Oracle® Fusion Middleware

Integration Guide for Oracle TopLink with Coherence Gird

11*g* Release 1 (11.1.1)

E16596-01

April 2010

Oracle TopLink 11*g* Release 1 (11.1.1) includes tight integration with Oracle Coherence. This integration, provided through the TopLink Grid feature, marries the standardization and simplicity of application development using the Java Persistence API (JPA) with the scalability and distributed processing power of Oracle Coherence Data Grid.

This document explains how to use:

- TopLink Grid to leverage the Coherence data grid as the primary data store for entities
- Coherence as a distributed shared cache
- Coherence's parallel processing support to perform JPQL queries on cached entities
- EclipseLink JPA's optimized `CacheStore` and `CacheLoader` implementations in traditional Coherence applications

This document includes the following information:

- Understanding the TopLink Grid Integration
- "Traditional Coherence" Configuration
- "JPA on the Grid" Configurations
- Queries
- Labs and Examples
- Documentation Accessibility

For additional information, see the following documents:

- *Oracle Fusion Middleware Developer's Guide for Oracle TopLink*
- *Oracle Coherence Developer's Guide*
- *Oracle Coherence Client Guide*

Understanding the TopLink Grid Integration

TopLink Grid integrates the TopLink JPA implementation (EclipseLink) with Oracle Coherence and provides two development approaches:

- You can use the Coherence API with caches backed by TopLink Grid to access relational data with JPA `CacheLoader` and `CacheStore` implementations. In this "traditional" Coherence approach, TopLink Grid provides `CacheLoader` and `CacheStore` implementations that are optimized for EclipseLink JPA.

ORACLE

Figure 1 "Traditional" Coherence Approach

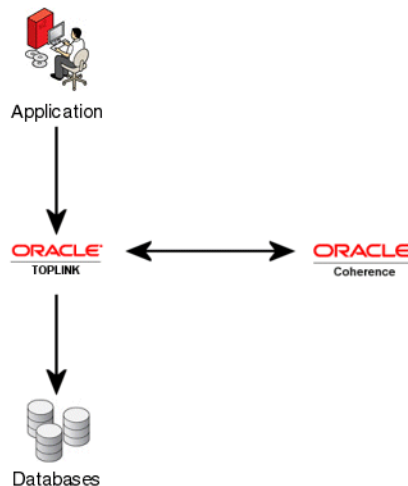
In the Traditional Coherence approach, use the Coherence APIs (with caches backed by TopLink) to access the database.

****

Refer to ""Traditional Coherence" Configuration" on page 4 and the *Coherence Integration Guide* for details on this approach.

- You can also build applications using JPA and transparently leverage the power of the data grid for improved scalability and performance. In this "JPA on the grid" approach, TopLink Grid provides a set of cache and query configuration options that allow you to control how EclipseLink JPA uses Coherence.

Figure 2 "JPA on the Grid" Approach



In the JPA on the Grid approach, TopLink provides options to control how EclipseLink uses Coherence.

****

You can configure Coherence as a distributed shared cache or use Coherence as the primary data store. You can also configure Entities to execute queries in the Coherence data grid instead of the database. This allows clustered application deployments to scale beyond database-bound operations.

Refer to ""JPA on the Grid" Configurations" on page 8 and the *Coherence Integration Guide* for details on this approach.

When integrating JPA applications with the Coherence Data Grid, you should be aware of potential benefits and restrictions. You must understand how the grid works and how it relates to your JPA configurations in order to realize the full potential.

This section includes information on the following configuration options:

- Entity Caching
- Reading and Querying
- Writing

Entity Caching

By default, EclipseLink provides an `EntityManagerFactory` managed shared Entity cache. This shared cache improves performance for multi-threaded and Java EE server hosted applications running in a single JVM.

With TopLink Grid, you can replace the default EclipseLink Shared (L2) cache with Coherence. This provides support for very large shared grid caches that span cluster nodes, and removes the need for additional configuration to ensure individual shared caches are coordinated. By configuring an Entity as Grid cached in Coherence, all `EntityManager.find()` calls for that Entity will result in a `get` on the associated Coherence cache. If Coherence doesn't contain the object the database is queried.

See "Grid Cache" on page 8 for more information.

Reading and Querying

In addition to Grid cache configuration, you can configure TopLink Grid to direct **read** queries to Coherence. By configuring a TopLink JPA `CacheLoader`, even when there is no cache hit, the object can be read from the database and then placed in the cache, thereby making it available for subsequent queries. Coherence's ability to manage very large numbers of objects increases the likelihood of a cache hit as reads in one cluster member become immediately available to others.

While using Coherence to spread an Entity Grid cache across the grid is useful, support for non-primary key queries is especially beneficial. When you configure an Entity as "Grid Read" all **reads** are directed to Coherence. JPQL queries are automatically translated into Coherence filters and objects that match the filter are retrieved from the grid. Coherence executes all filters in parallel on each member of a cluster. This results in significantly faster processing for a query, compared to if all the objects resided in a single member.

Because filters only apply to objects in the Coherence cache, the configuration of a `CacheStore` or `CacheLoader` has no impact on ad-hoc query processing. By default, with this configuration queries are not executed against the database. However, you can override this behavior and with query hints by using the `oracle.eclipselink.coherence.integrated.querying.IgnoreDefaultRedirector` class as shown in the following example:

```
query.setHint(QueryHints.QUERY_REDIRECTOR, new IgnoreDefaultRedirector());
```

This directs the query to the database instead of the Coherence cache.

For complete information on using EclipseLink JPA query hints, refer to the EclipseLink documentation.

Writing

Another key configuration option is specifying how to write Entities to the database. You can configure EclipseLink to:

- Directly write Entities to the database, then put them in Coherence (so that it reflects the database state)

or

- Put entities into Coherence, then have Coherence write to the database using a `CacheStore`.

The `CacheStore` method supports the Coherence write-behind feature to enable asynchronous database writes. By using this applications do not have to wait for the database to return in order to proceed.

However, this configuration contains some restrictions, such as the inability to use JTA integration or other EclipseLink performance features (including batch writing, parameter binding, stored procedures, and statement ordering).

"Traditional Coherence" Configuration

This section includes information on the CacheStore/CacheLoader configuration.

CacheStore/CacheLoader

The TopLink Grid CacheStore/CacheLoader configuration allows you to map classes using JPA, but use the Coherence API to interact with the Coherence cache. This allows Coherence to interact with the database.

You can also use asynchronous writing with the CacheStore/CacheLoader configuration, by using the Coherence "write behind" configuration.

In general:

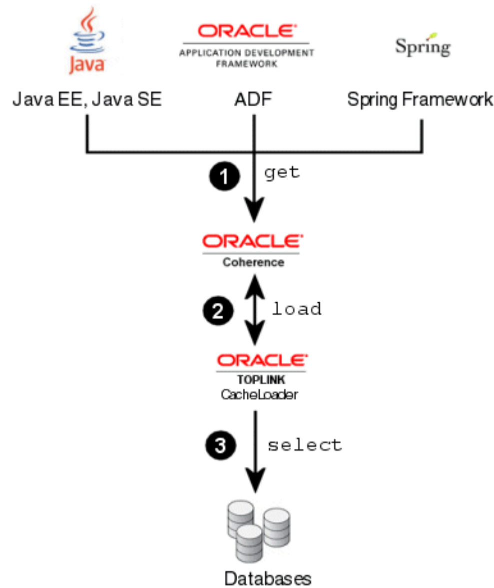
- Through a CacheLoader, all read operations get objects from the database. See "Reading Objects" on page 5.
- Through a CacheLoader, all write operations update the database. See "Writing Objects" on page 5.

See "Examples" on page 6 for detailed examples.

Reading Objects

In the Coherence CacheStore/CacheLoader, all read queries are directed to the database by the TopLink `CacheLoader`.

Figure 3 Reading Objects



This figure illustrates a query in the Grid Read configuration:

1. Application issues a `get` query.
2. By using a CacheLoader, Coherence will load the from TopLink.
3. TopLink will query the database.

****

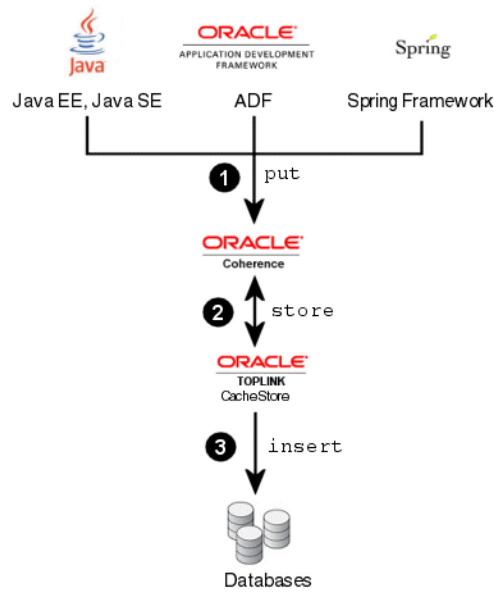
This figure illustrates a query in the Grid Read configuration:

1. Application issues a `get` query.
2. By using a CacheLoader, Coherence will load the from TopLink.
3. TopLink will query the database.

