Вкусные новинки EF CORE 8

Андрей Александров

Вкусные новинки EF CORE 8

Андрей Александров

Андрей Александров

О чем сегодня поговорим ?

- Complex type
- Sentinel Values
- Поддержка типа HierarchyId
- Работа с коллекциями примитивов
- Raw SQL queries for unmapped types
- и пара слов о других фичах

Complex Type

Complex Type

О чем речь и зачем?

Все объекты в БД можно разделить на:

- Unstructured
- Structured with id
- Structured without id

Complex Type

О чем речь и зачем?

Все объекты в БД можно разделить на:

- Unstructured
- Structured with id
- $\boxed{\text{Structured without id}}$

Complex Type

```
public class Passport
{
    public Guid Id { get; set; }
    public Person Owner { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public required PassportInfo PassportInfo { get; set; }
}
```

```
[ComplexType]
public class PassportInfo
{
    public int Serial { get; set; }
    public int Number { get; set; }
}
```

```
public class Passport
{
    public Guid Id { get; set; }
    public Person Owner { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public required PassportInfo PassportInfo { get; set; }
}
```

Complex Type

```
public class Person
{
    public Guid Id { get; set; }
    public string Name { get; set; }
}
```

```
[ComplexType]
public class PassportInfo
{
    public int Serial { get; set; }
    public int Number { get; set; }
}
```

```
public class Passport
{
    public Guid Id { get; set; }
    public Person Owner { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public required PassportInfo PassportInfo { get; set; }
}
```

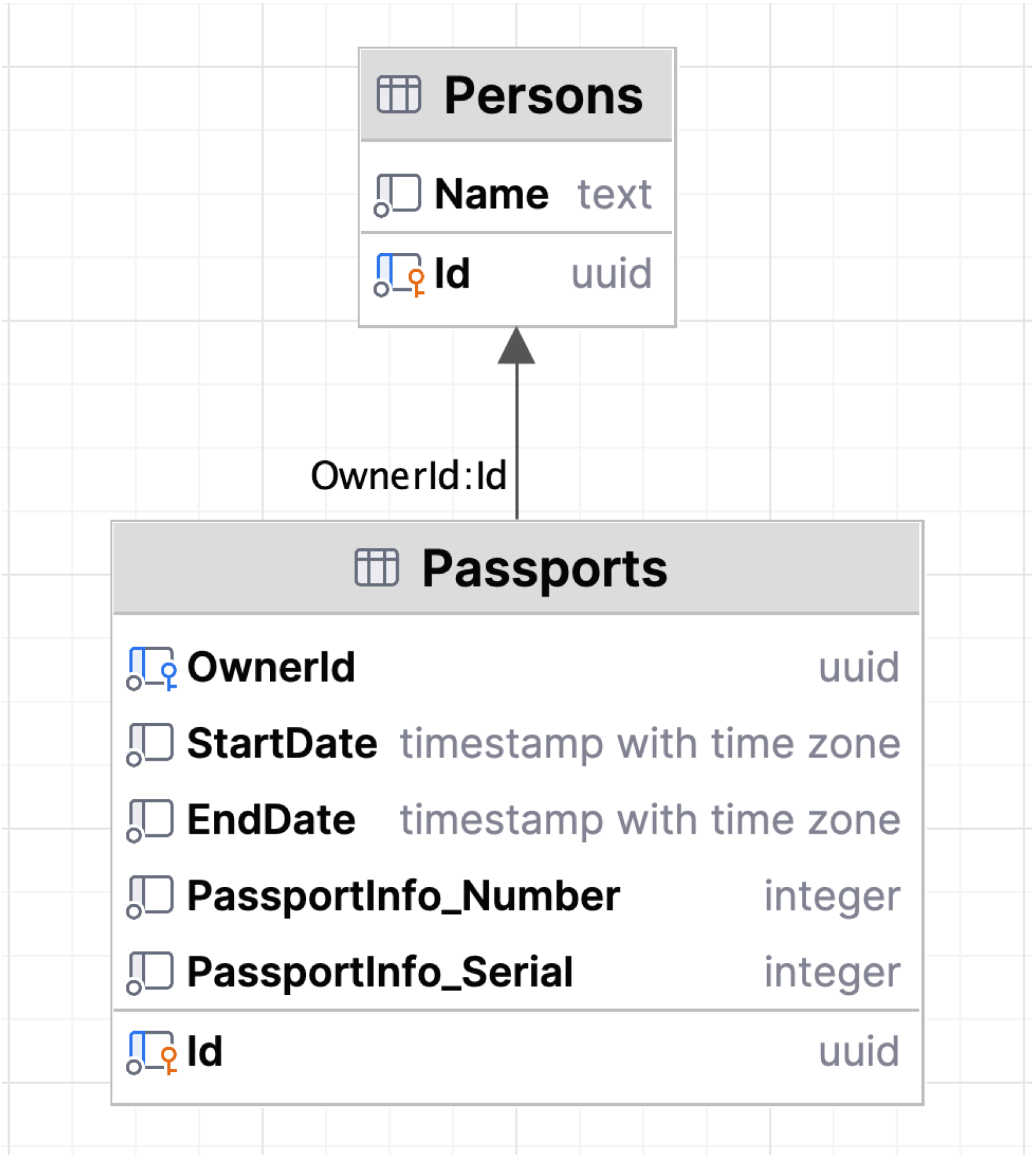
Complex Type

```
public class ApplicationContext : DbContext
{
    public DbSet<Person> Persons { get; set; }
    public DbSet<Passport> Passports { get; set; }

    //...

}
```

