

Lock Modes

The application may increase the level of locking for an entity by specifying the use of lock modes. Lock modes may be specified to increase the level of optimistic locking or to request the use of pessimistic locks.

The use of optimistic lock modes causes the persistence provider to check the version attributes for entities that were read (but not modified) during a transaction as well as for entities that were updated.

The use of pessimistic lock modes specifies that the persistence provider is to immediately acquire long-term read or write locks for the database data corresponding to entity state.

The lock mode for an entity operation may be set by specifying one of the lock modes defined in the `javax.persistence.LockModeType` enumerated type, listed in [Table 37-1](#).

Table 37-1 Lock Modes for Concurrent Entity Access

Lock Mode	Description
OPTIMISTIC	Obtain an optimistic read lock for all entities with version attributes.
OPTIMISTIC_FORCE_INCREMENT	Obtain an optimistic read lock for all entities with version attributes, and increment the version attribute value.
PESSIMISTIC_READ	Immediately obtain a long-term read lock on the data to prevent the data from being modified or deleted. Other transactions may read the data while the lock is maintained, but may not modify or delete the data. The persistence provider is permitted to obtain a database write lock when a read lock was requested, but not vice versa.
PESSIMISTIC_WRITE	Immediately obtain a long-term write lock on the data to prevent the data from being read, modified, or deleted.
PESSIMISTIC_FORCE_INCREMENT	Immediately obtain a long-term lock on the data to prevent the data from being modified or deleted, and increment the version attribute of versioned entities.
READ	A synonym for <code>OPTIMISTIC</code> . Use of <code>LockModeType.OPTIMISTIC</code> is to be preferred for new applications.
WRITE	A synonym for <code>OPTIMISTIC_FORCE_INCREMENT</code> . Use of <code>LockModeType.OPTIMISTIC_FORCE_INCREMENT</code> is to be preferred for new applications.
NONE	No additional locking will occur on the data in the database.

Setting the Lock Mode

The lock mode may be specified by one of the following techniques:

- Calling the `EntityManager.lock` and passing in one of the lock modes:

```
EntityManager em = ...;
Person person = ...;
em.lock(person, LockModeType.OPTIMISTIC);
```

- Calling one of the `EntityManager.find` methods that takes the lock mode as a parameter:

```
EntityManager em = ...;
String personPK = ...;
Person person = em.find(Person.class, personPK,
    LockModeType.PESSIMISTIC_WRITE);
```

- Calling one of the `EntityManager.refresh` methods that takes the lock mode as a parameter:

```
EntityManager em = ...;
String personPK = ...;
```

```

Person person = em.find(Person.class, personPK);
...
em.refresh(person, LockModeType.OPTIMISTIC_FORCE_INCREMENT);

```

- Calling the `Query.setLockMode` or `TypedQuery.setLockMode` method, passing the lock mode as the parameter:

```

Query q = em.createQuery(...);
q.setLockMode(LockModeType.PESSIMISTIC_FORCE_INCREMENT);

```

- Adding a `lockMode` element to the `@NamedQuery` annotation:

```

@NamedQuery(name="lockPersonQuery",
    query="SELECT p FROM Person p WHERE p.name LIKE :name",
    lockMode=PESSIMISTIC_READ)

```

Using Pessimistic Locking

Versioned entities as well as entities that do not have version attributes can be locked pessimistically.

To lock entities pessimistically, set the lock mode to `PESSIMISTIC_READ`, `PESSIMISTIC_WRITE`, or `PESSIMISTIC_FORCE_INCREMENT`.

If a pessimistic lock cannot be obtained on the database rows, and the failure to lock the data results in a transaction rollback, a `PessimisticLockException` is thrown. If a pessimistic lock cannot be obtained, but the locking failure doesn't result in a transaction rollback, a `LockTimeoutException` is thrown.

Pessimistically locking a version entity with `PESSIMISTIC_FORCE_INCREMENT` results in the version attribute being incremented even if the entity data is unmodified. When pessimistically locking a versioned entity, the persistence provider will perform the version checks that occur during optimistic locking, and if the version check fails, an `OptimisticLockException` will be thrown. Attempting to lock a non-versioned entity with `PESSIMISTIC_FORCE_INCREMENT` is not portable and may result in a `PersistenceException` if the persistence provider doesn't support optimistic locks for non-versioned entities. Locking a versioned entity with `PESSIMISTIC_WRITE` results in the version attribute being incremented if the transaction was successfully committed.

Pessimistic Locking Timeouts

The length of time in milliseconds the persistence provider should wait to obtain a lock on the database tables may be specified using the `javax.persistence.lock.timeout` property. If the time it takes to obtain a lock exceeds the value of this property, a `LockTimeoutException` will be thrown, but the current transaction will not be marked for rollback. If this property is set to 0, the persistence provider should throw a `LockTimeoutException` if it cannot immediately obtain a lock.

Note - Portable applications should not rely on the setting of `javax.persistence.lock.timeout`, as the locking strategy and underlying database may mean that the timeout value cannot be used. The value of `javax.persistence.lock.timeout` is a hint, not a contract.

This property may be set programmatically by passing it to the `EntityManager` methods that allow lock modes to be specified, the `Query.setLockMode` and `TypedQuery.setLockMode` methods, the `@NamedQuery` annotation, and as a property to the `Persistence.createEntityManagerFactory` method. It may also be set as a property in the `persistence.xml` deployment descriptor.

If `javax.persistence.lock.timeout` is set in multiple places, the value will be determined in the following order:

1. The argument to one of the `EntityManager` or `Query` methods.
2. The setting in the `@NamedQuery` annotation.
3. The argument to the `Persistence.createEntityManagerFactory` method.
4. The value in the `persistence.xml` deployment descriptor.