Writing Objects

In the Coherence CacheStore/CacheLoader, TopLink performs all database writes (insert, update, delete) through the `CacheStore`.

Figure 4 Writing Objects



This figure illustrates a query in the Grid Read configuration:

1. Application issues a `put` query.
2. By using a CacheStore, Coherence will store the from TopLink.
3. TopLink will insert the object into the database.

****

This figure illustrates a query in the Grid Read configuration:

1. Application issues a `put` query.
2. By using a CacheStore, Coherence will store the from TopLink.
3. TopLink will insert the object into the database.

Examples

In the cache configuration (`coherence-cache-config.xml`) define the cache, as shown in this example.

Example 1 Configuring the Cache

```
<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>Employee</cache-name>
      <scheme-name>distributed-eclipselink</scheme-name>
    </caching-scheme-mapping>
  </caching-scheme-mapping>
  <caching-schemes>
```

```
    <distributed-scheme>
      <scheme-name>distributed-eclipselink</scheme-name>
      <service-name>EclipseLinkJPA</service-name>
      <backing-map-scheme>
        <read-write-backing-map-scheme>
          <internal-cache-scheme>
            <local-scheme />
          </internal-cache-scheme>
          <!--
            Define the cache scheme
          -->
          <cachestore-scheme>
            <class-scheme>
            <!--
              Since the client code is using Coherence API we need the
"standalone" version of the cache loader
            -->
              <class-name>oracle.eclipselink.coherence.standalone.EclipseLinkJPACa
cheStore</class-name>
              <init-params>
                <init-param>
                  <param-type>java.lang.String</param-type>
                  <param-value>{cache-name}</param-value>
                </init-param>
                <init-param>
                  <param-type>java.lang.String</param-type>
                  <param-value>employee</param-value>
                </init-param>
              </init-params>
            </class-scheme>
          </cachestore-scheme>
        </read-write-backing-map-scheme>
      </backing-map-scheme>
      <autostart>true</autostart>
    </distributed-scheme>
  </caching-schemes>
</cache-config>
```

In Example 2, you set the employee information and add the new object to Coherence. This issues an INSERT to the `CacheStore`.

Example 2 Inserting Objects

```
Employee employee = new Employee();
employee.setId(NEW_EMP_ID);
employee.setFirstName("John");
employee.setLastName("Doe");
// Putting a new object into Coherence  will result in an INSERT in the CacheStore
employeeCache.put(NEW_EMP_ID, employee);
```

Getting an object from the cache produces no SQL statements. Getting an object that is *not* in the cache produces a `SELECT` statement.

Example 3 Reading Objects

```
// Getting an object from cache produces no SQL
```

```
System.out.println("New Employee from cache is: " + employeeCache.get(NEW_EMP_
ID));

// Getting an object not in cache will produce a SELECT in the CacheStore
System.out.println("Non-existant Employee from cache is: " +
employeeCache.get(NON_EXISTANT_EMP_ID));
```

"JPA on the Grid" Configurations

This section includes information on the following configurations:

- Grid Cache
- Grid Read
- Grid Entity

The sample code illustrated in the examples for each configuration can be obtained from:
http://www.oracle.com/technology/products/ias/toplink/doc/11110/grid/guide/TopLinkGrid-Examples.zip.

Grid Cache

The TopLink Grid Coherence Grid Cache configuration uses Coherence as the TopLink shared (L2) cache. This brings the power of the Coherence data grid to JPA applications that rely on database hosted data that cannot be entirely preloaded into a Coherence cache for reasons including: extremely complex queries that exceed the feature set of Coherence Filters, third party database updates that create stale caches, reliance on native SQL queries, stored procedures or triggers, etc.

By using Coherence as its Grid cache, you can scale TopLink up into large clusters while avoiding the need to coordinate local Grid caches. Updates made to entities are available in all Coherence cluster members immediately, upon a transaction commit.

In general:

- Primary key queries attempt to get entities first from Coherence and, if unsuccessful, will query the database, updating Coherence with query results. See "Reading Objects" on page 8.
- Non-primary key queries are executed against the database and the results checked against Coherence to avoid object construction costs for cached entities. Newly queried entities are put into Coherence.
- Write operations update the database and, if successfully committed, updated entities are put into Coherence. See "Writing Objects" on page 10.

See "Examples" on page 10 for detailed examples.

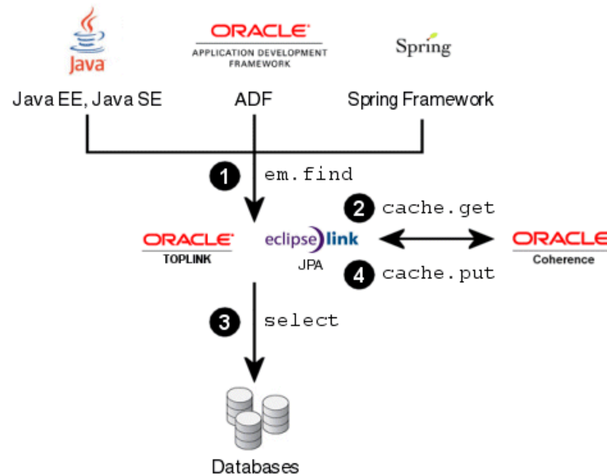
Reading Objects

In the Coherence Grid Cache configuration, all read queries are directed to the database *except* primary key queries, which are performed against the Coherence cache first. Any cache misses will result in a database query.

All entities queried from the database are placed in the Coherence cache. This makes them immediately available to all members of the cluster which is valuable because, by default, TopLink leverages the cache to avoid constructing new entities from database results.

For each row resulting from a query, TopLink uses the primary key of the result row to query the corresponding entity from the cache. If the cache contains the entity then the entity is used and a new entity isn't built. This approach can greatly improve application performance, especially with a warmed cache, as reduces the cost of a query.

Figure 5 Reading Objects



This figure illustrates a query in the Coherence Cache configuration:

1. Application issues a find query.
2. For primary key queries, TopLink queries the Coherence cache first.
3. If the object does not exist in the Coherence cache, TopLink queries the database.

   For all read queries except primary key queries, TopLink queries the database first.
4. Read objects are put into the Coherence cache.

****

This figure illustrates a query in the Coherence Grid Cache configuration:

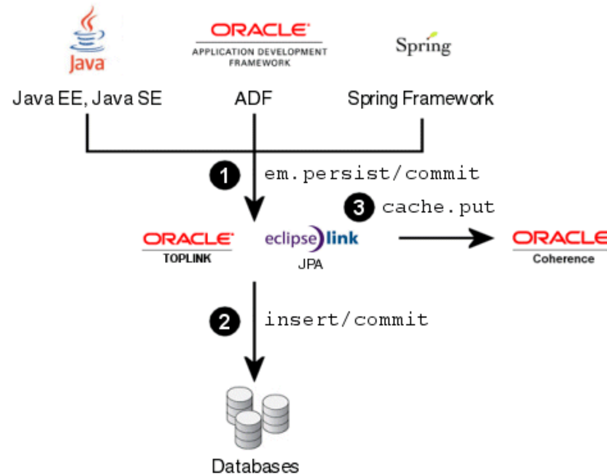
1. Application issues a `find` query.
2. For primary key queries, TopLink queries the Coherence cache first.
3. If the object does not exist in the Coherence cache, TopLink queries the database.

   For all read queries *except primary key queries*, TopLink queries the database first.
4. Read objects are `put` into the Coherence cache.

Writing Objects

In the Coherence Grid Cache configuration, TopLink performs all database writes (insert, update, delete). The Coherence cache is then updated to reflect the changes made to the database. TopLink offers a number of performance features when writing large amounts of data including: batch writing, parameter binding, stored procedure support, and statement ordering to ensure that database constraints are satisfied.

Figure 6 Writing and Persisting Objects



This figure illustrates a query in the Coherence Grid Cache configuration:

1. Application issues a `commit` query.
2. TopLink updates the database.
3. After a successful transaction, TopLink updates the Coherence cache.

****

This figure illustrates a query in the Coherence Grid Cache configuration:

1. Application issues a `commit` query.
2. TopLink updates the database.
3. After a successful transaction, TopLink updates the Coherence cache.

