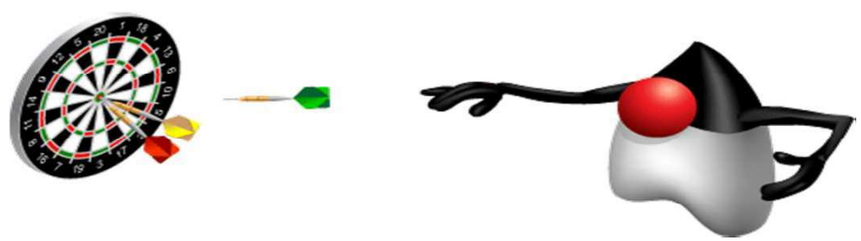
10

Hibernate Caching

Objectives

After completing this lesson, you should be able to do the following:

- Using Stored Procedures
- First level Caching
- Second Level Caching



Calling Stored Procedures

- **For querying, similiar syntax and process as named sql-query**
  - Defined inside or outside the class tags in the mapping file
  - If returning a value, can set an alias and return type
- **For insert, update, or delete, must be defined inside the class tag**
  - Overrides the default implementation for those events
  - Column order is random and unintuitive

    Documentation says to look at the SQL log output to see what order Hibernate lists the columns
- **Must set the 'callable' attribute**

Stored Procedure Setup

```
<class name="courses.hibernate.vo.EBill" table="EBILL">
   ...
   <!-- Calling procedure to execute the insert -->
   <sql-insert callable="true" check="param">
     {call create_ebill(?, ?, ?, ?, ?, ?,?, ?, ?, ?, ?)}
   </sql-insert>
</class>

<!-- named SQL query, but with callable
      and return value set -->
<sql-query name="getEbills" callable="true">
   <return alias="ebill"
           class="courses.hibernate.vo.EBill"/>
   { ? = call get_ebills() }
</sql-query>
```

DDL Generation

- Hibernate provides a tool to automatically generate database objects based on a domain model, or a domain model based on an already existing database.
- hbm2ddl
- Used through ANT tasks or with Hibernate configuration

Hibernate Configuration

```
<!-- in the Hiberante.cfg.xml file -->
<session-factory>
  <property name="hibernate.hbm2ddl.auto">
    create|create-drop
  </property>
  ...
</sessionFactory>
```

```
// programmatically
Configuration cfg =
  new Configuration().configure();
SchemaUpdate schemaUpdate =
  new SchemaUpdate(cfg);
schemaUpdate.execute();
```

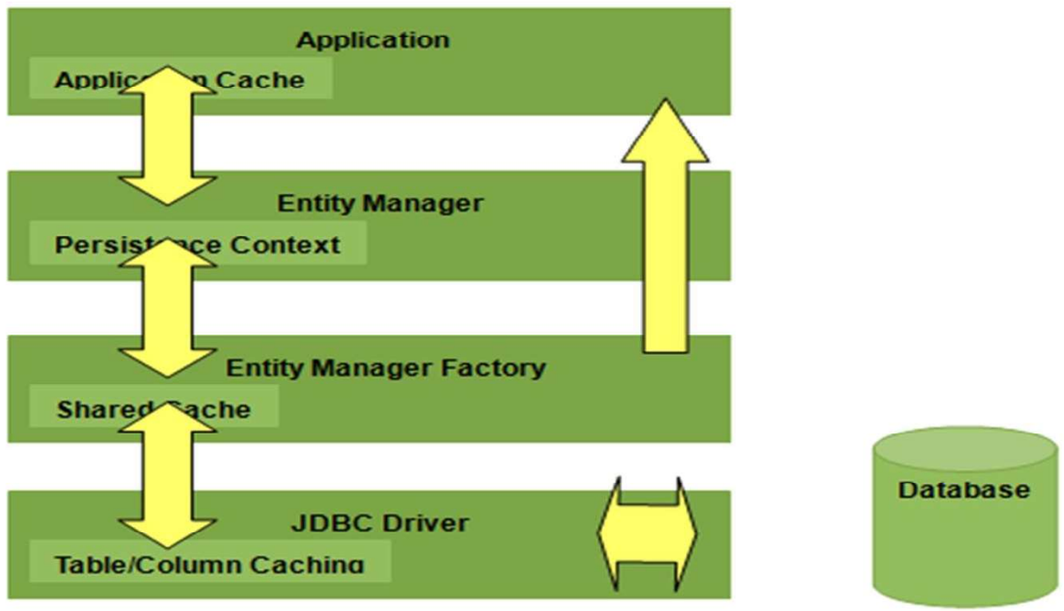
Caching in Hibernate

- Caching is one of the important features implemented by an application for better performance.
- In an ORM perspective, data retrieved from a database is cached in memory or to disk so that there is no need to make a call to the database for every request.

