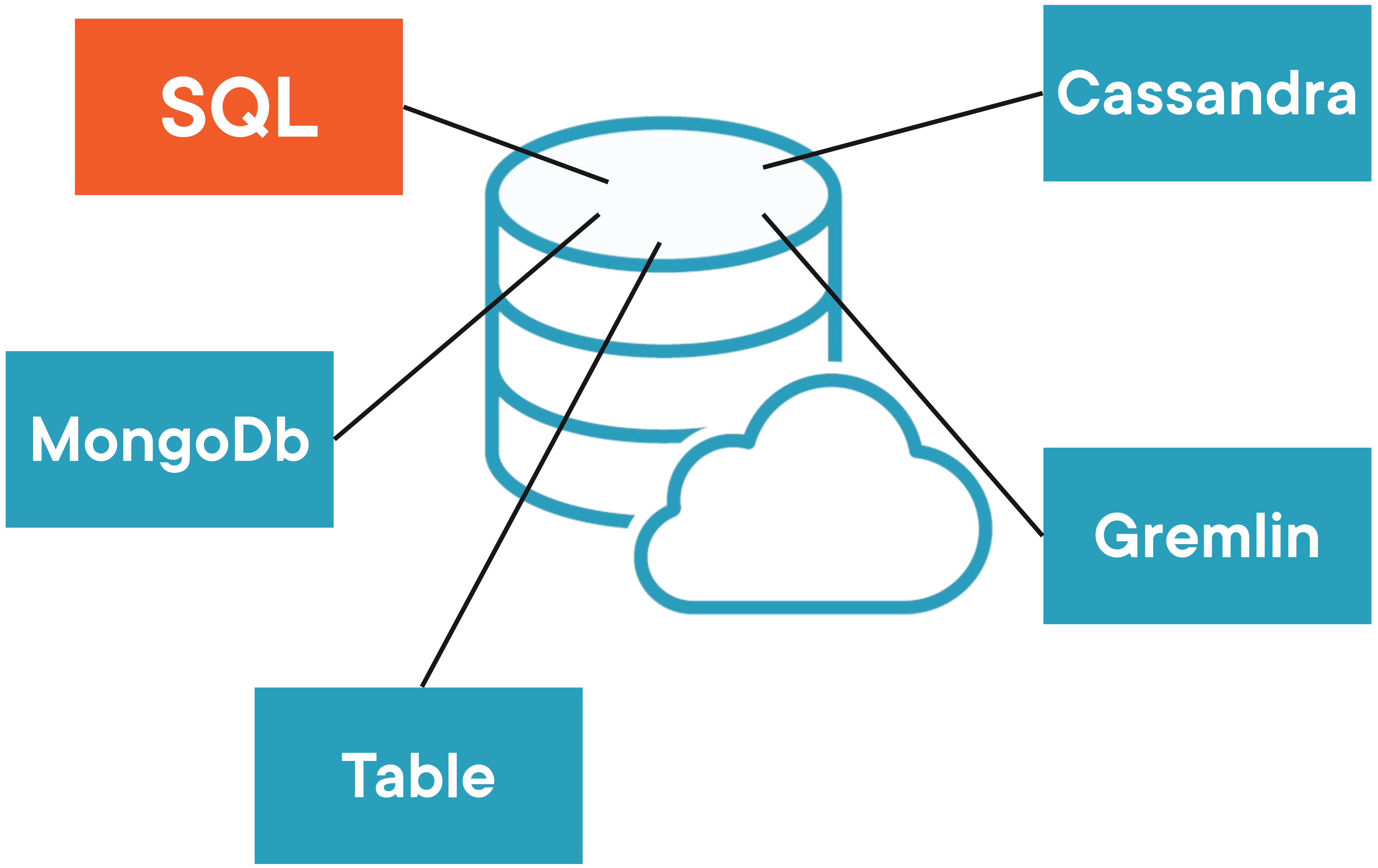
Highlights of the Azure Cosmos DB Provider