Examples

The sample code illustrated in these examples can be obtained from: `http://www.oracle.com/technology/products/ias/toplink/doc/11110/grid/guide/TopLinkGrid-Examples.zip`.

In the cache configuration (`coherence-cache-config.xml`) define the cache and configure a wrapper serializer in order to support serialization of relationships, as shown in this example:

Example 4   Configuring the Cache

```
<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>*</cache-name>
      <scheme-name>eclipselink-distributed</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>
  <caching-schemes>
    <distributed-scheme>
      <scheme-name>eclipselink-distributed</scheme-name>
      <service-name>EclipseLinkJPA</service-name>
      <!--
        Configure a wrapper serializer to support serialization of relationships.
      -->
      <serializer>
        <class-name>oracle.eclipselink.coherence.integrated.cache.WrapperSerialize
r</class-name>
      </serializer>
      <backing-map-scheme>
      <!--
        Backing map scheme with no eviction policy
      -->
        <local-scheme>
          <scheme-name>unlimited-backing-map</scheme-name>
        </local-scheme>
      </backing-map-scheme>
      </backing-map-scheme>
      <autostart>true</autostart>
    </distributed-scheme>
  </caching-schemes>
</cache-config>
```

To configure an entity to use the Coherence Grid cache, use the `CoherencInterceptor` class as shown in Example 5. This class intercepts all TopLink calls to the internal TopLink Grid cache and redirects them to Coherence.

Example 5   Configuring the Entity

```
import oracle.eclipselink.coherence.integrated.cache.CoherenceInterceptor;
import org.eclipse.persistence.annotations.Customizer;

@Entity
@CacheInterceptor(value = CoherenceInterceptor.class)
public class Employee {
...
```

In Example 6, TopLink performs the insert to create a new employee. Entities are persisted through the `EntityManger` and placed in the database.

Example 6   Inserting Objects

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("employee");

// Create an Employee with an Address and PhoneNumber
```

```
EntityManager em = emf.createEntityManager();
em.getTransaction().begin();
Employee employee = createEmployee();
em.persist(employee);
em.getTransaction().commit();
em.close();
```

After a successful transaction, the Coherence cache is updated.

In Example 7, when TopLink finds an employee, the read query is directed to Coherence cache.

Example 7 Querying Objects

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("employee");

EntityManager em = emf.createEntityManager();
List<Employee> employees =
em.createNamedQuery("Employee.findByLastNameLike").setParameter("lastName",
"Smit%").getResultList();

for (Employee employee : employees) {
   System.err.println(employee);
   for (PhoneNumber phone : employee.getPhoneNumbers()) {
      System.err.println("\t" + phone);
   }
}

emf.close();
```

Grid Read

The TopLink Grid Read configuration should be used for entities that require fast access to large amounts of (fairly stable) data and must write changes synchronously to the database. In these entities, cache warming would typically be used to populate the Coherence cache but individual queries could be directed to the database if necessary.

In general:

- Read operations get objects from the Coherence cache. Configuring a CacheLoader has no impact on JPQL queries. See "Reading Objects" on page 12.
- Write operations update the database and, if successfully committed, updated entities are put into Coherence. See "Writing Objects" on page 14.

See "Examples" on page 15 for detailed examples.

Reading Objects

In the Grid Read configuration, all read queries for an entity are directed to the Coherence cache. To reduce query processing time, TopLink Grid supports parallel processing of queries across the data grid. Coherence contains data in object form, avoiding the cost of database communication and object construction.

With the Grid Read configuration, if Coherence does not contain the Entity requested by `find(...)` then `null` is returned. However, if a `CacheLoader` is configured for

the Entity's cache, Coherence will attempt to load the object from the database. This is only true for primary key queries.

Configuring a `CacheLoader` has no impact on JPQL queries translated to Coherence filters. When searching with a filter, Coherence will operate *only* on the set of Entities in the caches; the database will not be queried. However, it is possible on a query-by-query basis to direct a query to the database instead of to Coherence by using the `oracle.eclipselink.coherence.integrated.querying.IgnoreDefaultRedirector` class, as shown in following example:

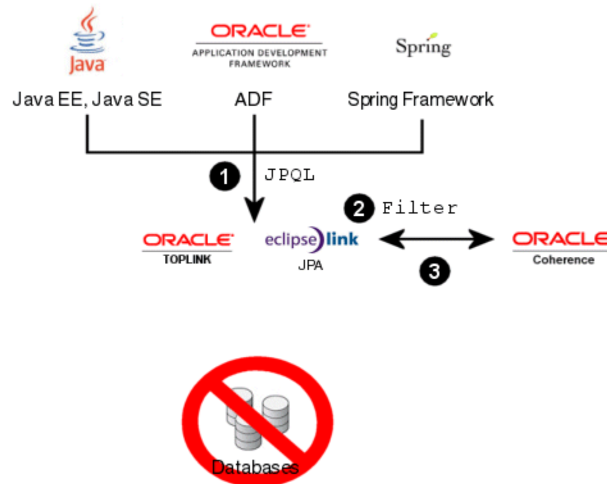
```
query.setHint(QueryHints.QUERY_REDIRECTOR, new IgnoreDefaultRedirector());
```

Any objects retrieved by a database query will be added to the Coherence cache so they are available for subsequent queries. Because, by default, this configuration resolves all queries for an entity through Coherence, the Coherence cache should be warmed with all the data that is to be queried.

A `CacheStore` is not compatible with the Grid Read configuration because EclipseLink will be performing all database updates and then propagating the updated objects into Coherence. If you use a `CacheStore`, Coherence will attempt to write out the just-written objects again.

For complete information on using EclipseLink JPA query hints, refer to the EclipseLink documentation.

Figure 7 Reading Objects with a Query



This figure illustrates a query in the Grid Read configuration:

1. Application issues a JPQL query.
2. TopLink executes a Filter on the Coherence cache.
3. TopLink returns results from the Coherence cache only; the database is not queried.

****

This figure illustrates a query in the Grid Read configuration:

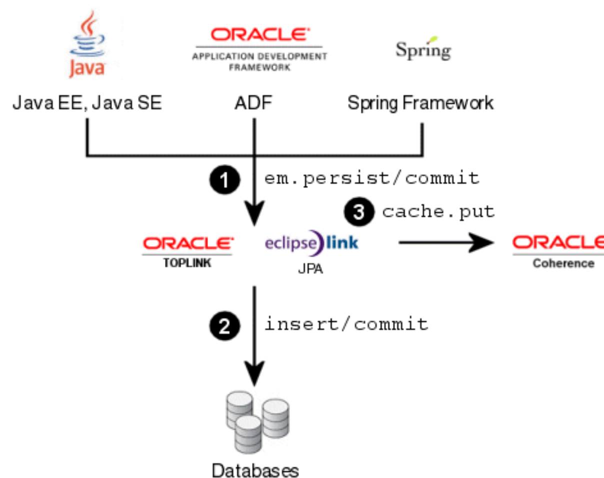
1. Application issues a JPQL query.
2. TopLink executes a Filter on the Coherence cache.
3. TopLink returns results from the Coherence cache only; the database is not queried.

Writing Objects

In the Grid Read configuration, TopLink performs all database writes directly (insert, update, delete). The Coherence caches are then updated to reflect the changes made to the database. TopLink offers a number of performance features when writing large amounts of data including: batch writing, parameter binding, stored procedure support, and statement ordering to ensure that database constraints are satisfied.

This approach offers the best of both worlds: database updates are performed efficiently *and* queries continue to be executed in parallel across the Coherence data grid, with the option of directing individual queries to the database.

Figure 8 Writing and Persisting Objects



This figure illustrates a query in the Grid Read configuration:

1. Application issues a `commit` query.
2. TopLink updates the database.
3. After a successful transaction, TopLink updates the Coherence cache.

****

This figure illustrates a query in the Grid Read configuration:

1. Application issues a `commit` query.
2. TopLink updates the database.
3. After a successful transaction, TopLink updates the Coherence cache.

Examples

The sample code illustrated in these examples can be obtained from: http://www.oracle.com/technology/products/ias/toplink/doc/11110/grid/guide/TopLinkGrid-Examples.zip.

