

Fini d'hiberner  
avec  
**Hypersistence !**



Arthur MURILLO – *Ingénieur Concepteur Developpeur @SQLI*

12/02/2026

sqli

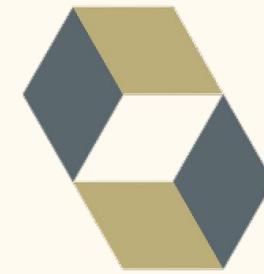




*30 – 31 octobre 2025*







**ORM** : mécanisme de traduction d'un objet en table relationnelle

**Entité** : classe qui représente une table en base de données

**Repository** : service qui encapsule la logique d'accès aux données (requêtes, filtres, persistance) pour une entité donnée

**Tout allait bien** durant le développement !

Mais lors des tests de charge sur des **données de production...**

Temps de chargement à la connexion : **14 secondes.**



**53%**

des utilisateur·trices partent après **3 s d'attente**

**1,9s**

temps moyen de **chargement** d'une page

Que se passait-il en **coulisses** ?

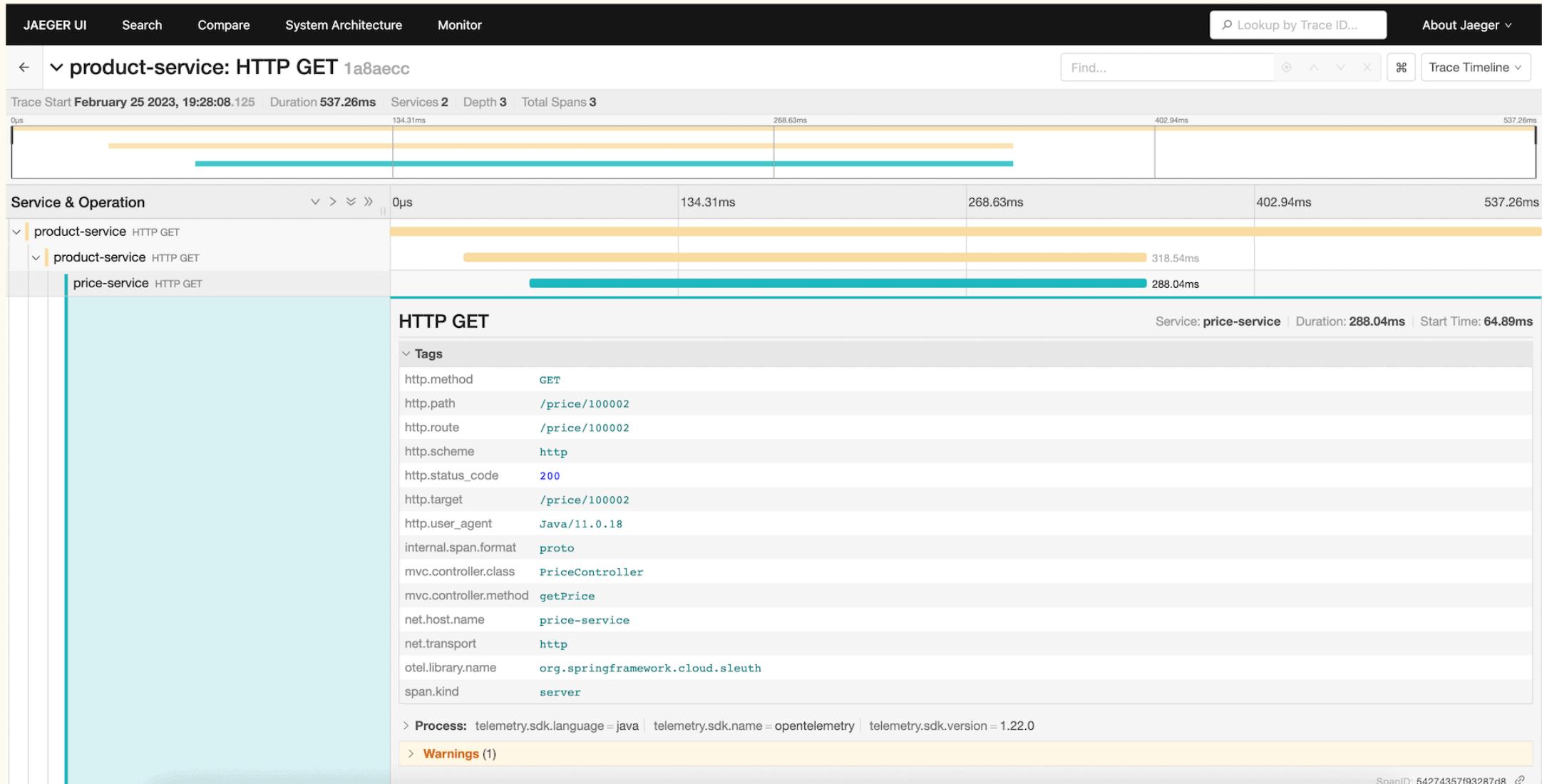
Pour trouver la source du problème, il fallait **plus d'informations**.

Il fallait donc améliorer **l'observabilité** de l'application.

L'observabilité, c'est rendre **visibles** des comportements **internes** à partir de données **externes** : logs, métriques, traces...

