Mapping Aggregates to Azure CosmosDB

Julie Lerman

EF Core Expert and Serial DDD Advocate

@julielerman | thedatafarm.com

Why Use an ORM* With Unstructured Data?

Leverage existing skills rather than have to learn a new set of APIs for interacting with CosmosDB

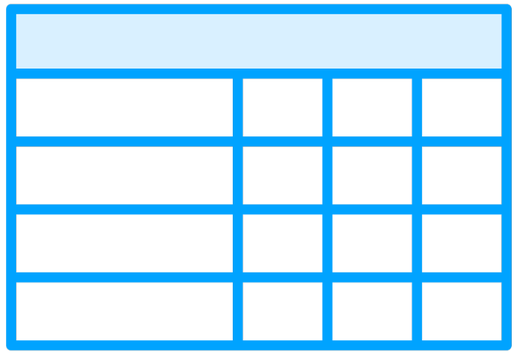
* EF Core is much more than just an ORM ☺

Much More About this Provider

Using EF Core 6 with Azure Cosmos DB

Jurgen Kevelaers

Document Database Removes Impedence Mismatch



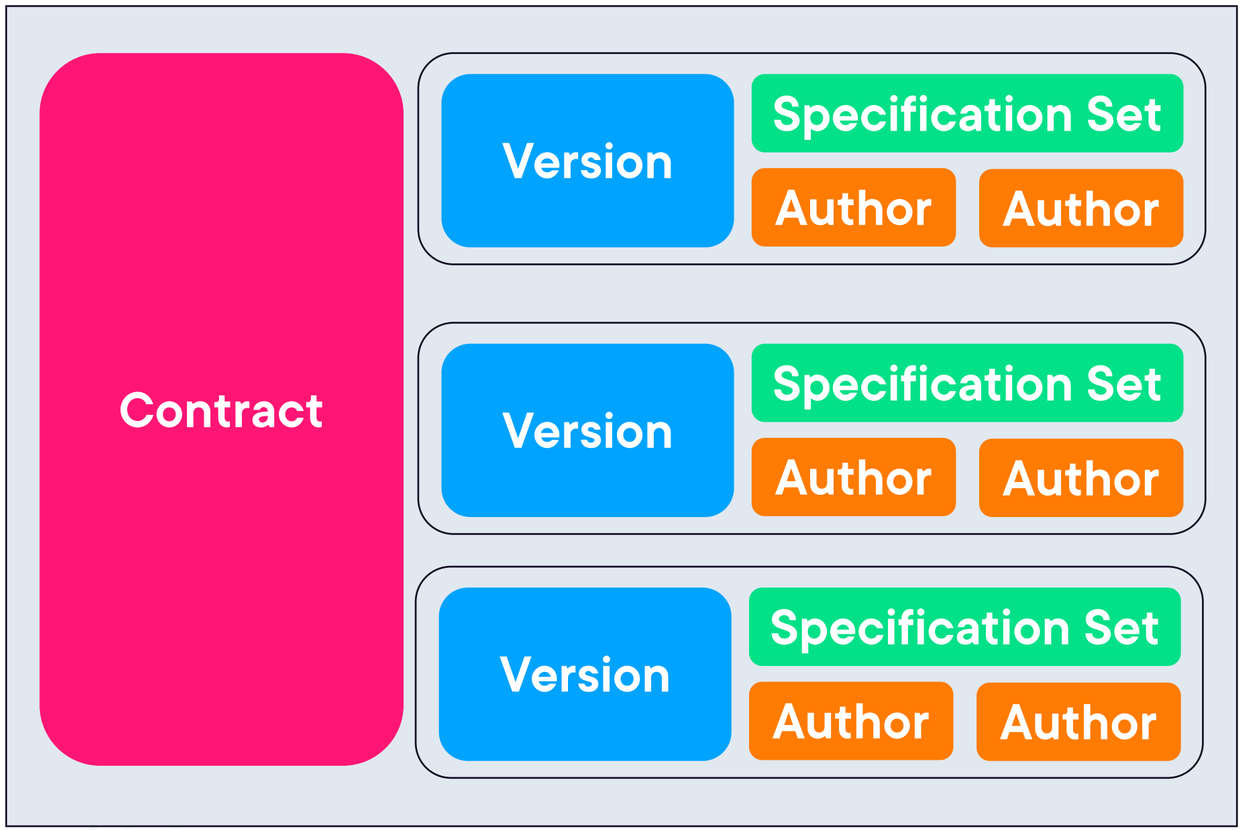
Relational database stores data in a highly structured schema

{JSON}

Document database stores data in JSON documents as unstructured data

This module will focus on the differences that are relevant to mapping the DDD-guided aggregate.

