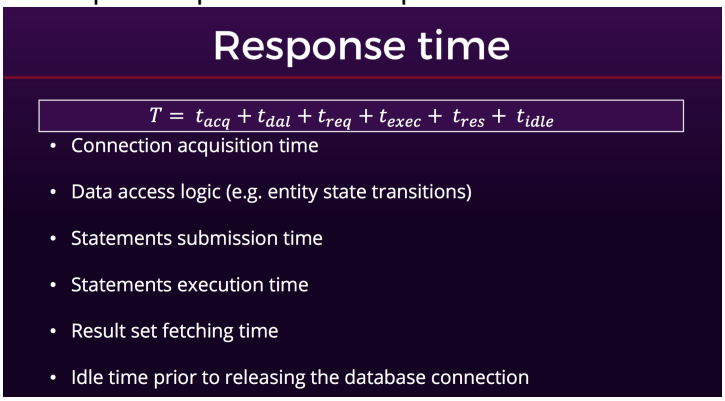
High-Performance Hibernate

Speaker : Vlad Mihalcea
Format : Conférence
Date : 21 avril 2016
Slides : http://fr.slideshare.net/VladMihalcea/high-performance-hibernate-devoxx-france

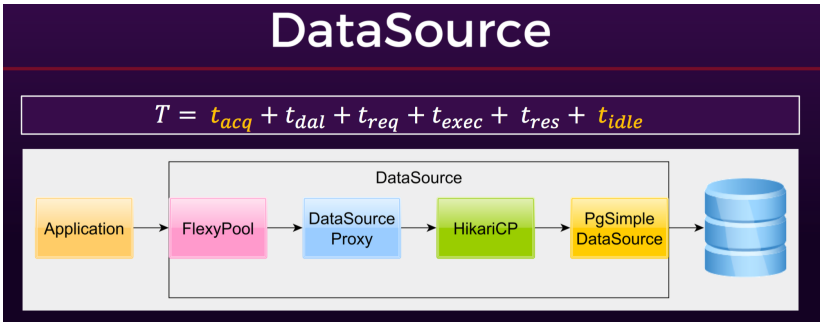
Vlad est Hibernate Developer Advocate et tient un blog : https://vladmihalcea.com/

D'après AppDynamics, « Plus de la moitié des problèmes de contentions viennent de la base de données ».

Le temps de réponse d'une requête est la somme des temps de réponse suivant :



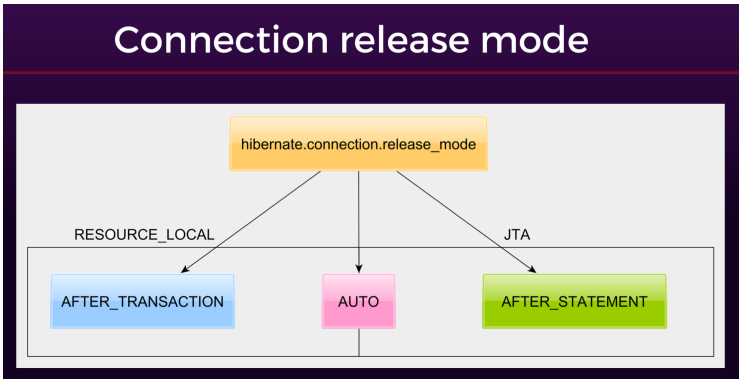
Connection providers



Plusieurs Connection providers sont fournis par Hibernate. Ex : HikariConnectionProvider, **DataSourceProxyConnectionProvider** (à privilégier), DriverMamangerConnectionProviderImpl.

FlexyPool est un framework qui permet de monitorer les connexions.

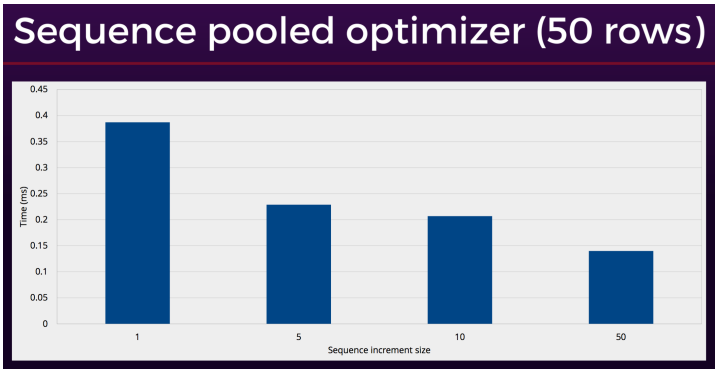
Plus courte est la transaction, meilleure sont ses performances.