Complex Type

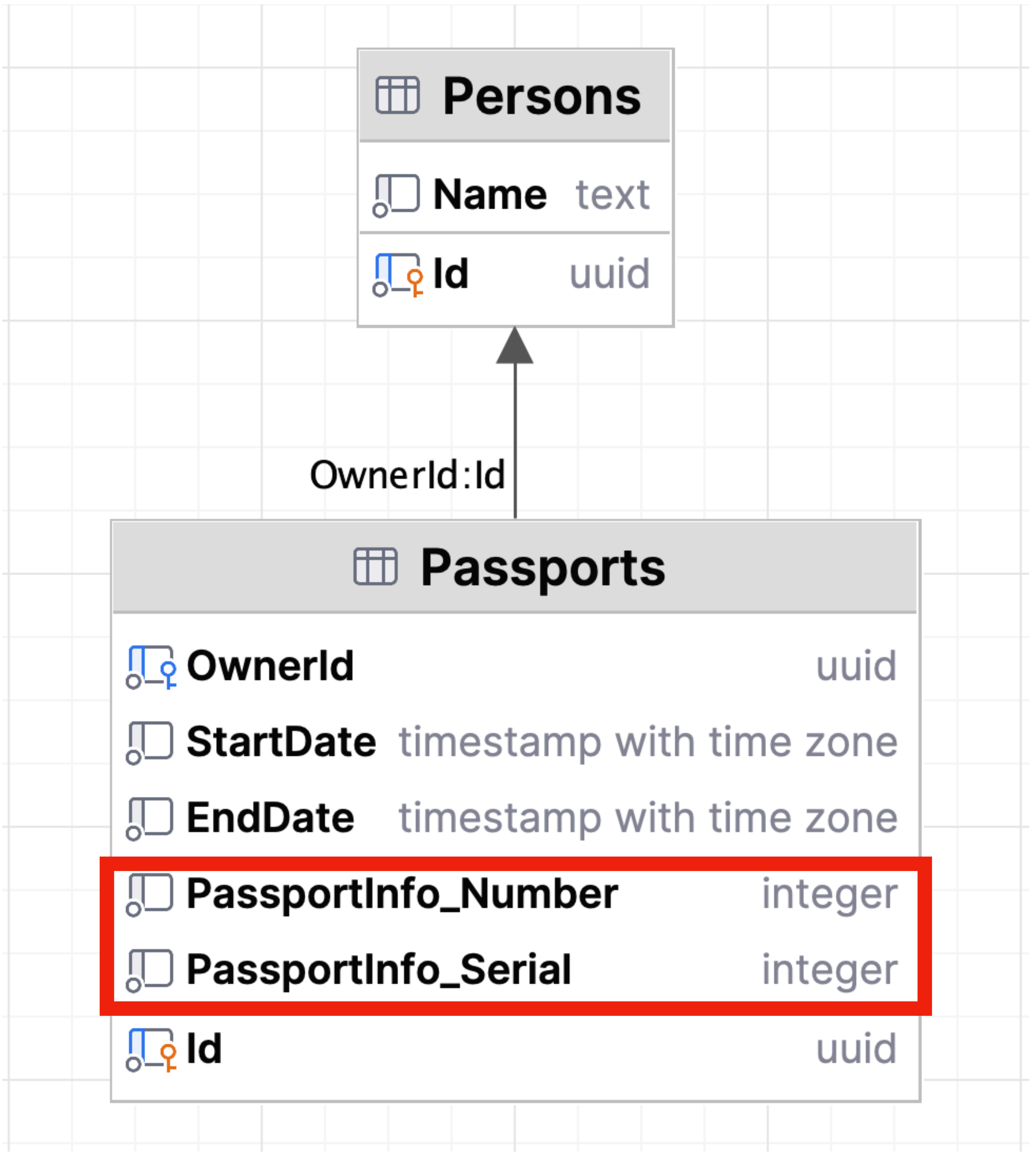


Persons	
Name	text
Id	uuid

OwnerId:Id

Passports	
OwnerId	uuid
StartDate	timestamp with time zone
EndDate	timestamp with time zone
PassportInfo_Number	integer
PassportInfo_Serial	integer
Id	uuid

Complex Type


Persons
Name text
Id uuid
OwnerId:Id
Passports
OwnerId uuid
StartDate timestamp with time zone
EndDate timestamp with time zone
PassportInfo_Number integer
PassportInfo_Serial integer
Id uuid

Complex Type

```
CREATE TABLE "Passports"
(
    "Id"                     uuid                     NOT NULL,
    "OwnerId"                uuid                     NOT NULL,
    "StartDate"              timestamp with time zone NOT NULL,
    "EndDate"                timestamp with time zone NOT NULL,
    "PassportInfo_Number" integer                     NOT NULL,
    "PassportInfo_Serial" integer                     NOT NULL,
    CONSTRAINT "PK_Passports" PRIMARY KEY ("Id"),
    CONSTRAINT "FK_Passports_Persons_OwnerId" FOREIGN KEY ("OwnerId")
REFERENCES "Persons" ("Id") ON DELETE CASCADE
);
```

Complex Type

```
await db.Passports
    .Where(p => p.PassportInfo.Number == 1)
    .ToListAsync(ct);
```

```
SELECT p."Id",
       p."EndDate",
       p.”OwnerId ",
p." StartDate", p."PassportInfo_Number", p."PassportInfo_Serial"
      FROM "Passports" AS p
      WHERE p.”PassportInfo_Number" = 1
```