CosmosDB Can Store the Aggregate in One Document




```
{
  “ContractId”: “123”,
  “ContractNumber”: “xyz”,
  “Versions”: [
     {
         “VersionId”: “456”,
          “Accepted”: false,
          "Authors": [
           {
               “Email”: a@a.com,
                “Name”: {
                    “FirstName” : “Julie”
                    “LastName” : “Lerman”
                },
      ],
     “Specs”: {
         “AdvanceAmountUSD”: 1,
         “DigitalRoyaltyPct”: 10
      }
   }
}
```

Some Key Mapping Differences

Some mappings don’t apply to CosmosDB

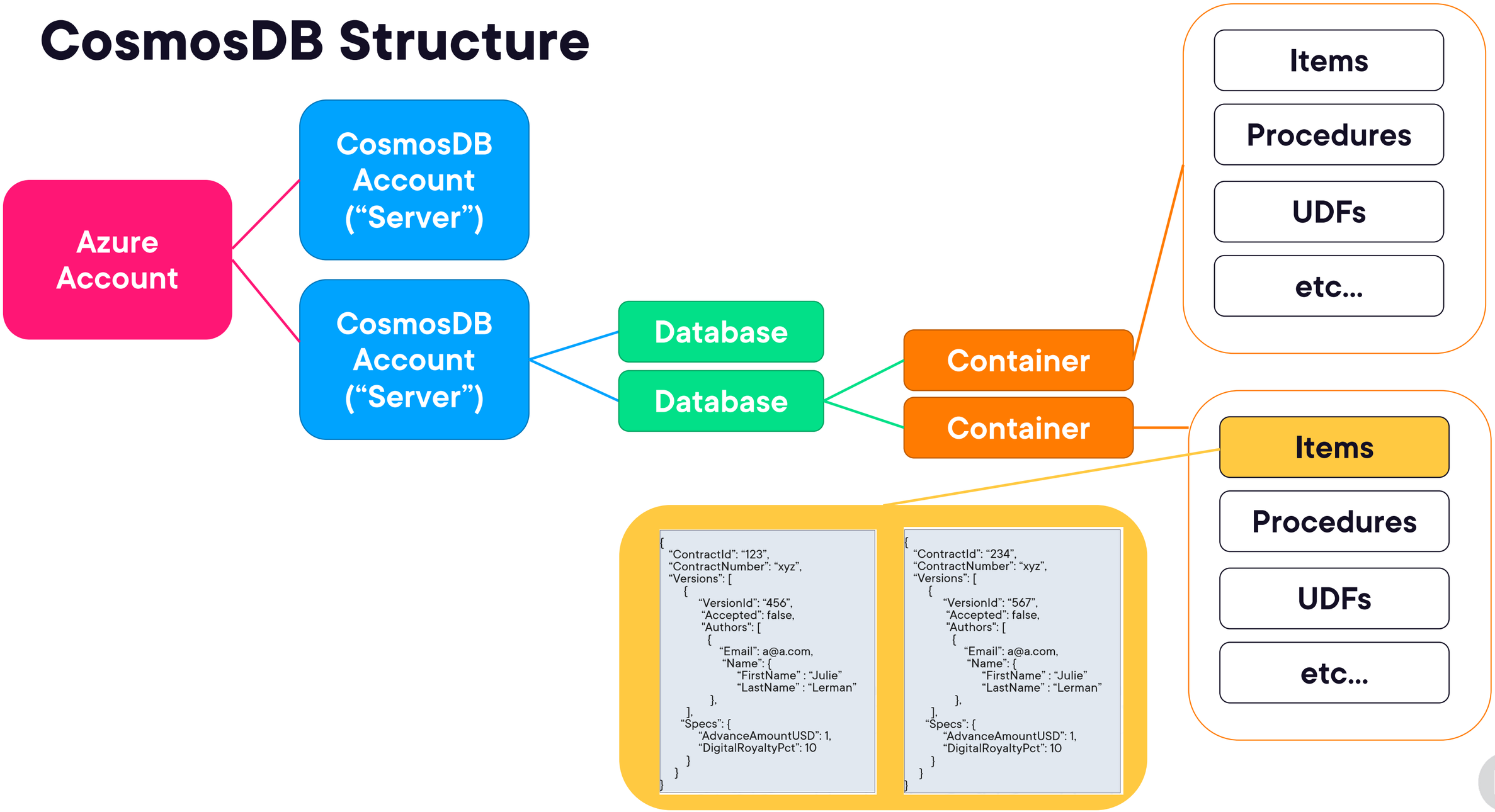
Automatically embedded value objects and related data