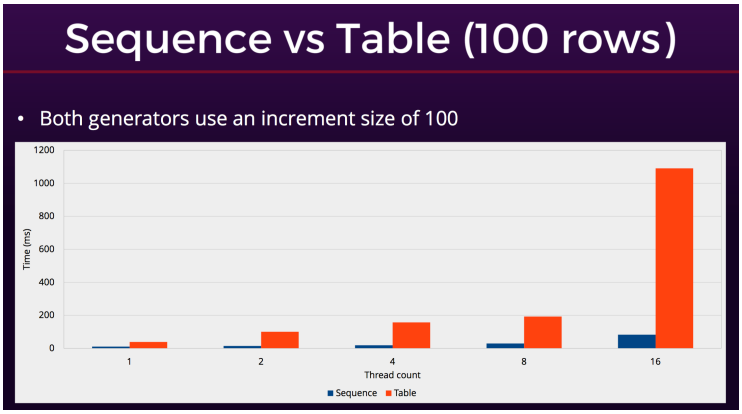
Connection release mode : triccky
hibernate.connection.release_mode
AFTER_TRANSACTION, AUTO, AFTER_STATEMEMENT (JPA)
Pour JPA, utiliser after_transaction expliciterment car plus performant.

Identifier generators

- Identify generator
  - Defaut for SQL Server et MySQL when using AUTO
  - Disables JDBC batch inserts
- Sequence generator
  - Default for Oracle and PostgreSQL when using auto
  - Paramétrage nécessaire de hibernate.id.new_generator_mappings à true avec Hibernate 5 (cf. photop)
  - Sequence pooled optimiser avec 50 rows
- Table generator
  - Pas une très bonne solution en soit. Moins performance que les séquences.