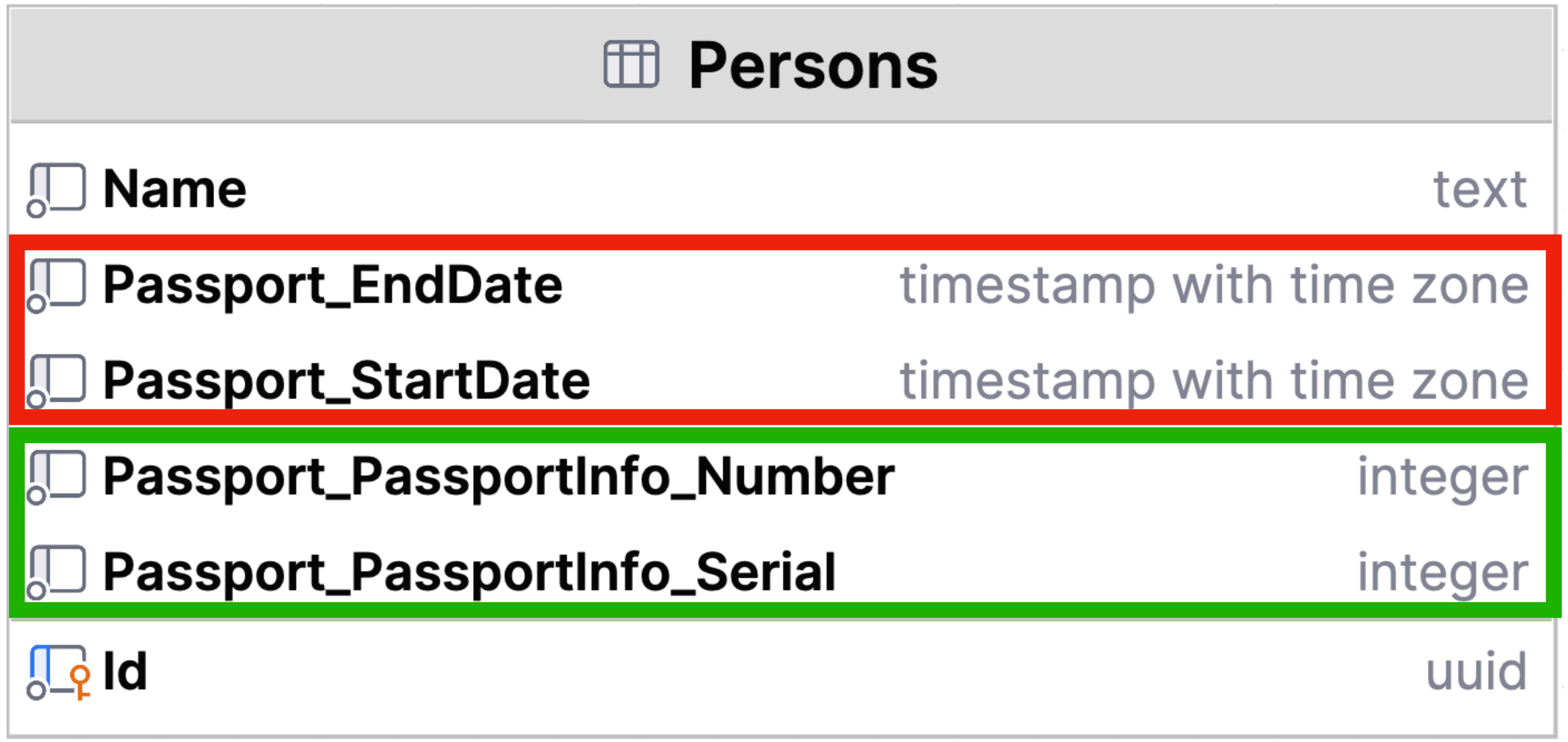
Complex Type

```
public class Person
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public required Passport Passport { get; set; }

}

[ComplexType]
public class Passport
{
    ~~public Guid Id { get; set; }~~
    ~~public Person Owner { get; set; }~~
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
    public required PassportInfo PassportInfo { get; set; }
}
```

Complex Type



Persons	
Name	text
Passport_EndDate	timestamp with time zone
Passport_StartDate	timestamp with time zone
Passport_PassportInfo_Number	integer
Passport_PassportInfo_Serial	integer
Id	uuid

Sentinel Values

Sentinel Values

```
public class Character
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public long Health { get; set; }
    public long PowerPoint { get; set; }
}
```

Sentinel Values

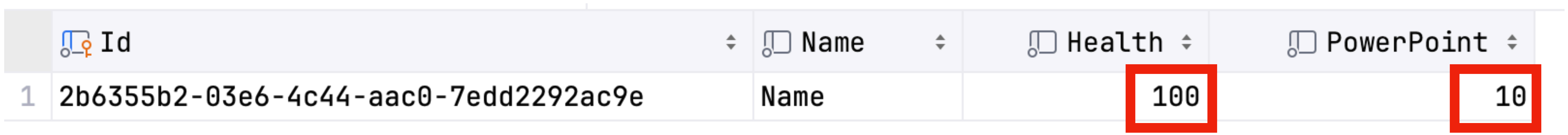
```
public class CharacterConfiguration : IEntityTypeConfiguration<Character>
{
    public void Configure(EntityTypeBuilder<Character> builder)
    {
        builder.Property(p => p.Health).HasDefaultValue(100);
        builder.Property(p => p.PowerPoint).HasDefaultValue(10);
    }
}
```