In the cache configuration (`coherence-cache-config.xml`) define the cache and configure a wrapper serializer in order to support serialization of relationships, as shown in this example:

Example 8 Configuring the Cache

```
<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>*</cache-name>
      <scheme-name>eclipselink-distributed-readonly</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>
  <caching-schemes>
    <distributed-scheme>
      <scheme-name>eclipselink-distributed-readonly</scheme-name>
      <service-name>EclipseLinkJPAReadOnly</service-name>
      <!--
        Configure a wrapper serializer to support serialization of relationships.
      -->
      <serializer>
        <class-name>oracle.eclipselink.coherence.integrated.cache.WrapperSerialize
r</class-name>
      </serializer>
      <backing-map-scheme>
        <read-write-backing-map-scheme>
          <internal-cache-scheme>
            <local-scheme />
          </internal-cache-scheme>
          <!--
            Define the cache scheme
          -->
          <cachestore-scheme>
            <class-scheme>
              <class-name>oracle.eclipselink.coherence.integrated.EclipseLinkJPACa
cheLoader</class-name>
              <init-params>
                <param-type>java.lang.String</param-type>
                <param-value>{cache-name}</param-value>
              </init-param>
                <init-param>
                  <param-type>java.lang.String</param-type>
                  <param-value>employee</param-value>
                </init-param>
              </init-params>
            </class-scheme>
          </cachestore-scheme>
          <!--
            The read-only = true required when using a CacheLoader. If omitted,
Coherence  will attempt to call CacheStore methods that are not available on
CacheLoader.
          -->
```

```
        <read-only>true</readonly>
      </read-write-backing-map-scheme>
    </backing-map-scheme>
    <autostart>true</autostart>
  </distributed-scheme>
 </caching-schemes>
</cache-config>
```

To configure an entity to read through Coherence, use the `CoherenceReadCustomizer` as shown in the following example:

Example 9 Configuring the Entity

```
import oracle.eclipselink.coherence.integrated.config.CoherenceReadCustomizer;
import org.eclipse.persistence.annotations.Customizer;

@Entity
@Customizer(CoherenceReadCustomizer.class)
public class Employee {
...
}
```

In Example 10, TopLink performs the insert to create a new employee. If the transaction is successful, the new object is placed into Coherence under it's primary key.

Example 10 Inserting Objects

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("employee");

// Create an Employee with an Address and PhoneNumber
EntityManager em = emf.createEntityManager();
em.getTransaction().begin();
Employee employee = createEmployee();
em.persist(employee);
em.getTransaction().commit();
em.close();

emf.close();
```

When finding an employee, the read query is directed to Coherence cache. The JPQL query is translated to Coherence filters, as shown in the following example.

Example 11 Querying Objects

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("employee");
EntityManager em = emf.createEntityManager();
List<Employee> employees =
em.createNamedQuery("Employee.findByLastNameLike").setParameter("lastName",
"Smit%").getResultList();
  for (Employee employee : employees) {
    System.err.println(employee);
      for (PhoneNumber phone : employee.getPhoneNumbers()) {
        System.err.println("\t" + phone);
      }
```

```
  }
emf.close();
```

To get an object from the Coherence cache with a specific ID (key), use: `em.find(Entity.class, ID)`. You can also configure a Coherence `CacheLoader` to query the database to find the object, if the cache does not contain one with the specified ID.

Grid Entity

The Grid Entity configuration should be used by applications that require fast access to large amounts of (fairly stable) data and that perform relatively few updates. This configuration can be combined with a Coherence `CacheStore` using write-behind to improve application response time by performing database updates asynchronously.

In general:

- Read operations get objects from the Coherence cache. See "Reading Objects" on page 17.
- Write operations put objects into the Coherence cache. If a CacheStore is configured, TopLink also performs write operations on the database. See "Writing Objects" on page 17.

See "Examples" on page 18 for detailed examples.

Reading Objects

In the Grid Entity configuration, reading objects is identical to the Grid Read configuration. See "Reading Objects" on page 12 for more information.

Writing Objects

In the Grid Entity configuration, all objects persisted, updated, or merged through an `EntityManger` will be put in the appropriate Coherence cache. To persist objects in a Coherence cache to the database, an EclipseLink `CacheStore` (`oracle.eclipselink.coherence.integrated.EclipseLinkJPACacheStore`) must be configured for each cache.

You can also configure the CacheStore to use a "write behind" to asynchronously batch-write updated objects. See the *Coherence Developer's Guide* for more information.

Issues to be Aware of When Writing Objects This section includes information on items you should be aware of when writing objects.

When using a `CacheStore`, Coherence assumes that all write operations succeed and will not inform TopLink of a failure. This could result in the Coherence cache differing from the database. You cannot use optimistic locking to protect against data corrupts that may occur if the database is concurrently modified by Coherence and a third-party application.

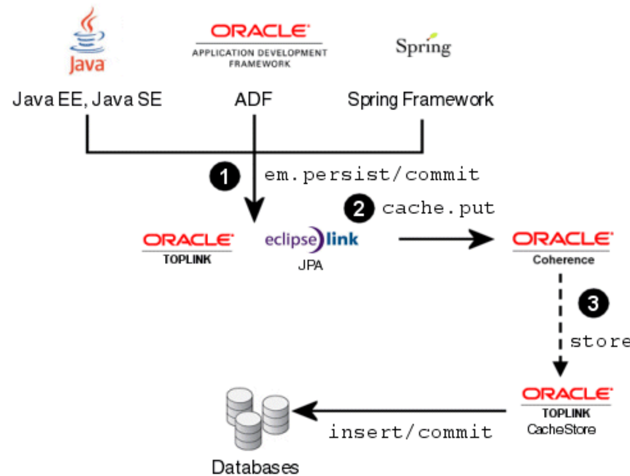
Because each class may be in a separate cache, Coherence may not issue the delete calls to the stores in the required order or with correct timing. As a result, constraint compliance is not guaranteed and write operations could fail with the following error:

```
org.eclipse.persistence.exceptions.DatabaseException
Internal Exception: java.sql.BatchUpdateException: ORA-02292: integrity constraint
```

```
violated - child record found
Error Code: 2292
```

The database foreign key restraints should be reviewed and may need to be removed.

Figure 9 Writing and Persisting Objects



This figure illustrates a query in the Grid Entity configuration:

1. Application issues a `commit`.
2. TopLink directs all queries to update the Coherence cache.
3. By configuring a Coherence CacheStore (optional), TopLink will also update the database.

****

This figure illustrates a query in the Grid Entity configuration:

1. Application issues a `commit`.
2. TopLink directs all queries to update the Coherence cache.
3. By configuring a Coherence CacheStore (optional), TopLink will also update the database.

Examples

In the cache configuration (`coherence-cache-config.xml`) configure a wrapper serializer in order to support serialization of relationships, as shown in this example:

The sample code illustrated in these examples can be obtained from: `http://www.oracle.com/technology/products/ias/toplink/doc/11110/grid/guide/TopLinkGrid-Examples.zip`.

Example 12 Configuring the Cache

```
<cache-config>
```

```
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>*</cache-name>
      <scheme-name>eclipselink-distributed-readwrite</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>
  <caching-schemes>
    <distributed-scheme>
      <scheme-name>eclipselink-distributed-readwrite</scheme-name>
      <service-name>EclipseLinkJPAReadWrite</service-name>
      <!--
        Configure a wrapper serializer to support serialization of relationships.
      -->
      <serializer>
        <class-name>oracle.eclipselink.coherence.integrated.cache.WrapperSerialize
r</class-name>
      </serializer>
      <backing-map-scheme>
       <read-write-backing-map-scheme>
        <internal-cache-scheme>
          <local-scheme />
          </internal-cache-scheme>
           <!--
             Define the cache scheme
           -->
            <cachestore-scheme>
              <class-scheme>
                <class-name>oracle.eclipselink.coherence.integrated.EclipseLinkJPA
CacheStore</class-name>
                <init-params>
                  <init-param>
                    <param-type>java.lang.String</param-type>
                    <param-value>{cache-name}</param-value>
                  </init-param>
                  <init-param>
                    <param-type>java.lang.String</param-type>
                    <param-value>employee</param-value>
                  </init-param>
                </init-params>
              </class-scheme>
            </cachestore-scheme>
        </read-write-backing-map-scheme>
      </backing-map-scheme>
      <autostart>true</autostart>
    </distributed-scheme>
  </caching-schemes>
</cache-config>
```

To configure an entity to read through Coherence, use the `CoherenceReadWriteCustomizer` as shown in the following example:

Example 13 Configuring the Entity

```
import
oracle.eclipselink.coherence.integrated.config.CoherenceReadWriteCustomizer;
import org.eclipse.persistence.annotations.Customizer;

@Entity
```

```
@Customizer(CoherenceReadWriteCustomizer.class)
public class Employee {
...
}
```

In Example 14, TopLink performs the insert to create a new employee. Entities are persisted through the `EntityManger` and placed in the appropriate Coherence cache.