Using Second Level Caching in Application

- Caching structurally implies a temporary store to keep data for quicker access later on.
- Second level shared cache is an auxiliary technique, mainly used in JPA, to enhance performance; it is used especially during large inflow, outflow of data between the database and the application.
- Caching also reduces search time when the searched entity is already in the cache, otherwise it is fetched from the database to serve the purpose.
- However, when a subsequent search query is fired it takes little time as the searched entity is already in the cache. Entity Manager stores entities for almost every CRUD operation in this shared cache before flushing the content to the database.

➢Second level cache is complementary to its first level counterpart. In fact, to get an overview of caching in particular, there are several layers of caching in JPA

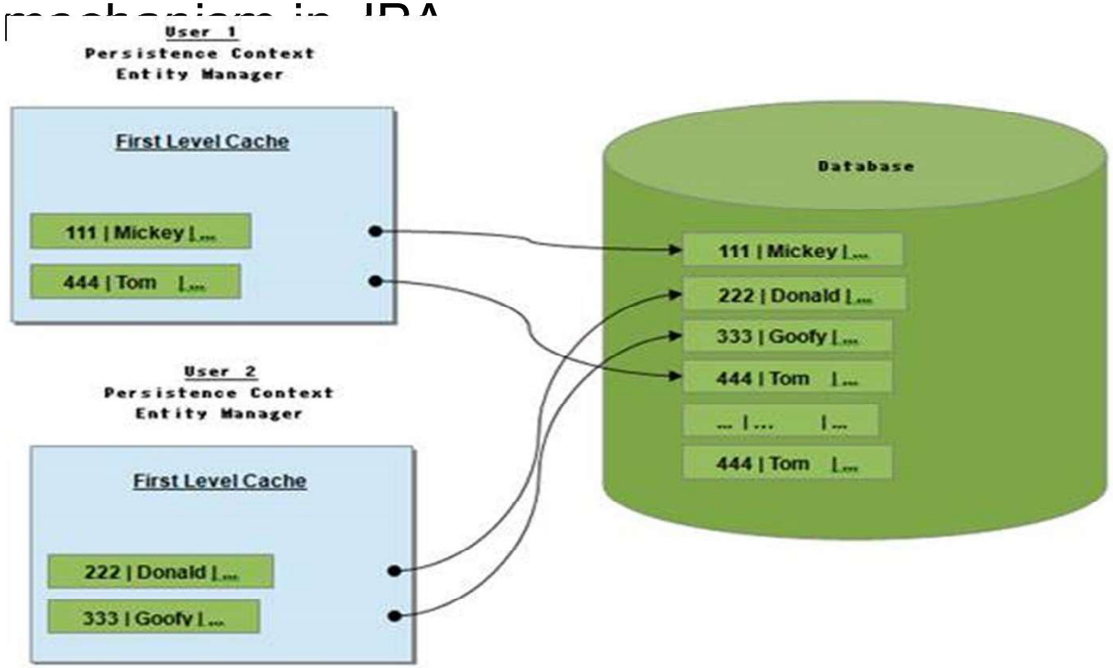


First Level Cache

- Entity Manager always maintains a cache called the first level cache, so it makes sense to call another shared caching technique in addition to first – a second level cache.
- In first level cache CRUD operations are performed per transaction basis to reduce the number of queries fired to the database.
- That is, an entity modified several times within the same transaction is done in the cache only, modification at the database level is slated until final UPDATE statement is fired at the end of the transaction.

Cont…

- JPA entities are cached at the persistence context level and guarantees that there will be one object instance per persistence context for a specific row of a database table.
- Concurrent transactions affecting the same row are managed by applying an appropriate locking mechanism in JPA