Store collections and dictionaries of scalar data

Notes about EF Core 7

Changing the Provider and Removing Incompatible Mappings

CosmosDB Structure

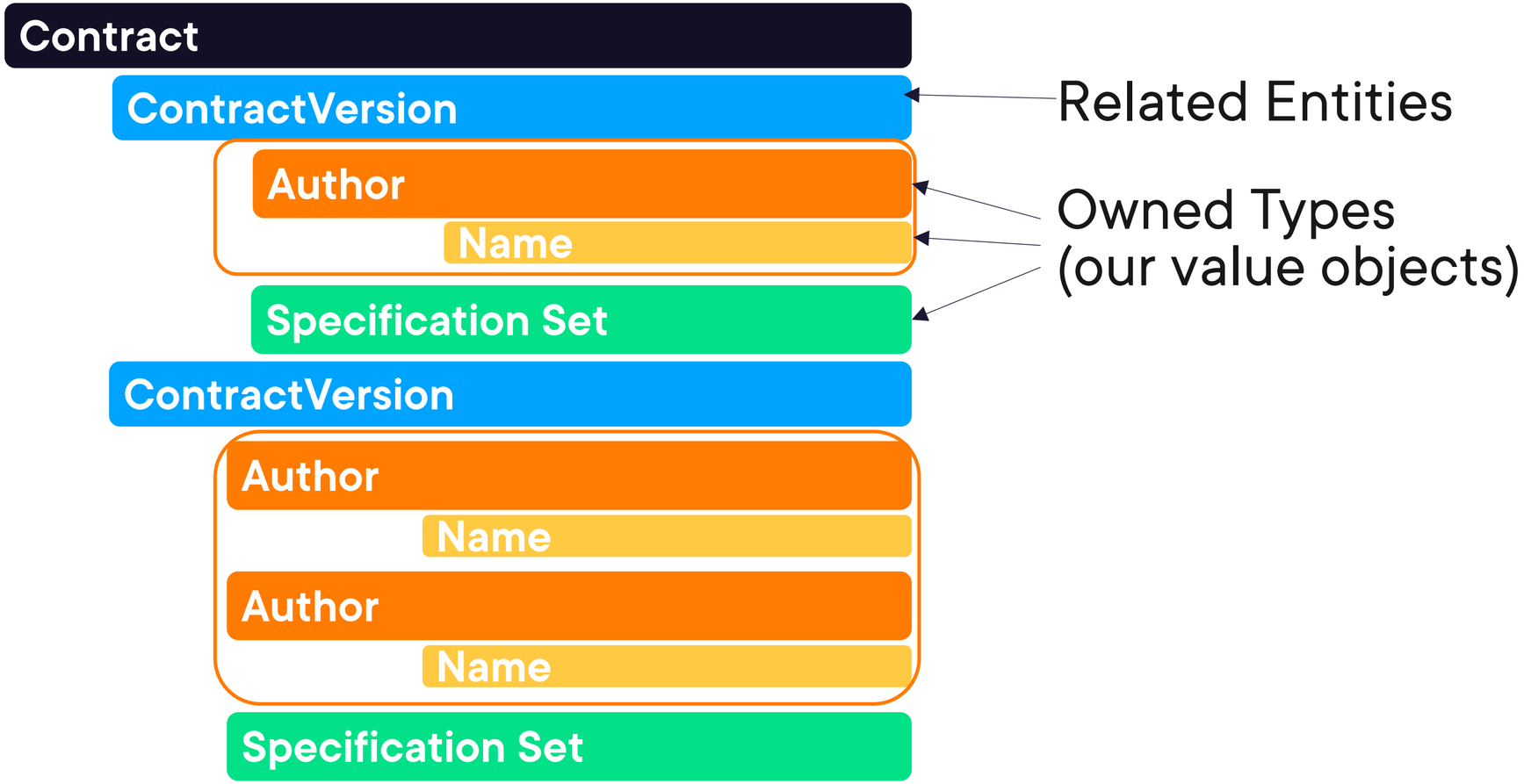


A Global Publishing Powerhouse!

It could become necessary to consider different mappings to improve throughput and distribution

Embedding Value Objects and Related Data in JSON Documents

Default Embedding aka Nesting



Default Embedding aka Nesting


```
{
  “ContractId”: “123”,
  “ContractNumber”: “xyz”,
  “Versions”: [
      {
          “VersionId”: “456”,
          “Accepted”: false,
          "Authors": [
            {
                “Email”: a@a.com,
                “Name”: {
                    “FirstName” : “Julie”
                    “LastName” : “Lerman”
                },
      ],
     “Specs”: {
          “AdvanceAmountUSD”: 1,
          “DigitalRoyaltyPct”: 10
      }
   }
}
```

Embedded By Convention

Value objects
(EF Core owned types)

- ☑ No key property
- ☑ Used as a property of another class

Embedded By Convention

Value objects
(EF Core owned types)

Dependent entities
with no DbSet

Using Integration Tests to Validate the CosmosDB Mapping

Embedded By Convention

Value objects
(EF Core owned types)

Dependent entities
with no DbSet

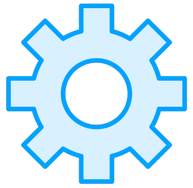
...or Entity mappings

Fluent API to the Rescue

Apply the needed mappings and still have the contract versions embedded

There is very helpful info in EF Core's exception messages!

Explicitly Configure the Entity as an Owned Type



Configure the dependent as an owned entity or owned collection

```
modelBuilder.Entity<Contract>().OwnsMany(c=>c.Versions);
```