Example 14 Persisting Objects

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("employee");

// Create an Employee with an Address and PhoneNumber
EntityManager em = emf.createEntityManager();
em.getTransaction().begin();
Employee employee = createEmployee();
em.persist(employee);
em.getTransaction().commit();
em.close();
```

When finding an employee, the read query is directed to Coherence cache, as shown in the following example.

Example 15 Querying Objects

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("employee");

EntityManager em = emf.createEntityManager();
List<Employee> employees =
em.createNamedQuery("Employee.findByLastNameLike").setParameter("lastName",
"Smit%").getResultList();

for (Employee employee : employees) {
   System.err.println(employee);
   for (PhoneNumber phone : employee.getPhoneNumbers()) {
      System.err.println("\t" + phone);
   }
}

emf.close();
```

To get an object from the Coherence cache with a specific ID (key), use: `em.find(Entity.class, ID)`. You can also configure a Coherence `CacheLoader` to query the database to find the object, if the cache does not contain one with the specified ID.

Relationships

This section includes information on the following:

- Wrapping and Unwrapping

Wrapping and Unwrapping

When storing Entities with relationships in the Coherence cache, TopLink Grid generates a wrapper class that maintains the relationship information. In this way, when the object is read from the Coherence cache (eager or lazy), the relationship can be resolved.

If you read Entities directly from the Coherence cache using the Coherence API, the wrappers are not automatically removed. You must configure automatic unwrapping on a `get` in your code or by setting the property on the serializer, as shown in Example 16.   You can also set the system property as `eclipselink.coherence.not-eclipselink` to automatically unwrap an entity.

When configured properly, the **read** will return the wrapped Entity.

Example 16   Unwrapping an Entity

```
WrapperSerializer wrapperSerializer =
(WrapperSerializer)myCache.getCacheService().getSerializer();
wrapperSerializer.setNotEclipseLink(true); // to let the Serializer know it needs
to unwrap when clients get() from the cache
```

Queries

This section includes information on the following:

- Querying Objects by ID
- Querying Objects with Criteria
- Limitations and Restrictions

Querying Objects by ID

To get an entity from the Coherence cache with a specific ID (key), use: `em.find(Entity.class, ID)`.

For example, `em.find(Employee.class, 8)` will get the entity with key **8**, from the Coherence **Employee** cache.

If the entity is not found in the Coherence cache, TopLink executes a `SELECT` statement against the database. If a result is found, then the entity is constructed and placed into Coherence.

Querying Objects with Criteria

To get an entity that matches a specific selection criteria, use: `em.createQuery("...")`. For example, `em.createQuery("select e from Employee e where e.name='John'")` will execute a SELECT statement against the database to find employees with the **name** of **John**.

The query's specific behavior will depend on you Coherence cache configuration.

- Grid Cache – The query will always check the database.

- Grid Read and Grid Entity – The query will check the Coherence cache. If Coherence does not contain the Entity then the database is queried.

  You can use query hints to direct the query to the database instead of the Coherence cache.

- CacheStore/CacheLoader – Non-primary key queries will check the database. Primary key queries are performed against the Coherence cache first.

Limitations and Restrictions

You should be aware of the following limitations when querying Coherence:

- JPQL Bulk Updates and Deletes – This release of TopLink Grid does not provide support for JPQL bulk updates and deletes.
- Joins – Because the Coherence Filter framework is limited to a single cache, JPQL **join** queries cannot be translated to Filters – all **join** queries will execute on the database.

  Coherence will continue to be used to avoid object constructions costs for the query results.

- Projection queries – This release of TopLink Grid does not provide support for projection queries (reports).

Labs and Examples

This lab introduces how to use Oracle TopLink Grid feature to grid enable a Java Persistence API (JPA) application with Oracle Coherence. TopLink Grid and Coherence can be combined using two distinct application architectures:

- a "traditional" Coherence application architecture with TopLink providing database access, or
- with Coherence backing TopLink in a "JPA on the Grid" architecture.

This lab will focuses on using TopLink Grid to build JPA on the Grid applications that provide a way to scale JPA applications through the use of Coherence.

This document includes the following sections:

- Configuring the Lab Environment
- Running the Labs
- Summary

Configuring the Lab Environment

This lab requires the following (minimum requirements):

- Java 1.5 JDK/JRE

  `http://www.java.com/download`

- Coherence 3.5

  `http://www.oracle.com/technology/software/products/ias/htdocs/coherence.html`

- TopLink 11*g* Release 1 (11.1.1), which includes EclipseLink

  http://www.oracle.com/technology/software/products/ias/htdocs/1111topsoft.html

- Oracle Enterprise Pack for Eclipse (OEPE) 11*g* Release 1 (11.1.1)

  http://www.oracle.com/technology/software/products/oepe/oepe_11gR1.html

- Oracle XE Database

  http://www.oracle.com/technology/products/database/xe/index.html

Setup

Before running the labs, you must install the necessary software and configure your environment.

You can download the lab samples from:

http://www.oracle.com/technology/products/ias/toplink/doc/11110/grid/labs/toplinkgridlab.zip

Software Installation Use this procedure to install the necessary software for the labs.

1. If not already installed on your machine, download and install a Java 1.5 or higher JDK or JRE.
2. Download and unzip the lab `.ZIP` file into a folder, referred to as **LAB_ROOT**. After unzipped unzipping the file, the folder will contain several sub-folders, including **toplink** and **coherence**.
3. Download Oracle TopLink 11*g* Release 1 (11.1.1) and unzip into **LAB_ROOT/toplink**.
4. Download Coherence 3.5 and unzip into **LAB_ROOT**. It will extract into the **LAB_ROOT/coherence** folder.

> **Note:** Confirm that the folder structure is correct; that you have **LAB_ROOT/coherence** and *not* **LAB_ROOT/coherence/coherence**.

5. Download OEPE 11*g* Release 1 (11.1.1), based on Eclipse Galileo (3.5), and unzip into any folder, referred to as **OEPE install**.
6. Download and install Oracle XE. Follow the installation instructions included with the download. You can install XE in any folder, referred to as **ORACLE_XE_ROOT**.
7. Create or enable the Oracle database user **scott** with password **tiger**. Grant the **connect** and **resource** permissions to the user **scott**.

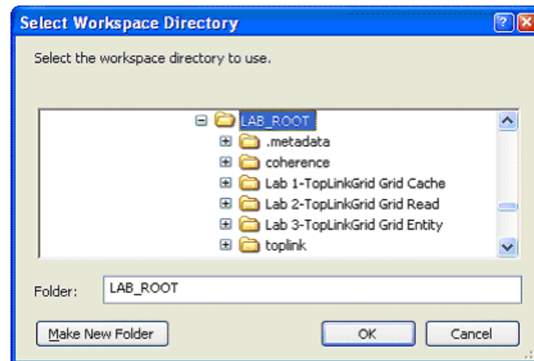
   The labs are configured for username **scott** with password **tiger**. To use a different user, you must change the database login values used in the labs.

Environment Configuration Use this procedure to configure your lab environment.

1. Open OEPE by running `eclipse.exe` from the **OEPE install** directory.

You'll be prompted for a workspace. Select the **LAB_ROOT** folder, which contains your Eclipse workspace.

Figure 10 Select Workspace Directory dialog

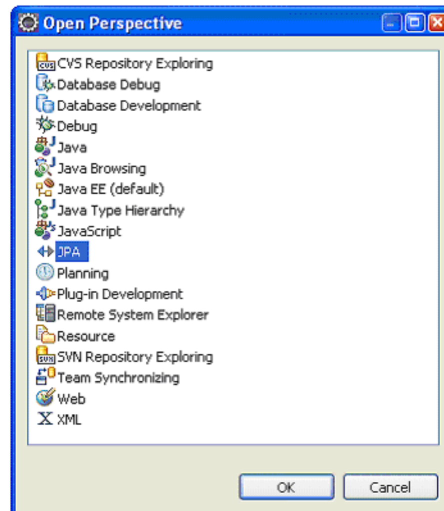


2. If you've unzipped TopLink and Coherence into the correct folders, the workspace will open with no errors in the **Problems** view.

   If you do have errors, select **Project > Clean** from the menu and select **Clean all projects** to recompile and revalidate the labs. If errors remain, confirm that you have unzipped both TopLink and Coherence into the correct folders.

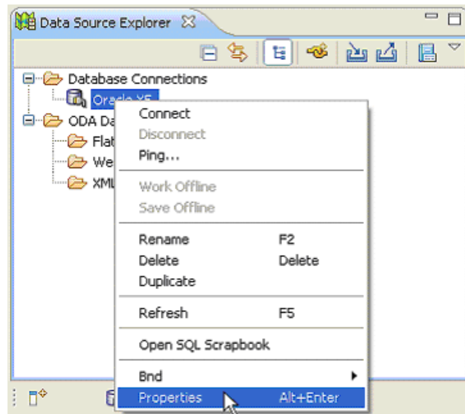
3. From the menu, select **Window > Perspectives** to open the JPA perspective.