Sentinel Values

```
var character = new Character
{
    Id = Guid.NewGuid(),
    Name = “Name”,
    Health = 0,
    PowerPoint = 0

};
await db.Character.AddAsync(character);
```

Sentinel Values



	Id	Name	Health	PowerPoint
1	2b6355b2-03e6-4c44-aac0-7edd2292ac9e	Name	100	10

Sentinel Values

EF использует значение

по умолчанию

в качестве маркера

Sentinel Values

```
public class CharacterConfiguration : IEntityTypeConfiguration<Character>
{
    public void Configure(EntityTypeBuilder<Character> builder)
    {
        builder.Property(p => p.Health).HasDefaultValue(100).HasSentinel(-1);
        builder.Property(p => p.PowerPoint).HasDefaultValue(10).HasSentinel(-1);
    }
}
```

Hierarchyld

HierarchyId

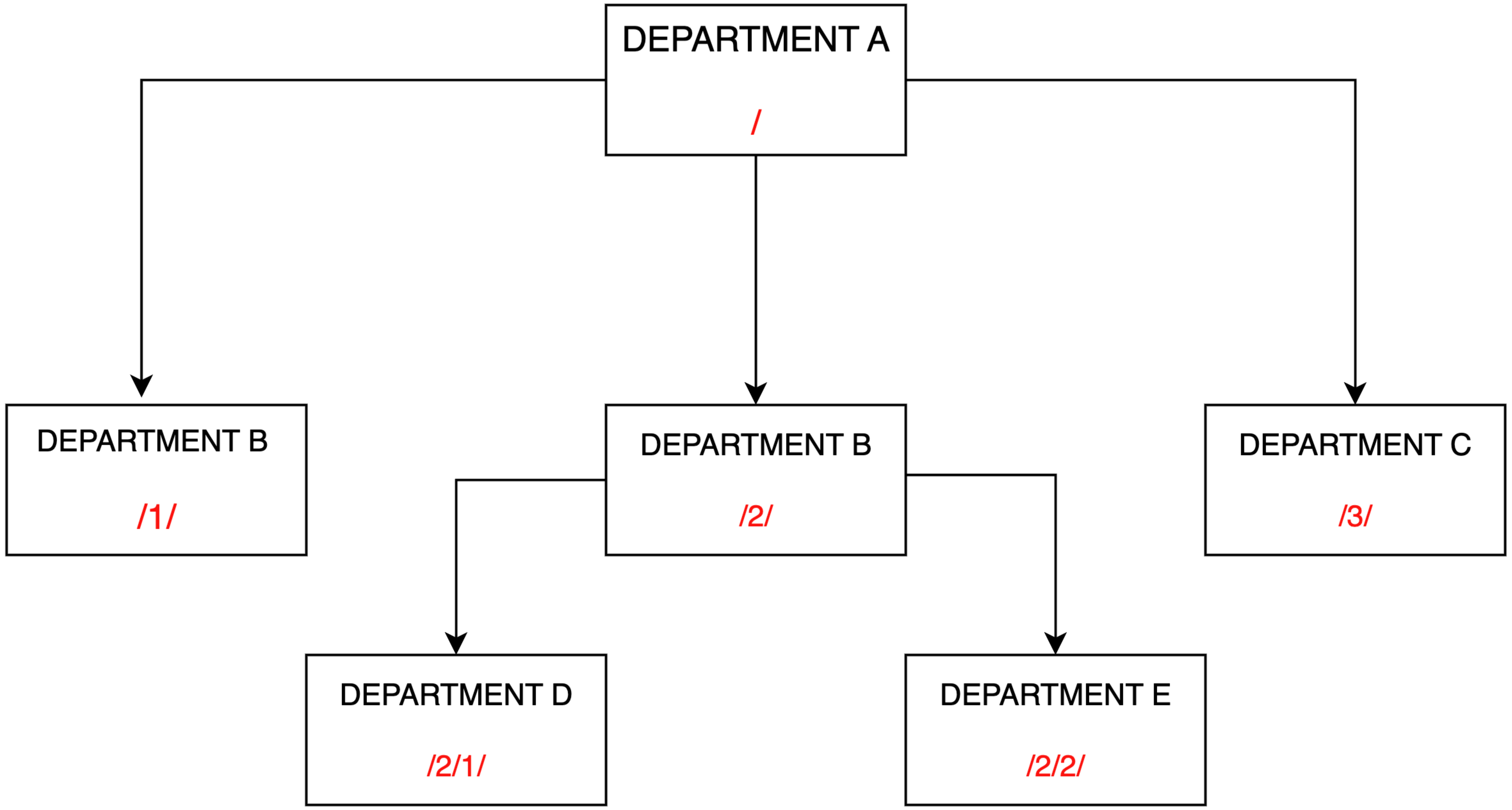
А это что такое?

