Improving the Performance of Java Persistence API Applications By Setting a Second-Level Cache

This chapter explains how to modify the second-level cache mode settings to improve the performance of applications that use the Java Persistence API.

The following topics are addressed here:

Overview of the Second-Level Cache

A second-level cache is a local store of entity data managed by the persistence provider to improve application performance.

A second-level cache helps improve performance by avoiding expensive database calls, keeping the entity data local to the application.

A second-level cache is typically transparent to the application, as it is managed by the persistence provider and underlies the persistence context of an application.

That is, the application reads and commits data through the normal entity manager operations without knowing about the cache.

Note - Persistence providers are not required to support a second-level cache.

Portable applications should not rely on support by persistence providers for a second-level cache.

The second-level cache for a persistence unit may be configured to one of several second-level cache modes.

The following cache mode settings are defined by the Java Persistence API:

Table 38-1 Cache Mode Settings for the Second-Level Cache

Cache Mode Setting	Description
`ALL`	All entity data is stored in the second-level cache for this persistence unit.
`NONE`	No data is cached in the persistence unit. The persistence provider must not cache any data.

`ENABLE_SELECTIVE`	Enable caching for entities that have been explicitly set with the `@Cacheable` annotation.
`DISABLE_SELECTIVE`	Enable caching for all entities except those that have been explicitly set with the `@Cacheable(false)` annotation.
`UNSPECIFIED`	The caching behavior for the persistence unit is undefined. The persistence provider's default caching behavior will be used.

One consequence of using a second-level cache in an application is that the underlying data may have changed in the database tables, but the value in the cache has not, a circumstance called a stale read.

Stale reads may be avoided by changing the second-level cache to one of the cache mode settings, controlling which entities may be cached (described in Controlling Whether Entities May Be Cached),

or changing the cache's retrieval or store modes (described in Setting the Cache Retrieval and Store Modes).

Which strategies best avoid stale reads are application dependent.

Controlling Whether Entities May Be Cached

The `javax.persistence.Cacheable` annotation is used to specify that an entity class, and any subclasses, may be cached when using the `ENABLE_SELECTIVE` or `DISABLE_SELECTIVE` cache modes.

Subclasses may override the `@Cacheable` setting by adding a `@Cacheable` annotation and changing the value.

To specify that an entity may be cached, add a `Cacheable` annotation at the class level:

```
@Cacheable
@Entity
public class Person {...}
```

By default, the `@Cacheable` annotation is `true`.

The following example is equivalent:

```
@Cacheable(true)
@Entity
public class Person{...}
```

To specify that an entity must not be cached, add a `@Cacheable` annotation and set it to `false`:

```
@Cacheable(false)
@Entity
public class OrderStatus {...}
```

When the `ENABLE_SELECTIVE` cache mode is set, the persistence provider will cache any entities that have a `@Cacheable(true)` annotation and any subclasses of that entity that have not been overridden.

The persistence provider will not cache entities that have `@Cacheable(false)` or have no `@Cacheable` annotation.

That is, the `ENABLE_SELECTIVE` mode will only cache entities that have been explicitly marked for the cache using the `@Cacheable` annotation.

When the `DISABLE_SELECTIVE` cache mode is set, the persistence provider will cache any entities that do not have a `@Cacheable(false)` annotation.

Entities that do not have a `@Cacheable` annotation, and entities with a `@Cacheable(true)` annotation will be cached.

That is, the `DISABLE_SELECTIVE` mode will cache all entities that have not been explicitly prevented from being cached.

If the cache mode is set to `UNDEFINED`, or is left unset, the behavior of entities annotated with `@Cacheable` is undefined.

If the cache mode is set to `ALL` or `NONE`, the value of the `@Cacheable` annotation is ignored by the persistence provider.

Specifying the Cache Mode Settings to Improve Performance

To adjust the cache mode settings for a persistence unit, specify one of the cache modes as the value of the `shared-cache-mode` element in the `persistence.xml` deployment descriptor:

```
<persistence-unit
name="examplePU"
transaction-type="JTA">
<provider>
org.eclipse.persistence.jpa.
PersistenceProvider
</provider>
<jta-data-source>
jdbc/__default
</jta-data-source>
```

```
<shared-cache-mode>
DISABLE_SELECTIVE
</shared-cache-mode>
</persistence-unit>
```

Note - Because support for a second-level cache is not required by the Java Persistence API specification, setting the second-level cache mode in `persistence.xml` will have no affect when using a persistence provider that does not implement a second-level cache.

Alternately, the shared cache mode may be specified by setting the `javax.persistence.sharedCache.mode` property to one of the shared cache mode settings:

```
EntityManagerFactor emf =
Persistence.
createEntityManagerFactory
("myExamplePU",
new Properties().add
("javax.persistence.sharedCache.mode"
, "ENABLE_SELECTIVE"));
```

Setting
the Cache Retrieval and Store Modes

If the second-level cache has been enabled for a persistence unit by setting the shared cache mode, the behavior of the second-level cache can be further modified by setting the `javax.persistence.cache.retrieveMode` and `javax.persistence.cache.storeMode` properties.