Figure 11 Open Perspective dialog



4. In the JPA Perspective use the Data Source Explorer view to change the Oracle XE connection, if needed. By default, the Oracle XE connection uses **scott**/**tiger** @ localhost.

Figure 12 Data Source Explorer



To use a different schema, right-click the Oracle XE connection in the Data Source Explorer and select **Properties**, as shown in Figure 12. Select the **Driver Properties** section and edit the values to reflect your connection as shown in Figure 13.

Note: If you change the defaults, you must also edit the connection values in the `persistence.xml` configuration files used in the labs.

Figure 13 Driver Properties

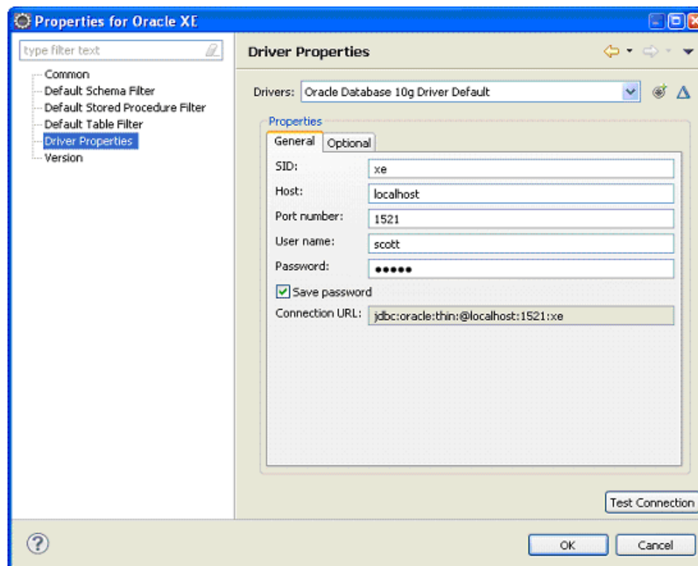
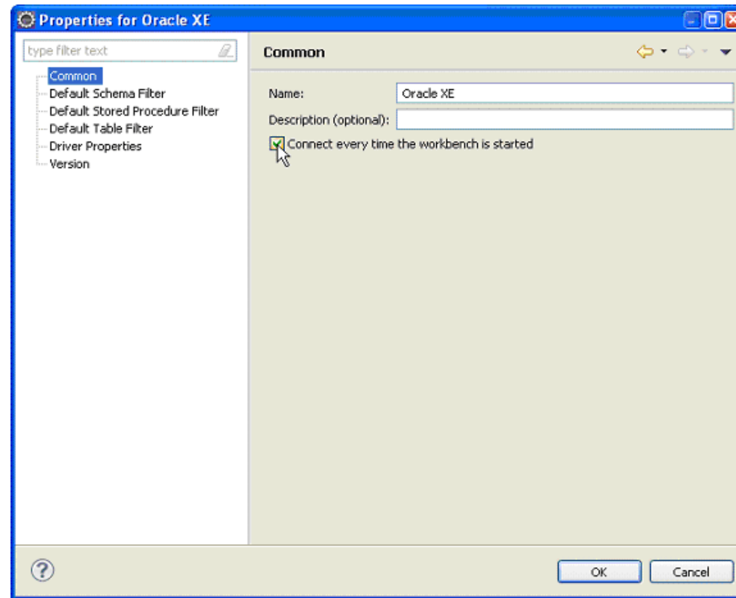


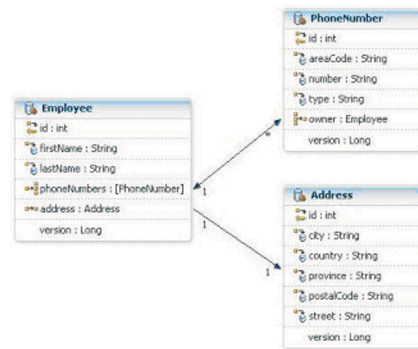
Figure 14 Common connection properties



Running the Labs

Each of the three labs works with the same simple domain model shown in Figure 15. It consists of an **Employee** entity with an **Address** and a collection of **PhoneNumbers**.

Figure 15 Lab Domain Model



The domain model for the labs consists of an Employee entity that has a 1:1 relationship with an Address and a 1:M collection of PhoneNumbers.

****

The entities are all pre-mapped, allowing you to focus on the TopLink Grid configuration – not JPA mapping.

This document contains the following Labs:

- Lab 1: Grid Cache Configuration
- Lab 2: "Grid Read" Configuration
- Lab 3: "Grid Entity" Configuration

Lab 1: Grid Cache Configuration

The most basic configuration is using Coherence with EclipseLink as a shared (L2) cache. EclipseLink has a local, built-in shared-object cache that allows concurrent and successive transactions to benefit from reads and updates performed by other transactions. When entities are updated, after a successful database commit, the updates are then applied to the shared cache, allowing transactions can see the changes.

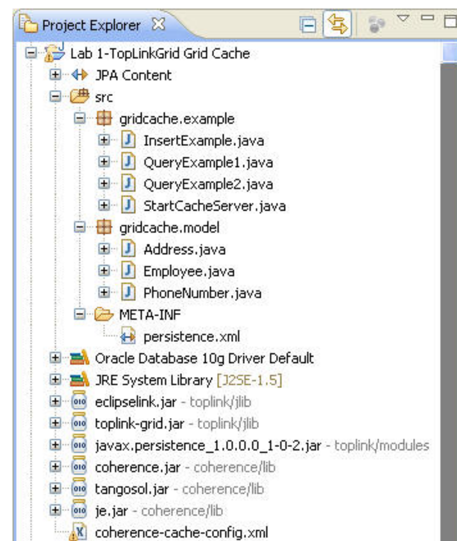
Replacing the local, built-in shared-object cache with Coherence maintains the same cache semantics. Applications do not need to be changed – they are unaware that the cache implementation has become a distributed cache rather than a local one.

Configuring Coherence as a shared cache involves:

- Configuring entities to use Coherence
- Defining a Coherence cache configuration for those entities

Lab 1 (see Figure 16) is provided as a ready-to-run JPA project in Eclipse.

Figure 16 Lab 1 Project



Getting Started Open the **Employee**, **Address**, and **PhoneNumber** entities and examine how they are mapped.

Use the following procedure once, before configuring Coherence as the shared cache, to confirm your environment is correct.

1. If your database connection differs from the defaults, you must update the following EclipseLink JDBC properties in the `META-INF/persistence.xml` file:
   - **eclipselink.jdbc.driver**
   - **eclipselink.jdbc.url**
   - **eclipselink.jdbc.user**
   - **eclipselink.jdbc.password**
2. From the Project Explorer view, right-click the **Lab 1-TopLink Grid Cache** project and select **JPA > Generate Tables from Entities…** from the menu to drop and create the database tables used in the lab. You can safely ignore any errors due to

failures to **DROP** non-existent tables. Generating tables will log you into the database.

Note: f you encounter a **Schema "null" cannot be resolved for table "<TABLE_NAME>"** error, you've hit a known Eclipse bug. To resolve the issue, right-click the project and choose select **Validate** from menu. Eclipse will clear the errors.

---

3. Run **gridcache.example.InsertExample** to populate the database with an Employee, Address, and a PhoneNumber. The `createEmployee()` method, shown here, defines the entities:

```
public static Employee createEmployee() {
  Employee employee = new Employee();
  employee.setFirstName("Bob");
  employee.setLastName("Smith");

  Address address = new Address();
  address.setCity("Toronto");
  address.setPostalCode("L5J2B5");
  address.setProvince("ON");
  address.setStreet("1450 Acme Cr., Suite 4");
  address.setCountry("Canada");
  employee.setAddress(address);

  employee.addPhoneNumber("Work", "613", "5558812");

  return employee;
  }
```

The console output displays the following:

```
[EL Fine]: Connection(876215)--UPDATE SEQUENCE SET SEQ_COUNT = SEQ_COUNT + ?
WHERE SEQ_NAME = ?
[EL Fine]: Connection(876215)--
     bind => [50, SEQ_GEN]
[EL Fine]: Connection(876215)--SELECT SEQ_COUNT FROM SEQUENCE WHERE SEQ_NAME =
?
    bind => [SEQ_GEN]
[EL Fine]: Connection(876215)--INSERT INTO GRIDCACHE_ADDRESS (ID, POSTALCODE,
STREET, PROVINCE, VERSION, COUNTRY, CITY) VALUES (?, ?, ?, ?, ?, ?, ?)
[EL Fine]: Connection(876215)--
     bind => [2, L5J2B5, 1450 Acme Cr., Suite 4, ON, 1, Canada, Toronto]
[EL Fine]: Connection(876215)--INSERT INTO GRIDCACHE_EMPLOYEE (ID, LASTNAME,
FIRSTNAME, VERSION, ADDRESS_ID) VALUES (?, ?, ?, ?, ?)
[EL Fine]: Connection(876215)--
     bind => [1, Smith, Bob, 1, 2]
[EL Fine]: Connection(876215)--INSERT INTO GRIDCACHE_PHONE (ID, AREACODE, NUM,
TYPE, VERSION, OWNER_ID) VALUES (?, ?, ?, ?, ?, ?)
[EL Fine]: Connection(876215)--
     bind => [3, 613, 5558812, Work, 1, 1]
[EL Config]: Connection(14361585)--disconnect
```

EclipseLink updates an ID, generates the SEQUENCE table, and inserting the ID into three tables. All tables in this lab are prefixed with **GRIDCACHE_**.