- В Azure SQL и в SQL Server для определения иерархических структур можно использовать специальный тип данных **HierarchyId**
- EF CORE 8 позволяет использовать этот тип нативно
- **HierarchyId** позволяет работать с иерархическими данными не изобретая “велосипеды”

HierarchyId

```
public class Department
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public HierarchyId PathFromRoot { get; set; }
}
```

Hierarchyld



HierarchyId

```
await db.Departments
    .Where(d => d.PathFromRoot.GetLevel() == currentLevel)
    .ToListAsync();
```

HierarchyId

```
short currentLevel = 3;
await db.Departments
    .Where(d => d.PathFromRoot.GetLevel() == currentLevel)
    .ToListAsync();
```

HierarchyId

```
SELECT [d].[Id], [d].[Name], [d].[PathFromRoot]
FROM [Departments] AS [d]
WHERE [d].[PathFromRoot].GetLevel() = @__level_0
```

Primitive Collections

Primitive Collections

Как оно было раньше?

- создание связи many-to-many между сущностью и таблицей и примитивами (например tag, которые являются Enum)
- рукописные конверторы в jsonb и ему подобные

Primitive Collections

```
public class Book
{
    public Guid Id { get; set; }
    public List<BookTag> Tags { get; set; } = new ();
    public string Name { get; set; }
}

public enum BookTag
{
    Common,
    Bestseller,
    Other
}
```

Primitive Collections

```sql
CREATE TABLE "Books"
(
    "Id"   uuid NOT NULL,
    "Tags" integer[] NOT NULL,
    "Name" text NOT NULL,
    CONSTRAINT "PK_Books" PRIMARY KEY ("Id")
);
```

Primitive Collections

```
CREATE TABLE "Books"
(
    "Id"   uuid NOT NULL,
    "Tags" integer[] NOT NULL,
    "Name" text NOT NULL,
    CONSTRAINT "PK_Books" PRIMARY KEY ("Id")
);
```

Primitive Collections

```
var book = new Book
{
    Name = "Book1",
    Id = Guid.NewGuid(),
    Tags = new List<BookTag>(new[] { BookTag.Bestseller, BookTag.Common })
};
await db.AddAsync(book);
```

```
INSERT INTO "Books" ("Id", "Name", "Tags")
VALUES (@p0, @p1, @p2);
```

Primitive Collections

```csharp
await db.Books.Where(b => b.Tags.Contains(BookTag.Bestseller))
    .ToListAsync(ct);
```

```sql
SELECT b."Id", b."Name", b."Tags"
FROM "Books" AS b
WHERE b."Tags" @> ARRAY[1]::integer[]
```

Primitive Collections

```
await db.Books.Where(b => b.Tags.Contains(BookTag.Bestseller))
    .ToListAsync(ct);
```

```
SELECT b."Id", b."Name", b."Tags"
FROM "Books" AS b
WHERE b."Tags" @> ARRAY[1]::integer[]
```

Primitive Collections

```
public IEnumerable<int> Ints { get; set; }
public ICollection<string> Strings { get; set; }
public ISet<DateTime> DateTimes { get; set; }
public IList<DateOnly> Dates { get; set; }
public uint[] Unsignedints { get; set; }
public List<bool> Booleans { get; set; }
public IEnumerable<BookTag> Tags { get; set; }
```

Raw SQL queries

Raw SQL queries

- EF CORE 8 принес возможность использования “сырых” upmapped sql запросов, что позволило легко использовать сложны sql запросы, которые не поддерживаются EF CORE при работе с базами данных:
- рекурсия
- оконные функции и т.д.