A Frequently Asked Question

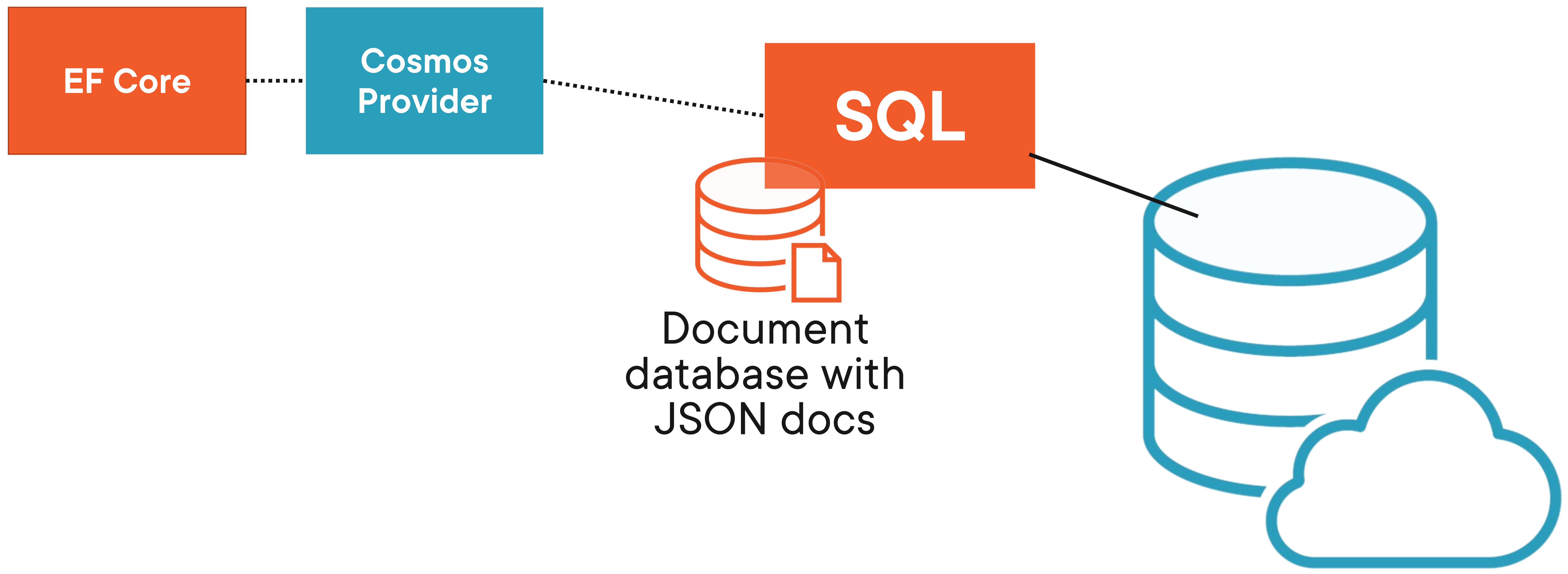
Why use an ORM for a non-relational data store?

Devs wanted to use a familiar way (EF Core) to interact with CosmosDB.

CosmosDB Exposes Various Database APIs



EF Core Can Work with Cosmos' SQL API



Use EF Core as with any other database to query, add, update, and delete.

Configure Connection & More with UseCosmos

```
optionsBuilder.UseCosmos(connectionstring, databasename)
```

EF Core’s Transaction Support and Concurrency Handling

EF Core Transaction Basics

SaveChanges is always wrapped in a DB Transaction

Control workflow of default via Database.Transaction

Override with an ADO.NET database transaction

Override with System.Transactions

Cancel a Book Contract

Delete the book

Add artist notes about the cancellation

```
try
{
  context.SaveChanges();
}
catch (DbUpdateConcurrencyException ex)
{
   //Apply your logic for handling concurrency exceptions
}
```

SaveChanges Uses Optimistic Concurrency

Throws a DbUpdateConcurrencyException on error & rolls back the transaction
Docs provide guidance on handling concurrency exceptions
docs.microsoft.com/ef/core/saving/concurrency
Note: There's also a SaveChangesFailed event handler

Answering Some DB Connection FAQs

Can you dynamically specify connection strings?

```
ASP.NET Core’s program.cs demonstrated reading from environment variables:
 builder.Services.AddDbContext<PublisherData.PubContext>(
    opt => opt.UseSqlServer(
                builder.Configuration.GetConnectionString("PubConnection"));
```

Some Paths to Apply Dynamic Connection Strings

ASP.NET Core appsettings.json has Environment, Production & Development alternate files
Read from environment variables via Microsoft.Extensions.Configuration
Use EF Core interceptors (next module) to change connection string on the fly
Compose from various sources via Felipe Gavilán blog: bit.ly/GavilanExample

What about connection pooling & reusing DbContexts?

Connection pooling is controlled by the provider, not EF Core.

```
builder.Services.AddDbContextPool<PublisherData.PubContext>(
    opt => opt.UseSqlServer(
                builder.Configuration.GetConnectionString("PubConnection"));
```

DbContext Pooling for Performance

Meant to be used in ASP.NET Core apps where scope is controlled
Apply with AddDbContextPool instead of AddDbContext
Also pools connection and other database resources
More at: bit.ly/PoolingDocs

What if there are connection problems during execution?

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    optionsBuilder
        .UseSqlServer(myconnection,
            options => options.EnableRetryOnFailure());
}
```

Built in Connection Resiliency

Use default EnableRetryOnFailure
Specify custom behavior via ExecutionStrategy class to control retry counts and more
More at bit.ly/EFCResiliencyDoc

Review

EF Core supports what's common across the databases

Querying and data mods are the same for all

Even for CosmosDB, but it's your job to understand about modeling for document dbs

Rich database transaction support

Many ways to store/use dynamic conn strings

Connection pooling is driven by the provider

DbContext pooling can help web app perf

EF Core can retry connections as needed

Up Next: Tapping into EF Core’s Pipeline

Resources

Jeremy Likness GitHub for Cosmos and other great samples
github.com/JeremyLikness

Shay Rojansky GitHub for PostgreSQL provider github.com/roji

Modeling Guidance for Azure Cosmos DB
docs.microsoft.com/en-us/azure/cosmos-db/sql/modeling-data

EF Core Docs on Connection Resiliency
bit.ly/EFCResiliencyDoc