4. Run **gridcache.example.QueryExample1**. This queries all Employees with JPQL and a single Employee with an `EntityManager.find` call, then prints the results.

```
...
List<Employee> employees = em.createQuery("select e from Employee
e").getResultList();
...
  Employee employee = em.find(Employee.class, employeeId);
...
```

The console output displays the following:

```
------------------JPQL Query
[EL Fine]: Connection(876215)--SELECT ID, LASTNAME, FIRSTNAME, VERSION,
ADDRESS_ID FROM GRIDCACHE_EMPLOYEE
l2.model.Employee@e33e18(1: Smith, Bob)
[EL Fine]: Connection(876215)--SELECT ID, POSTALCODE, STREET, PROVINCE,
VERSION, COUNTRY, CITY FROM GRIDCACHE_ADDRESS WHERE (ID = ?)
     bind => [2]
     City: Toronto
[EL Fine]: Connection(876215)--SELECT ID, AREACODE, NUM, TYPE, VERSION, OWNER_
ID FROM GRIDCACHE_PHONE WHERE (OWNER_ID = ?)
     bind => [1]
     l2.model.PhoneNumber@21d23b(3: Work: 613-5558812)
------------------em.find Query
     l2.model.Employee@e33e18(1: Smith, Bob)
     City: Toronto
     l2.model.PhoneNumber@21d23b(3: Work: 613-5558812)
```

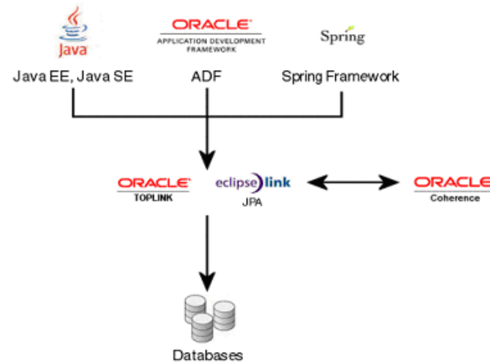
EclipseLink issues three SELECT statements:

- To read all the Employees
- To read the one Employee's Address
- To read its PhoneNumbers

The `EntityManager.find()` does not result in a SQL statement because the Employee is found in the local cache without needing a database query.

Configuring Coherence Grid Caching To instruct EclipseLink to cache in Coherence you need to configure a **CoherenceInterceptor** as the cache interceptor for an Entity. A Coherence interceptor will reroute all cache `get` and `put` operations to Coherence instead of the built-in EclipseLink shared cache. This configuration, shown in Figure 17, is referred to as "cache aside."

Figure 17 Coherence as Shared (L2) Cache



In the Coherence as Shared (L2) Cache approach, a Coherence interceptor will reroute all cache get and put operations to Coherence instead of the built-in EclipseLink shared cahce.

****

Add the `@CacheInterceptor` annotation to all lab entities with the `CoherenceInterceptor.class` as its sole parameter, as shown in this example:

```
import oracle.eclipselink.coherence.integrated.cache.CoherenceInterceptor;
import org.eclipse.persistence.annotations.CacheInterceptor;

@CacheInterceptor(CoherenceInterceptor.class)
public class Employee implements Serializable {
...
```

Coherence Cache Configuration The labs include a `coherence-cache-config.xml` file. In the lab configuration file, all entities are configured identically. Although there is no need to edit the file, you should note and review the following elements:

```
<caching-scheme-mapping>
  <cache-mapping>
    <cache-name>*</cache-name>
    <scheme-name>eclipselink-distributed</scheme-name>
  </cache-mapping>
</caching-scheme-mapping>
```

The asterisk, *, in the **cache-name** element will match any entity name. By default, TopLink Grid uses the entity name as the name of the associated cache. You can override this behavior by using the EclipseLink `@Property` annotation on an entity, as shown in this example:

```
@Property(name="coherence.cache.name", value="Employee")
@CacheInterceptor(CoherenceInterceptor.class)
public class Employee implements Serializable {
...
```

In these labs, the cache names are default to the entity name.

In the Grid Cache lab, all entities use a cache scheme named **eclipselink-distributed** which is a distributed (partitioned) cache. The distributed backing map scheme

eclipselink-distributed has a serializer configured to support serialization of entity relationships into Coherence, as shown in this example:

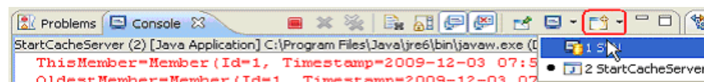
```
<caching-schemes>
  <distributed-scheme>
    <scheme-name>eclipselink-distributed</scheme-name>
    <service-name>EclipseLinkJPA</service-name>
  <!--
   Configure a wrapper serializer to support serialization of
   relationships.
  -->
  <serializer>
    <class-name>
      oracle.eclipselink.coherence.integrated.cache.WrapperSerializer
    </class-name>
  </serializer>
...
```

Running Lab 1 with Coherence Now that the entities are configured to be cached in Coherence and a `coherence-cache-config.xml` file is defined, you can re-run the lab with Coherence.

1. Reset the database by right-clicking the **Lab 1-TopLinkGrid Grid Cache** project in the Project Explorer view and selecting **JPA > Generate Tables from Entities**… from the pop-up menu to drop and create the lab tables.
2. Run **gridcache.example.StartCacheServer** to start the Coherence cache server. After being configured to use Coherence, of the examples require a cache server to be running.

> **Tip:** When running multiple programs in Eclipse, use the drop-down list to switch between each program's console, as shown in Figure 18. When reviewing the console output, be aware of which console you are viewing: the cache server console or the example console.

Figure 18 Eclipse Console



3. Run **gridcache.example.InsertExample** to create example entities in the database and populate the Coherence cache. In addition to Coherence boot messages and INSERT statements in the console you will also see messages showing entities being put into Coherence under their primary key, as shown in this example:

```
[EL Fine]: Coherence(PhoneNumber)::Put: 3 value: l2.model.PhoneNumber@59cbda(3:
Work: 613-5558812)
[EL Fine]: Coherence(Employee)::Put: 1 value: l2.model.Employee@11c55bb(1:
Smith, Bob)
[EL Fine]: Coherence(Address)::Put: 2 value: l2.model.Address@135133(Toronto)
```

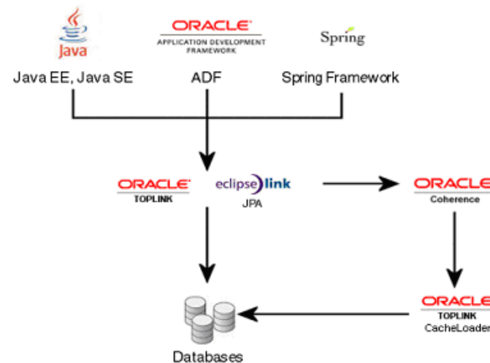
4. Run **gridcache.example.QueryExample1**. In the console you will see a single **SELECT** statement for all Employees resulting from the JQPL query `select e from Employee e` -- that's all. You will also see the messages logging Coherence interaction. Compare this output with the output generated when you ran **QueryExample1** before *enabling* the Coherence cache.

5. Run **gridcache.example.QueryExample2** to illustrate which database queries are produced when you have a series of EntityManagers, such as for an application deployed to an application server. This example is useful when compared to the output generated in subsequent labs.

