**1.** Предположим, у нас есть следующее описание контекста EF Core:

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using Microsoft.EntityFrameworkCore;

public class MyDbContext : DbContext

{

public DbSet<Account> Accounts { get; set; }

}

public class Account

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public long AccountId { get; set; }

public string Username { get; set; }

[NotMapped]

public string Password

{

get

{

return Encryption.DecryptString(PasswordEncrypted);

}

set

{

PasswordEncrypted = Encryption.EncryptString(value);

}

}

public string PasswordEncrypted { get; set; }

}

где EncryptString и DecryptString выполняют шифрацию/дешифрацию строки.

Мы хотим сделать так, чтобы свойство PasswordEncrypted стало private (и не торчало более наружу). Что нужно поменять в коде, чтобы PasswordEncrypted после этого маппилось к БД, а приложение продолжало корректно взаимодействовать с БД?

**2.** У нас есть следующий JSON (см. аттач)

Он имеет следующую структуру: в data → elements содержится массив ссылок на корневые элементы, а в included сложен массив из разнообразных объектов, включая корневые. Обратите внимание, что ноды в included не имеют никакой внутренней иерархии. Ссылки из data → elements ссылаются на ноды, имеющие свойство "$type": "Event" и адресуются по entityUrn, например "entityUrn": "urn:li:fs\_event:(6569262996214943744,S6569262996214943744\_500)".

Данный JSON нужно десериализовать в объекты типа Event, преобразуя их из “плоской” к иерархической структуре, а именно:

Event []

├─ createdAt

├─ entityURN

├─ subtype

├─ eventContent ─ MessageEvent []

│ ├─ $type

│ ├─ $id

│ ├─ body

│ └─ attributedBody ─ AttributedText

│ ├─ $type

│ ├─ $id

│ └─ text

├─ from ─ MessagingMember

│ ├─ $type

│ ├─ entityUrn

│ ├─ nameInitials

│ └─ miniProfile ─ MiniProfile

│ ├─ $type

│ ├─ firstName

│ ├─ lastName

│ ├─ firstName

│ ├─ occupation

│ ├─ objectUrn

│ ├─ entityUrn

│ ├─ publicIdentifier

│ ├─ trackingId

│ └─ picture ─ VectorImage

│ ├─ $type

│ ├─ $id

│ ├─ rootUrl

│ └─ artifacts ─ VectorArtifact []

│ ├─ $type

│ ├─ $id

│ ├─ width

│ ├─ height

│ ├─ expiresAt

│ └─ fileIdentifyingUrlPathSegment

└─ previousEventInConversation ─ Event

**2.1.** Необходимо предложить решение, как выполнить десериализацию данного JSON (целиком код десериализатора писать естественно необязательно, но нужно объяснить, как это сделать в принципе; при этом можно использовать любые внешние инструменты и библиотеки).

**2.2.** Предложите универсальное решение, позволяющее десериализовать такой JSON: предположим у нас есть множество объектов, представленных в данном формате JSON, причем использующиеся в них типы повторяются, таким образом писать отдельный десериализатор для каждого нового объекта нерационально. В таком случае, оптимальным решением было бы отдельно описать модель каждого объекта и использовать универсальную функцию десериализации для обработки всех объектов в подобном формате. Каким вы видите решение для реализации подобной функции?

**3.** В таблицу Tasks периодически помещаются некоторые задания (raw запросы SQL). Каждый task имеет атрибуты: TaskId, ClientId, CommandSQL, WaitingTime (ms), TaskStatus и ErrorCounter.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TaskId | ClientId | CommandSQL | WaitingTime | TaskStatus | ErrorCounter |
| 1 | 1001 | SELECT X FROM Y WHERE X = 23 | 12322 | InProgress | 1 |
| 2 | 1002 | SELECT X FROM Y WHERE X = 516 | 34223 | New | 0 |
| 3 | 1003 | SELECT X FROM Y WHERE X = 234 | 22424 | New | 0 |
| 4 | 1001 | SELECT X FROM Y WHERE X = 888 | 4882 | New | 0 |

Каждое задание выполняется с помощью *N* запущенных параллельно рабочих процессов (workers), каждый worker выполняет задание, после чего получает следующее или засыпает на некоторое время (если больше заданий для него нет). При этом необходимо выполнение следующих правил:

1. Каждое новое задание помещается в таблицу со статусом New.

2. Каждое новое задание должен выполнять в данный момент только один worker. Недопустимо, чтобы одно и тоже задание выполнялось одновременно двумя и более workers. Задание, находящееся в процессе исполнения, помечается статусом InProgress.

3. Если задание было выполнено успешно, ему присваивается статус Success.

4. Если задание было выполнено с ошибкой, которую worker смог обработать, оно снова может быть выполнено, при этом счетчик ErrorCounter увеличивается на 1.

5. Если по истечении определенного времени задание не было выполнено (worker свалился в необработанный exception, например) задание снова становится доступным для выполнения, а счетчик ошибок увеличивается на 1.

6. Если ErrorCounter превышает 3, задание помечается статусом Failed и более не должно отдаваться на выполнение.

7. Необходимо придерживаться правила, что для каждого ClientId следующее задание может быть выполнено не ранее, чем через то количество миллисекунд, которое указано в поле WaitingTime. Например, worker получил на выполнение задание TaskId = 1. По окончании выполнения задания worker проверяет, нет ли в таблице других заданий, назначенных ClientId = 1001 – такое задание есть, это TaskId = 4. Это задание должно быть выполнено не ранее чем через 12322 мс (но можно и позже) после того, как будет выполнен TaskId = 1. Важно, чтобы при выполнении двух разных заданий, принадлежащих одному и тому же ClientId, между ними была задержка, не меньше, чем WaitingTime (однако допустимо, чтоб задержка была больше, чем это значение).

*Примечание.* Обратите внимание, что поскольку количество ClientId может быть существенно большим, чем количество рабочих процессов worker, мы не можем здесь просто использовать sleep, иначе система будет постоянно находиться в простое, в то время как в таблице заданий будут иметься доступные для выполнения задачи.

Вопрос: предположим, что необходимо создать менеджер заданий, распределяющий их из таблицы Tasks среди рабочих процессов и контролирующий выполнение. Опишите, как бы вы стали технически реализовывать данную задачу? Какие инструменты AWS вы бы предложили для этого использовать?