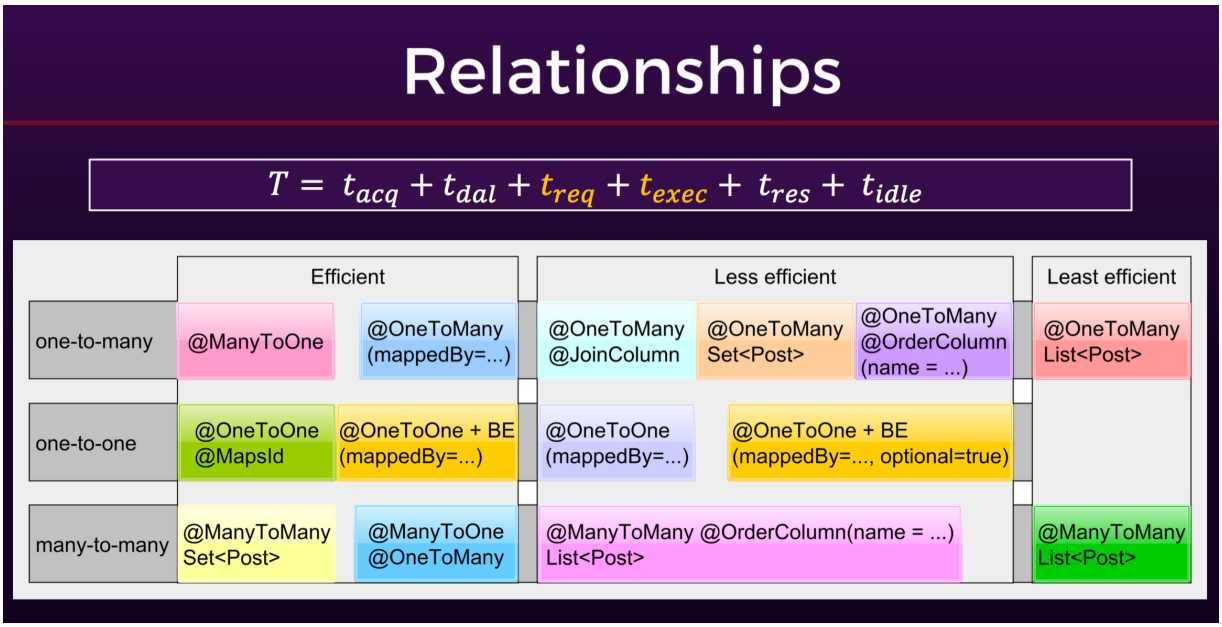


Identify vs Table sur 100 rows



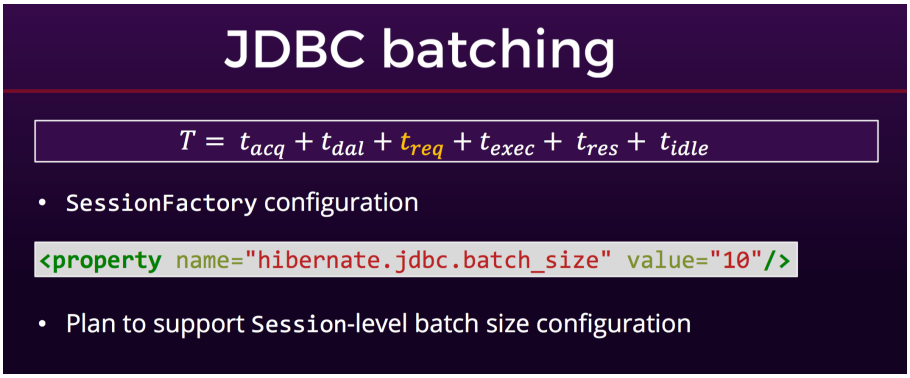
Dépend grandement du nombre de threads

Relationships

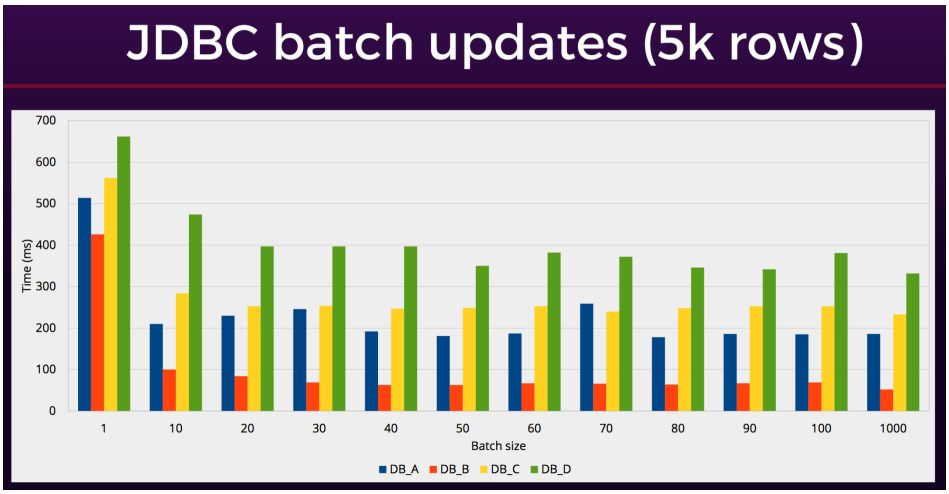


Least efficient : OneToMany List<Post>

JDBC Batching



Propriété *hibernate.jdbc.batch_size* pour une configuration globale
Il est prévu d'avoir une configuration plus granulaire, au niveau de la session
Pas de contradiction à son utilisation. Même avec une valeur de 10, on gagne en performance.



Cascading and batching : hibernate.order_inserts and order_updates à true
Dans une future version d'Hibernate ORM, il est prévu de supporter les delete.

Versionning and batching :
Désactivé par défaut sur Oracle10g dialects en Hibernate 3 et 4
Activé par défaut dans Hibernate 5.
hibernate.jdbc.batch_versioned_data = true

Fetching

Problème de n°1 : ramener trop de données.