Remove (Ignore) the dependent's default mapping, first

```
modelBuilder.Entity<Contract>().Ignore(c => c.Versions);
```

OwnsOne and OwnsMany have 17 overloads!

Single Mapping

```
modelBuilder.Entity<Contract>(c =>
     c.OwnsMany(c => c.Versions,
                owned => owned.Property("_hasRevisedSpecSet")));
```

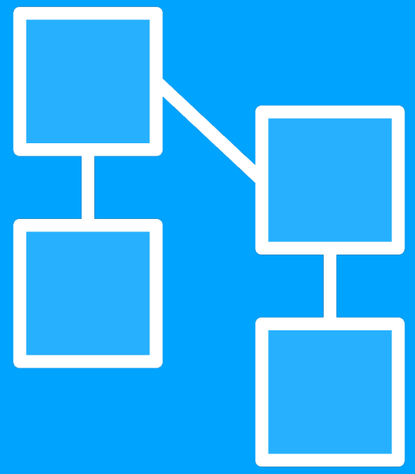
Multiple Mappings

```
modelBuilder.Entity<Contract>(c =>
       c.OwnsMany(c => c.Versions,
       owned => {
                  owned.Property("_hasRevisedSpecSet");
                  owned.Property(v => v.Id).ValueGeneratedNever();
                }));
```

Configuring Owned Types

Lambda expression for a single mapping
Lambda method for multiple mappings

Setting Expectations for Querying Related Data



Embedded Objects are Eager Loaded by Default

No need to use Include methods

Value Objects and Embedded Dependents

It’s standard to retrieve the owned types that are properties of an object

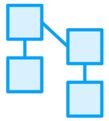
The embedded related objects are treated like owned types and will also come along by default

Querying the Root Will Bring Along the Full Aggregate

```
{
  “ContractId”: “123”,
  “ContractNumber”: “xyz”,
  “Versions”: [
      {
          “VersionId”: “456”,
           “Accepted”: false,
           "Authors": [
            {
                “Email”: a@a.com,
                 “Name”: {
                     “FirstName” : “Julie”
                     “LastName” : “Lerman”
              },
      ],
     “Specs”: {
          “AdvanceAmountUSD”: 1,
          “DigitalRoyaltyPct”: 10
       }
   }
}
```

An aggregate should always be in a valid state; therefore, it should be complete.

