Для чого потрібне кешування?

Програма

Кеш

База даних

Принцип кешування в Hibernate

В Hibernate немає власного кешу. Є лише можливість роботи із сторонніми реалізаціями кешу.

Приклади Hibernate-сумісних реалізацій кешу:

- **EHCache**
- **OSCache**
- **JBoss TreeCache**

Види кешування в Hibernate

- First-level Cache
- Second-level Cache
- Query Cache

Стратегії кешування

- Read-only
- Nonstrict Read-Write
- Read Write
- Transactional

First-level cache

Ввімкнений завжди. Працює на рівні сесії.

…

session.get(Entity.class, 1);

session.get(Entity.class,1); // **Спрацює кеш**

...

Second-level cache

По замовчуванню вимкнений. Працює на рівні SessionFactory.

…

session.get(Entity.class, 1);

session.close(); // **Закрили сесію**

Session = sessionFactory.openSession(); // **Нова сесія**

session.get(Entity.class,1); // **Спрацює кеш другого рівня**

...

Ввімкнення second-level cache

В hibernate.hbm.xml:

```
<property
name="hibernate.cache.region.factory_class">net.sf.ehcache.hibernate.Singleton
EhCacheRegionFactory</property>
<property name="hibernate.cache.use_second_level_cache">true</property>
<property name="hibernate.cache.use_query_cache">true</property>
```

Ввімкнення кешування для конкретного класу:

```
@Entity
@Table(name="MY_ENTITY")
@Cache(usage = CacheConcurrencyStrategy.READ_ONLY)
class MyEntity {
...
```

Second-level cache в xml

```
...
<meta attribute="class-description">
            Institute.
 </meta>

<cache usage="read-write"/>
...
```

Кешування залежностей класу

```
@Table (name=”MY_TABLE”)
@Entity
@Cache
Class MyEntity {
    @Cache // Явно вказуємо на кешування
    private List<AnotherEntity> children;
```

Налаштування в ehcache.xml

- регіони кешу

- maxEntriesLocalHeap,

- timeToIdleSeconds

-timeToLiveSeconds

-eternal

-overflowToDisk

Cache Persistence

- none
- localRestartable
- localTempSwap
- distributed

Кешування запитів

createQuery("select id from Entity").**setCacheable(true)**.list()**;**

Ручне управління кешем

- evict()
- clear();
- contains();
- flush();

Інвалідація кешу

Є проблема інвалідації кешу. Якщо використовувати, наприклад, Non-Strict Read Write, є **ризик отримати застарілі дані**.

Пакетна обробка даних (batch)

Ризик **OutOfMemoryException**:

```
for(int i = 0; i < 10000000; i++) {
        Institute institute = new Institute("Institute " + i);
        session.save(institute);
}
```

Вирішення проблеми – крок 1

1) **Розмір batch size для JDBC-драйвера:**

<property name="**hibernate.jdbc.batch_size**">**20**</property>

<property name="**hibernate.order_inserts**">**true**</property>

<property name="**hibernate.order_updates**">**true**</property>

Вирішення проблеми – крок 2

2) **Періодично записуємо дані в БД:**

```
for(int i = 0; i < 10000; i++) {
        Institute institute = new Institute("Institute " + i);
        session.save(institute);

         if (i%20 == 0) { // Кожні 20 елементів
            session.flush();
            session.clear();
        }
    }
```

Швидкість і надійність

Кешування пришвидшує роботу програми. Але лише у випадку його правильного налаштування. Ціна за це – підвищене споживання оперативної пам’яті.

Пакетна обробка даних (batch insert/update) часто теж пришвидшує роботу.

У випадку великих об’ємів даних для запису/оновлення пакетна обробка – єдиний спосіб уникнути **OutOfMemoryException**.