These properties may be set at the persistence context level by passing the property name and value to the `EntityManager.setProperty` method, or may be set on

a per-`EntityManager` operation (`EntityManager.find` or `EntityManager.refresh`) or per-query level.

Cache Retrieval Mode

The cache retrieval mode, set by the `javax.persistence.retrieveMode` property, controls how data is read from the cache for calls to the `EntityManager.find` method and from queries.

The `retrieveMode` property can be set to one of the constants defined by the `javax.persistence.CacheRetrieveMode` enumerated type, either `USE` (the default) or `BYPASS`.

When set to `USE`, data is retrieved from the second-level cache, if available.

If the data is not in the cache, the persistence provider will read it from the database.

When set to `BYPASS`, the second-level cache is bypassed and a call to the database is made to retrieve the data.

Cache Store Mode

The cache store mode, set by the `javax.persistence.storeMode` property, controls how data is stored in the cache.

The `storeMode` property can be set to one of the constants defined by the `javax.persistence.CacheStoreMode` enumerated type, either `USE` (the default), `BYPASS`, or `REFRESH`.

When set to `USE` the cache data is created or updated when data is read from or committed to the database.

If data is already in the cache, setting the store mode to `USE` will not force a refresh when data is read from the database.

When the store mode is set to `BYPASS`, data read from or committed to the database is not inserted or updated in the cache.

That is, the cache is unchanged.

When the store mode is set to `REFRESH` the cache data is created or updated when data is read from or committed to the database, and a refresh is forced on data in the cache upon database reads.

Setting the Cache Retrieval or Store Mode

To set the cache retrieval or store mode for the persistence context, call the `EntityManager.setProperty` method with the property name and value pair:

```
EntityManager em = ...;
em.setProperty("javax.persistence.c
ache.storeMode", "BYPASS");
```

To set the cache retrieval or store mode when calling the `EntityManger.find` or `EntityManager.refresh` methods, first create a `Map<String, Object>` instance and add a name/value pair as follows:

```
EntityManager em = ...;
Map<String, Object> props =
new HashMap<String, Object>();
```

```
props.put(
"javax.persistence.cache.retrieveMode",
"BYPASS");
String personPK = ...;
Person person = em.find
(Person.class, personPK, props);
```

Note - The cache retrieve mode is ignored when calling the `EntityManager.refresh` method, as calls to `refresh` always result in data being read from the database, not the cache.

To set the retrieval or store mode when using queries, call the `Query.setHint` or `TypedQuery.setHint` methods, depending on the type of query:

```
EntityManager em = ...;
CriteriaQuery<Person> cq = ...;
TypedQuery<Person> q =
em.createQuery(cq);
q.setHint(
"javax.persistence.cache.storeMode"
, "REFRESH");
...
```

Setting the store or retrieve mode in a query or when calling the `EntityManager.find` or `EntityManager.refresh` methods overrides the setting of the entity manager.

Controlling the Second-Level Cache Programmatically

The `javax.persistence.Cache` interface defines methods for interacting with the second-level cache programmatically.

The `Cache` interface defines methods to check whether a particular entity has cached data, to remove a particular entity from the cache, to remove all instances (and instances of subclasses) of an entity class from the cache, and to clear the cache of all entity data.

Note - If the second-level cache has been disabled, calls to the `Cache` interface's methods have no effect, except for `contains`, which will always return `false`.

Checking Whether An Entity's Data is Cached

Call the `Cache.contains` method to find out whether a given entity is currently in the second-level cache.

The `contains` method returns `true` if the entity's data is cached, and `false` if the data is not in the cache.

```
EntityManager em = ...;
Cache cache =
em.getEntityManagerFactory().
getCache();
String personPK = ...;
if (cache.contains
(Person.class, personPK)) {
 // the data is cached
} else {
 // the data is NOT cached
}
```

Removing an Entity from the Cache

Call one of the `Cache.evict` methods to remove a particular entity or all entities of a given type from the second-level cache.

To remove a particular entity from the cache, call the `evict` method and pass in the entity class and the primary key of the entity:

```
EntityManager em = ...;
Cache cache =
em.getEntityManagerFactory().
getCache();
String personPK = ...;
cache.evict(Person.class,
personPK);
```

To remove all instances of a particular entity class, including subclasses, call the `evict` method and specify the entity class:

```
EntityManager em = ...;
Cache cache =
em.getEntityManagerFactory().
getCache();
cache.evict(Person.class);
```

All instances of the `Person` entity class will be removed from the cache.

If the `Person` entity has a subclass, `Student`, calls to the above method will remove all instances of `Student` from the cache as well.

Removing All Data from the Cache

Call the `Cache.evictAll` method to completely clear the second-level cache:

```
EntityManager em = ...;
Cache cache =
em.getEntityManagerFactory().
getCache();
cache.evictAll();
```