Overview of Entity Locking and Concurrency

Entity data is **concurrently accessed** if the data in a data source is accessed at the same time by multiple applications. Special care must be taken to ensure that the underlying data's integrity is preserved when accessed concurrently.

When data is updated in the database tables in a transaction, the persistence provider assumes the database management system will hold short-term read locks and long-term write locks to maintain data integrity. Most persistence providers will delay database writes until the end of the transaction, except when the application explicitly calls for a flush (that is, the application calls the`EntityManager.flush` method or executes queries with the flush mode set to `AUTO`).

By default, persistence providers use **optimistic locking**, where, before committing changes to the data, the persistence provider checks that no other transaction has modified or deleted the data since the data was read. This is accomplished by a version column in the database table, with a corresponding version attribute in the entity class. When a row is modified, the version value is incremented. The original transaction checks the version attribute, and if the data has been modified by another transaction, a`javax.persistence.OptimisticLockException`will be thrown, and the original transaction will be rolled back. When the application specifies optimistic lock modes, the persistence provider verifies that a particular entity has not changed since it was read from the database even if the entity data was not modified.

Pessimistic locking goes further than optimistic locking. With pessimistic locking, the persistence provider creates a transaction that obtains a long-term lock on the data until the transaction is completed, which prevents other transactions from modifying or deleting the data until the lock has ended. Pessimistic locking is a better strategy than optimistic locking when the underlying data is frequently accessed and modified by many transactions.

Note - Using pessimistic locks on entities that are not subject to frequent modification may result in decreased application performance.

Using Optimistic Locking

The `javax.persistence.Version` annotation is used to mark a persistent field or property as a version attribute of an entity. By adding a version attribute, the entity is enabled for optimistic concurrency control. The version attribute is read and updated by the persistence provider when an entity instance is modified during a transaction. The application may read the version attribute, but **must not** modify the value.

Note - Although some persistence providers may support optimistic locking for entities that do not have version attributes, portable applications should always use entities with version attributes when using optimistic locking. If the application attempts to lock an entity without a version attribute, and the persistence provider doesn't support optimistic locking for non-versioned entities, a`PersistenceException` will be thrown.

The `@Version` annotation has the following requirements:

- Only a single `@Version` attribute may be defined per entity.
- The `@Version` attribute must be in the primary table for an entity mapped to multiple tables.
- The type of the `@Version` attribute must be one of the following: `int`, `Integer`, `long`, `Long`,`short`, `Short`, or `java.sql.Timestamp`.

The following code snippet shows how to define a version attribute in an entity with persistent fields:

```
@Version
protected int version;
```

The following code snippet shows how to define a version attribute in an entity with persistent properties:

```
@Version
protected Short getVersion() { ... }
```