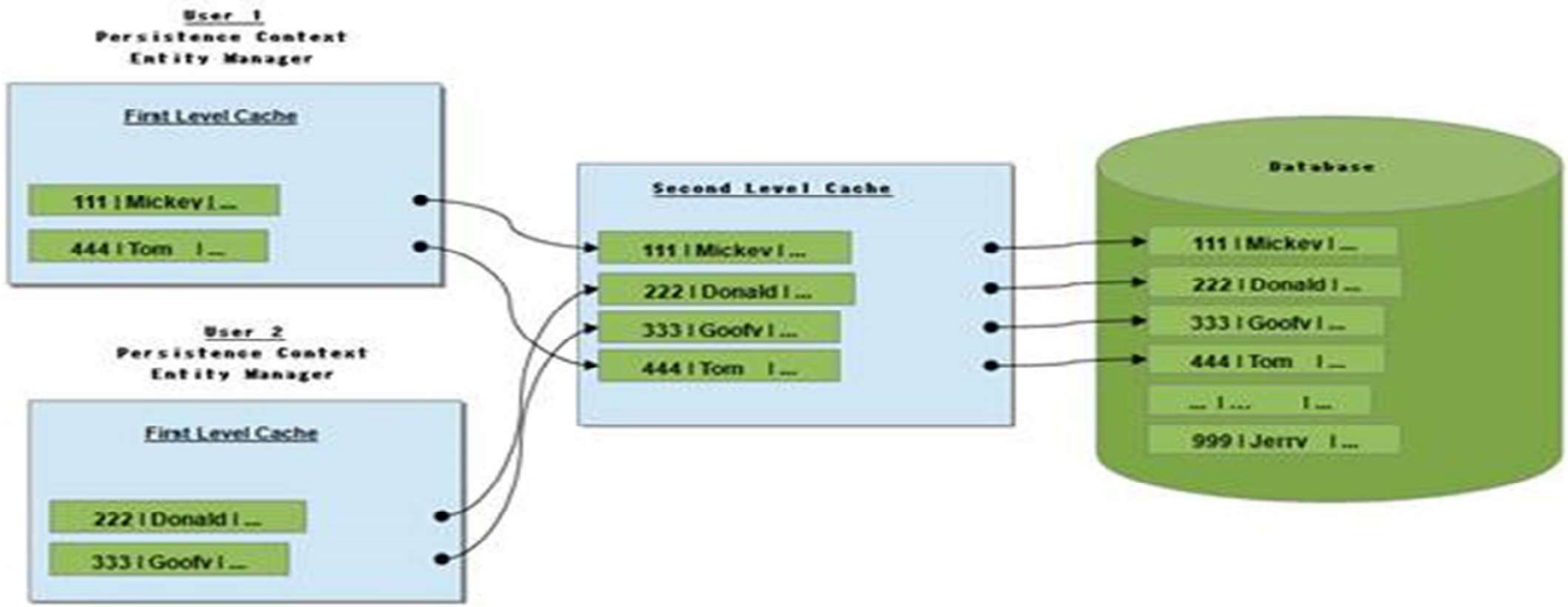
Second Level Cache

- This level of cache emerged more due to performance reasons than absolute necessity; in fact adding more shared cache increases the possibility of the problem of a stale read.
- Second level cache sits between Entity Manager and the database.
- Persistence context shares the cache, making the second level cache available throughout the application.
- Database traffic is reduced considerably because entities are loaded in to the shared cache and made available from there.

Second Level Caching Benefits

- Persistence provider manages local store of entity data
- Leverages performance by avoiding expensive database calls
- Data are kept transparent to the application
- CRUD operation can be performed through normal entity manager functions
- Application can remain oblivious of the underlying cache and do its job inadvertently

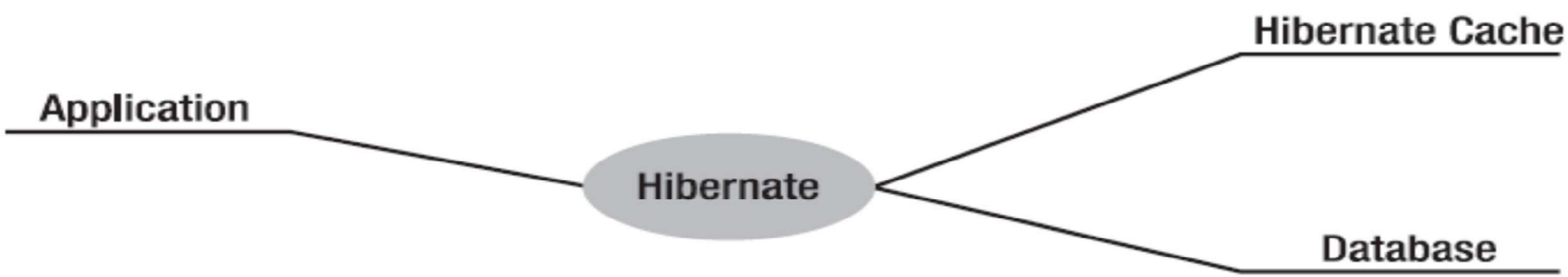
Diagrammatical Representation



2nd Level Cache

- **Performance increase for objects with a much greater READ to WRITE ratio**
- **Great for reference or immutable objects**
- **Not advised for**
  - Frequently changing data
  - Tables accessed from other applications
- **Requires a cache strategy and pluggable cache provider**

- A *cache* is a local copy of the information from the database that may be used to avoid a database call whenever:
  - The application performs a lookup by identifier.
  - The persistence layer resolves an association or collection lazily.