L'observabilité est simplifiée dans Java depuis **2021**

via **OpenTelemetry**, un standard open source.



On a donc beaucoup d'informations **supplémentaires** sur le processus

tout ça juste en **observant** 🚩 !

Et le constat était **sans appel...**

**Une seule requête** à un repository ralentissait tout le processus ! 

Hibernate effectue **une** requête pour récupérer une entité, puis une requête supplémentaire pour les **N** entités associées.

```
List<Client> clients = clientRepository.findAll();
```

Mais il y avait aussi plusieurs problèmes sur le **schéma de données...**

Utilisation excessive de Lazy loading sur des relations très utilisées...

```
@OneToMany(fetch = FetchType.LAZY)  
private List<Orders> orders;
```

... et utilisation excessive de Eager fetching sur des relations peu utilisées !

```
@OneToMany(fetch = FetchType.EAGER)  
private List<Compte> comptes;
```

Mauvaise optimisation du stockage de certains attributs via les annotations

```
@Enumerated(EnumType.STRING)  
private TypeCompte typeCompte;
```



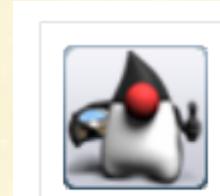
```
@Enumerated(EnumType.ORDINAL)  
private TypeCompte typeCompte;
```



Et encore bien **d'autres...**

Il fallait trouver un outil pour **répertorier** les **nombreux** problèmes...

...et **éviter d'en créer d'autres** dans le futur !



hypersistence-op  
timizer.jar

```
<dependency>
    <groupId>io.hypersistence</groupId>
    <artifactId>hypersistence-optimizer</artifactId>
    <version>2.10.0-trial</version>
    <classifier>jakarta</classifier>
</dependency>
```

```
@Configuration
public class HypersistenceConfiguration {
    @Bean
    public HypersistenceOptimizer hypersistenceOptimizer(EntityManagerFactory entityManagerFactory) {
        return new HypersistenceOptimizer(
            new JpaConfig(entityManagerFactory)
        );
    }
}
```

ERROR 21404 --- [...] Hypersistence Optimizer : **CRITICAL** - EagerFetchingEvent - The optimization event description is hidden in the Trial Version.

ERROR 21404 --- [...] Hypersistence Optimizer : **CRITICAL** - AssociationEvent - The optimization event description is hidden in the Trial Version.

ERROR 21404 --- [...] Hypersistence Optimizer : **CRITICAL** - AssociationEvent - The optimization event description is hidden in the Trial Version.

ERROR 21404 --- [...] Hypersistence Optimizer : **CRITICAL** - JdbcBatchSizeEvent - The optimization event description is hidden in the Trial Version.

..... (+ 43 others)



Hypersistence Optimizer : MINOR - **IdentityGeneratorEvent** -  
The [id] identifier attribute in the [org.springframework.samples.banking.entity.Compte] entity uses  
the [IdentityGenerator] strategy, which prevents Hibernate from enabling JDBC batch inserts.  
Since the database does not support the SEQUENCE identifier strategy, **you need to use plain JDBC or  
some other data access framework to batch insert statements.**  
<https://vladmirhalcea.com/hypersistence-optimizer/docs/user-guide/#IdentityGeneratorEvent>

**Problème + emplacement + solution + documentation !**

**83**

Optimisations

**100%**

Documentées

```
class HypersistenceTest {  
  
    public HypersistenceOptimizer hypersistenceOptimizer = hypersistenceOptimizer();  
  
    @Test  
    public void assertNoHypersistenceEvents() {  
        assertEquals(0, hypersistenceOptimizer.getEvents().size());  
    }  
}
```

En appliquant les (très nombreuses) recommandations :

réduction de **14 à 1,5 secondes** de la requête de connexion !

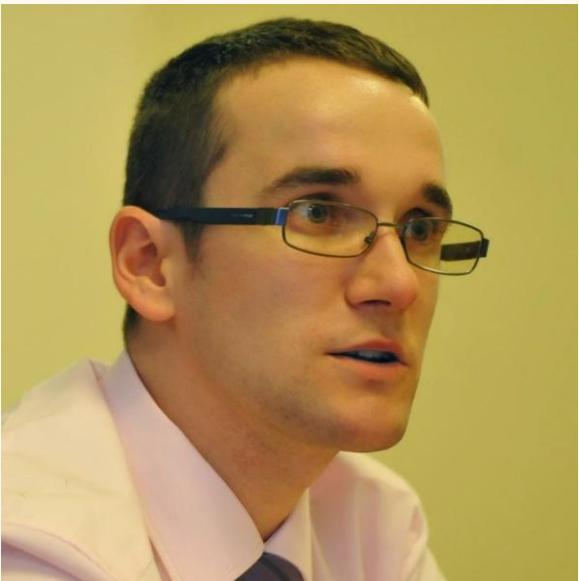
Marrant, ça ressemble à un certain logiciel, non ?



Ça ne dispense pas **d'aussi** améliorer son schéma de données !



Pour aller plus loin 



Vlad Mihalcea



<https://vladmihalcea.com/>

Merci ! 



openfeedback,