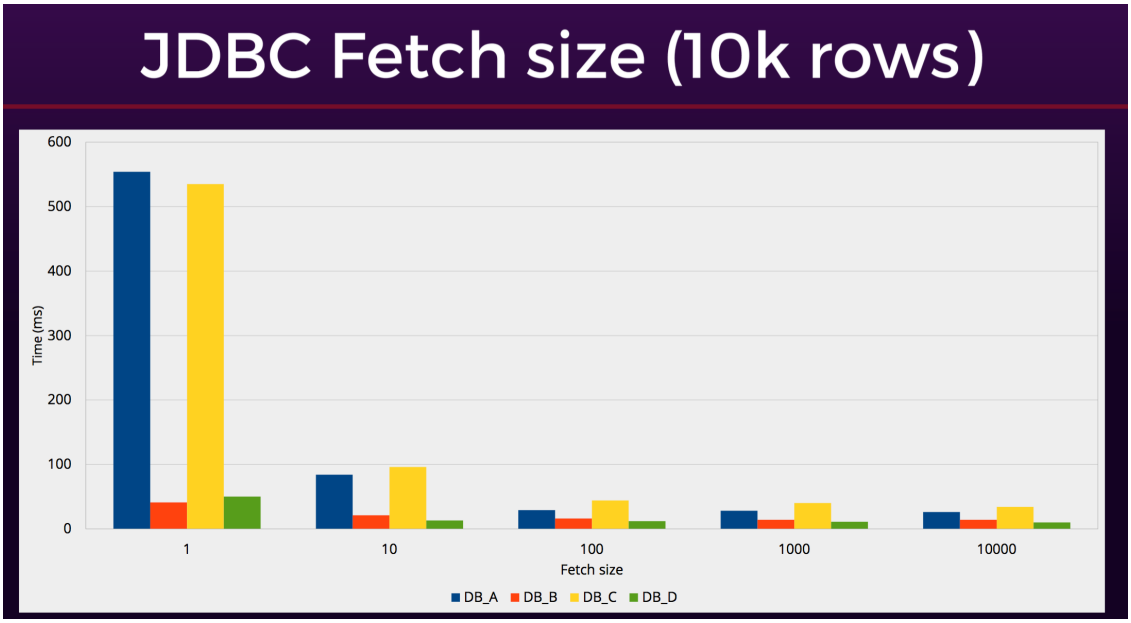
Plusieurs points à regarder

1. JDBC fetch size
2. JDBC ResultSet size

3. DTO vs Entity queries
4. Association fetching

JDBC fetch size

Sur Oracle, 10 par défaut.
SessionFactory settings : hibernate.jdbc.fetch_size à 100
Configuration locale à une requête avec QueryHints.HINT_FETCH_SIZE (Hint Hibernate et non Oracle)



Result set size limit

Limiter le nombre de lignes renvoyées avec setMaxResults()
Transactions plus courtes.
Ne fait pas partie du standard SQL, mais implémenté par toutes les bases.

Result set column count

Selecting all columns vs selection a custom projection
Le fameux SELECT * vs SELECT pc.version
Le développeur doit se poser la question : quelle data je dois avoir besoin ?

DTO projections recommandées pour :

- Read-only views
- Tree structures (Recursive CTE)
- Paginated Tables
- Analytics (Windows functions)

Entity quesries utilisées pour :

- Writing data
- Web flows/ Multi-request logical teansactions
- Application-level repeatable reads
- Detached entities/ PersistenceContextType.EXTENDED
- Optimistic concurrency control (e.g. version, dirty properties)

Fetching associations :

Valeur par défaut en fonction de l'annotation.
« You cannot turn an Eager into a Lazy. You can turn a lazy into an Eager »
Choix qu'on peut difficilement faire à la conception du modèle.
Conseil : utiliser lazt par défaut, puis les fetch directive de jPQL/Criteria, Entity Graph (@FetchProfile)
Utiliser Eager implique des temps de réponses supplémentaires.

Open Session in View anti-pattern

A ne pas utiliser.
Son utilisation avec MySQL entraine beaucoup de « pressure » sur la base.

Temporary Session anti-pattern

Ne pas utiliser le hibernate.enable_lazy_load_no_trans

Caching
1er conseil : tuner le Shared Buffer de la database

Caching strategies

Strategy	Cache type	Particularity
READ_ONLY	READ-THROUGH	Immutable
NONSTRICT_READ_WRITE	READ-THROUGH	Invalidation/ Inconsistency risk
READ_WRITE	WRITE-THROUGH	Soft Locks
TRANSACTIONAL	WRITE-THROUGH	JTA

DB Master et DB Slave

« There are only thow things hard in … » Citation de Phil Karlton

Stratégie de caching : le plus fiable est la stratégie TRANSACTIONAL de JTA. Mais READ_WRITE est en général le choix par défaut le plus approprié.

Collection and Query caches

- Complement entity caching
- Store only entity identifiers
- Read-Through
- Invalidation-based