Level of Caching

- Hibernate provides a way to configure caching at the
  - class level
  - collection level
  - the query result-set level

Scope of Caching : *Transaction*

A transaction is defined as a unit of work.

- How does caching affect data in a transaction?
  - If data is fetched in one session, and the same query is executed again in that same session before that unit of work is completed, is that data stored in memory/disk?
- Does this avoid a call to the database?
  - Yes. This data is stored in memory/disk by default, so the call to the database for the second query is avoided. Because a unit of work is for one request, this data isn't shared or accessed concurrently.

Level of Cache : *Process scope*

- Data is cached across sessions or across units of work.
- If a query is executed in one session and the same query is executed in a second session after closing the first, this data is cached so that the second call to the database is avoided.
- Because two requests can access the same data in these two sessions, this data is accessed concurrently. You should use this scope with caution.

Level of Cache : *Cluster scope*

- Data is shared between multiple processes on the same machine or different machines in a cluster.

Hibernate has a two-level cache architecture

- The first-level cache is the *persistence context cache*.
  - This is at the unit-of-work level.
  - It corresponds to one session in Hibernate for a single request and is by default enabled for the Hibernate session.
- The second-level cache is either at the *process scope or the cluster scope*.
  - This is the cache of the state of the persistence entities.
  - A cache-concurrency strategy defines the transaction isolation details for a particular item of data, and the cache provider represents the physical cache implementation.

Cache Providers

- *EHCache:* This is an open source standards-based cache that supports processscope cache. It can cache in memory or to disk, and it supports query cache. EHCache Distributed supports the cluster-scope cache. It has its ownconfiguration file (**ehcache.xml**) where all the required parameters are set forcaching. You can check out the tutorial at **http://ehcache.org/**.
- *OSCache:* This is also open source caching provider that writes to memory and disk. It also has cluster cache support. The URL **www.opensymphony.com/oscache/** has documentation.

- *SwarmCache:* This is a simple but effective distributed cache. It uses IP multicast to communicate with any number of hosts in a distributed system. The documentation is at **http://swarmcache.sourceforge.net**.
- *JBoss cache:* This is a distributed cache system. It replicates data across different nodes in a distributed system. State is always kept in synch with other servers in the system. The documentation is at **www.jboss.org/jbosscache/**.

Setting up the Cache

- **Four caching strategies; each level increases performance and risk of stale data**
  - Transactional
    - Slowest of caching, but most risk free
  - Read-write
  - Nonstrict-read-write
  - Read-only
    - Fastest performance, but most risky.
    - Use if the data never changes
- **Four cache providers are built into Hibernate**
  - EHCache: Simple process scope cache in a single JVM
  - OSCache: Richer set of expiration policies and support
  - SwarmCache: Cluster cache, but doesn't suppor 'Query Cache'
  - JBoss Cache: Fully transactional, replicated clustering

ehcache.xml

```
<ehcache>
   <diskStore path="java.io.tmpdir" />
   <defaultCache maxElementsInMemory="10000"
                 eternal="false"
                 timeToIdleSeconds="120"
                 timeToLiveSeconds="120"
                 overflowToDisk="true" />

   <!-- setup special rules for Account objects -->
   <cache name="courses.hibernate.vo.Account"
          maxElementsInMemory="1000"
          eternal="false"
          timeToIdleSeconds="300"
          timeToLiveSeconds="600"
          overflowToDisk="false" />
</encache>
```

Configuring Hibernate Cache

```
<session-factory>
   ...
   <property name="cache.provider_class">
     org.hibernate.cache.EhCacheProvider
   </property>

   <property name="cache.use_second_level_cache">
     true
   </property>
   ...
</session-factory>
```

Mapping File

```
<class name="courses.hibernate.vo.Account"
       table="ACCOUNT">
  <cache usage="read-write" />
  <id name="accountId" column="ACCOUNT_ID">
    <generator class="native" />
  </id>
  <discriminator column="ACCOUNT_TYPE" type="string" />
  <version name="version" column="VERSION"
           type="long"    access="field" />
  <property name="creationDate" column="CREATION_DATE"
            type="timestamp"    update="false" />
  ...

</class>
```

Summary

In this lesson, you should have learned how to:

- Using Stored Procedures
- First Level Caching
- 2nd Level Caching