Some of the Current Querying Limitations



Embedded objects are *always* included, even if you don't want them

Filtered Include of embedded objects is not supported

Projecting the embedded object without the root must be via NoTracking queries

Filtering or aggregating embedded objects in projections is not supported

See the Azure Cosmos Task List (https://github.com/dotnet/efcore/issues/12086)

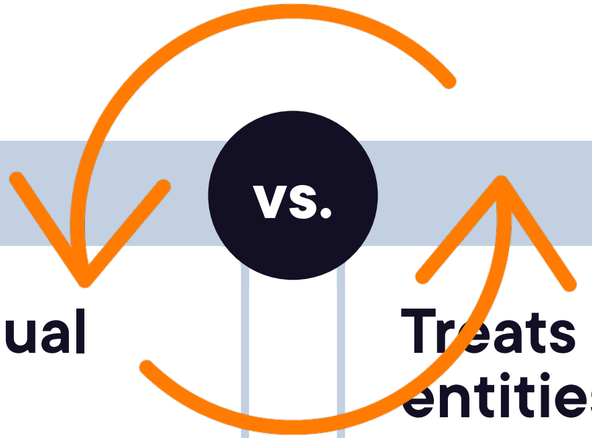
Cosmos Provider Task List

github.com/dotnet/efcore/issues/12086

EF Core Team

Adjusting the Final Tests for Cosmos

EF Core Under the Covers Tricks



Relational Mapping

Treats owned entities as actual entities

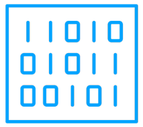
Creates shadow properties for fake keys

Cosmos Mapping

Treats dependent embedded entities like owned entities

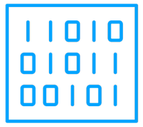
Effortlessly Storing Dictionaries and Lists of Primitives

.NET Primitive Types (Collections Map by Convention)

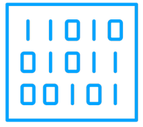


Byte, Sbyte

Int16, UInt16, Int32, UInt32, Int64, UInt64, IntPtr, UIntPtr



Char



Double

Single

Cosmos Persists by Convention:

```
public Dictionary<int,string>
  SomePairs{get; set;}
```

```
“SomePairs ": [
   “a“ : “the first letter”,
   “z“ : “the last letter”
],
```

Dictionaries of <string,string>

```
public List<string> SomeStrings{get; set;}
```

```
“SomeStrings ": [
   “item 1",
   “item 2“
],
```

Collections of Primitive Types

“We should track social media for potential authors during contracting.”

Azure Cosmos DB

Build or modernize scalable, high-performance apps.

Try Azure Cosmos DB free | Create a pay-as-you-go account

Overview Features Security Pricing Get started Customer stories Resources FAQ

Fast, distributed NoSQL and relational database at any scale

Develop high-performance applications of any size or scale with a fully managed and serverless distributed database supporting open-source PostgreSQL, MongoDB, and Apache Cassandra. Get automatic and instant scalability, with SLA-backed single-digit millisecond reads and writes and 99.999 percent availability for NoSQL data. Deploy and scale applications with distributed PostgreSQL using the latest version, tools, and extensions.

Review

Teams have chosen CosmosDB and devs want to benefit from EF Core skills

DDD tactical design fits nicely in JSON

Conventionally embedded objects

Retrieve entire document

Easily support dictionaries & primitive lists

Limitations on querying partial documents

Different mapping needs than RDBMS

Resources from This Module

Cosmos EF provider modifying JSON values with the "id" in identifier
https://github.com/dotnet/efcore/issues/28600

Cosmos Provider Task List
https://github.com/dotnet/efcore/issues/12086

Accessing JSON via Cosmos Provider
https://learn.microsoft.com/en-us/ef/core/providers/cosmos/unstructured-data

Up Next:

Organizing Persistence Logic to Support DDD Design