Raw SQL queries

```
public class ItemWithNameHistory
{
    public Guid Id { get; set; }
    public DateTime DtStart { get; set; }
    public DateTime? DtStop { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
}
```

Raw SQL queries

```
public class ApplicationContext : DbContext
{
    public DbSet<ItemWithName> ItemWithName { get; set; }
    public DbSet<ItemWithNameHistory> ItemWithNameHistory { get; set; }

    //...
}

public class UnmappedValue
{
    public string Name { get; set; }
}
```

Raw SQL queries

```
public class ApplicationContext : DbContext
{
    public DbSet<ItemWithName> ItemWithName { get; set; }
    public DbSet<ItemWithNameHistory> ItemWithNameHistory { get; set; }

    //...
}
```

```
public class UnmappedValue
{
    public string Name { get; set; }
}
```

Raw SQL queries

```
var result = await db.Database.SqlQuery<UnmappedValue>(
    $"""
    SELECT
         concat(
                 concat(hh."FirstName", ' ', hh."LastName"),
                 ' => ',
                 concat(phh."FirstName", ' ',phh."LastName")) as "Name"
     FROM "ItemWithNameHistory" as hh
     JOIN "ItemWithNameHistory" as phh
          ON hh."HumanId" = phh."HumanId" AND hh."DtStart" = phh."DtStop"
     WHERE hh."HumanId" = {requestId}
    """)
    .FirstAsync(ct);
```

Raw SQL queries

```
var result = await db.Database.SqlQuery<UnmappedValue>(
    $"""
    SELECT
         concat(
                concat(hh."FirstName", ' ', hh."LastName"),
                ' => ',
                concat(phh."FirstName", ' ',phh."LastName")) as "Name"
     FROM "ItemWithNameHistory" as hh
     JOIN "ItemWithNameHistory" as phh
          ON hh."HumanId" = phh."HumanId" AND hh."DtStart" = phh."DtStop"
     WHERE hh."HumanId" = {requestId}
    """)
    .FirstAsync(ct);
```

Raw SQL queries

```
SELECT u."Name"
FROM (
         SELECT
             concat(
                     concat(hh."FirstName", ' ', hh."LastName"),
                     ' => ',
                     concat(phh."FirstName", ' ',phh."LastName")) as "Name"
         FROM "ItemWithNameHistory" as hh
                  JOIN "ItemWithNameHistory" as phh
                       ON hh."HumanId" = phh."HumanId" AND hh."DtStart" =
phh."DtStop"
         WHERE hh."HumanId" = @p0
     ) AS u
    LIMIT 1
```

Raw SQL queries

```
SELECT u."Name"
FROM (
         SELECT
             concat(
                     concat(hh."FirstName", ' ', hh."LastName"),
                     ' => ',
                     concat(phh."FirstName", ' ',phh."LastName")) as "Name"
         FROM "ItemWithNameHistory" as hh
                  JOIN "ItemWithNameHistory" as phh
                       ON hh."HumanId" = phh."HumanId" AND hh."DtStart" =
phh."DtStop"
         WHERE hh."HumanId" = @p0
     ) AS u
    LIMIT 1
```

А что еще есть интересного ?

А вот что!

- Lazy-loading для AsNoTracking()
- Улучшили работы с JsonColumnMapping
- Перформас, перформанс, перформас
- Новыя логика работы с contains запросами

А нужно ли обновляться ?

Хорошие ссылки (я проверил)

- https://learn.microsoft.com/en-us/ef/core/what-is-new/ef-core-8.0/whatsnew
- https://devblogs.microsoft.com/dotnet/announcing-ef8-rc1
- https://www.youtube.com/watch?v=5HapqzoxJ60&list=PLdo4fOcmZ0oX0ObHwBrJ0vJpZ7PiYMqeA
- https://devblogs.microsoft.com/dotnet/announcing-ef8-rc2/

Congratulations, you've reached the end of the internet!

СПАСИБО ЗА ВНИМАНИЕ

@ANDREY_PUBLIC