Lab 2: "Grid Read" Configuration

In "Lab 1: Grid Cache Configuration" you learned how to configure Coherence as a shared entity cache. In this lab you will increase the use of Coherence by directing all read queries (i.e., `select` and `find`) to Coherence. In this "Grid Read" configuration, all read operations are directed to Coherence but write operations are handled by EclipseLink and directed to the database. This configuration is described as "read through," as shown in Figure 19. Use this configuration when you need highly available data and a database that is always up-to-date. This configuration supports JTA so you can participate in a distributed transaction, Coherence is only updated once the entire JTA transaction commits.

Figure 19 Grid Read Configuration



In the "Grid Read" configuration, all read operations are directed to Coherence but write operations are handled by EclipseLink and directed to the database

****

Configuring Grid Read The Grid Read configuration extends the Grid Cache configuration. In addition to a cache interceptor, various read query types are also directed to Coherence. To simplify configuration, TopLink Grid provides an EclipseLink customizer class that performs the necessary configuration changes, including setting the cache interceptor you configured in Lab 1.

The **CoherenceReadCustomizer** is configured using the EclipseLink `@Customizer` annotation, as shown here:

```
import oracle.eclipselink.coherence.integrated.config.CoherenceReadCustomizer;
import org.eclipse.persistence.annotations.Customizer;

@Customizer(CoherenceReadCustomizer.class)
public class Employee implements Serializable {
```

Alternatively, you can also a Customizer in the `persistence.xml` and `eclipselink-orm.xml`.

From the Project Explorer view, right click the **Lab 2-TopLinkGrid Grid Read project** and select **Open Project** from the popup menu. Eclipse may report errors due to missing tables that correspond to the JPA entities in the project-we'll resolve the errors by generating the tables.

Add `@Customizer(CoherenceReadCustomizer.class)`, as shown in the previous code example, to all the lab entities.

Coherence Cache Configuration With all read queries directed to Coherence, the `coherence-cache-config.xml` is different than in the Grid Cache configuration. In particular, it makes sense to configure a **CacheLoader** so that Coherence can query the database for an individual object if it doesn't contain it.

This example shows an excerpt of the `coherence-cache-config.xml` from Lab 2:

```
<distributed-scheme>
  <scheme-name>eclipselink-distributed-readonly</scheme-name>
  <service-name>EclipseLinkJPAReadOnly</service-name>
  ...
  <backing-map-scheme>
    <read-write-backing-map-scheme>
      ...
      <!-- Define the cache scheme -->
      <cachestore-scheme>
        <class-scheme>
          <class-name>
            oracle.eclipselink.coherence.integrated.EclipseLinkJPACacheLoader
          </class-name>
          <init-params>
            <init-param>
              <param-type>java.lang.String</param-type>
              <param-value>{cache-name}</param-value>
            </init-param>
            <init-param>
              <param-type>java.lang.String</param-type>
              <param-value>employee</param-value>
            </init-param>
          </init-params>
        </class-scheme>
      </cachestore-scheme>
      <read-only>true</read-only>
    </read-write-backing-map-scheme>
  </backing-map-scheme>
  <autostart>true</autostart>
</distributed-scheme>
```

We configure an **EclipseLinkJPACacheLoader** on the backing map from the `oracle.eclipselink.coherence.integrated` package. Note the *integrated* package name -- use this CacheLoader when using TopLink Grid with a JPA front end, as in this lab.

When using a TopLink Grid CacheLoader in a *traditional* Coherence API based application, use the EclipseLinkJPACacheLoader in the `oracle.eclipselink.coherence.standalone package`.

Running Lab 2 with Coherence After configuring the entities with the **CoherenceReadCustomizer** and configuring the caches in the `coherence-cache-config.xml` with a **CacheLoader** you can run the lab.

1. Stop any CacheServers that may still be running from the previous Lab.
2. If your database connection differs from the defaults, you must update the following EclipseLink JDBC properties in the `META-INF/persistence.xml` file:
   - **eclipselink.jdbc.driver**
   - **eclipselink.jdbc.url**
   - **eclipselink.jdbc.user**
   - **eclipselink.jdbc.password**
3. From the Project Explorer view, right-click the Lab 2 project and select **JPA > Generate Tables from Entities...** from the pop-up menu to reset the database
4. Run **gridread.example.StartCacheServer** to start the a Coherence cache server.
5. Run **gridread.example.InsertExample** to create example entities in the database and populate the Coherence cache. In addition to Coherence boot messages and **INSERT** statements in the console you will also see messages showing entities being put into Coherence under their primary key, as shown in this example.

```
[EL Fine]: Connection(4889213)--INSERT INTO GRIDREAD_ADDRESS (ID, POSTALCODE,
STREET, PROVINCE, VERSION, COUNTRY, CITY) VALUES (?, ?, ?, ?, ?, ?, ?)
[EL Fine]: Connection(4889213)--
     bind => [2, L5J2B5, 1450 Acme Cr., Suite 4, ONT, 1, Canada, Toronto]
[EL Fine]: Connection(4889213)--INSERT INTO GRIDREAD_EMPLOYEE (ID, LASTNAME,
FIRSTNAME, VERSION, ADDRESS_ID) VALUES (?, ?, ?, ?, ?)
[EL Fine]: Connection(4889213)--
     bind => [1, Smith, Bob, 1, 2]
[EL Fine]: Connection(4889213)--INSERT INTO GRIDREAD_PHONE (ID, AREACODE, NUM,
TYPE, VERSION, OWNER_ID) VALUES (?, ?, ?, ?, ?, ?)
[EL Fine]: Connection(4889213)--
     bind => [3, 613, 5558812, Work, 1, 1]
[EL Fine]: Coherence(Employee)::Put: 1 value: read.model.Employee@948069(1:
Smith, Bob)
[EL Fine]: Coherence(Address)::Put: 2 value:
read.model.Address@1b59919(Toronto)
[EL Fine]: Coherence(PhoneNumber)::Put: 3 value:
read.model.PhoneNumber@1588325(3: Work: 613-5558812)
```

   The console is similar to the console output from Lab 1 with the Grid Cache configuration
6. Run **gridread.example.QueryExample1**. In the **QueryExample1** console EclipseLink *does not* issue a **SELECT** statement from the JQPL query `select e from Employee e` because the query was translated to a Coherence Filter and passed to Coherence for evaluation.

Compare this output with the output from **gridcache.example.QueryExample1** in "Lab 3: "Grid Entity" Configuration":

- In Lab 1, JPQL queries were always translated to SQL and executed on the database.
- In Lab 2, the `EntityManager.find()`, is evaluated against Coherence and produces no SQL.

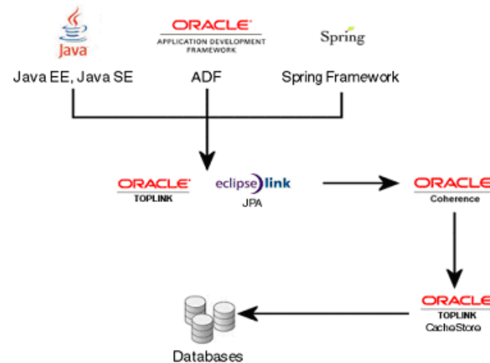
Lab 3: "Grid Entity" Configuration

The Grid Entity configuration extends the Grid Read configuration, redirecting all write queries (`INSERT`, `UPDATE`, and `DELETE`) to Coherence instead of the database.

This configuration is useful when you need quick response time and can use a write-behind strategy to periodically flush updates to the database. This is achieved through configuration of a CacheStore on the Coherence cache. This configuration may be described to as both "read through" and "write through" (Figure 20).

Potential downsides to this configuration are the loss of JTA transaction participation, EclipseLink write optimizations like batch writing, and the need to relax referential integrity rules due to the unpredictable order in which Coherence caches may write updates to the database.

Figure 20 Grid Entity Configuration



The Grid Entity configuration extends the Grid Read configuration, redirecting all write queries (INSERT, UPDATE, and DELETE) to Coherence instead of the database. This configuration is useful when you need quick response time and can use a write-behind strategy to periodically flush updates to the database. This is achieved through configuration of a CacheStore on the Coherence cache.

****

Configuring Grid Entity Like the "Lab 2: "Grid Read" Configuration", Grid Entity is enabled by an EclipseLink Customizer, CoherenceReadWriteCustomizer, set on an individual entity as shown in the following example:

```
import
oracle.eclipselink.coherence.integrated.config.CoherenceReadWriteCustomizer;
import org.eclipse.persistence.annotations.Customizer;

@Customizer(CoherenceReadWriteCustomizer.class)
public class Employee implements Serializable {
```

From the Project Explorer view, right-click the **Lab 3-TopLinkGrid Grid Entity** project and select **Open Project** from the popup menu. As in the other labs, you may see errors due to missing tables that correspond to the JPA entities in the project. These will be resolved after generating the tables.

Add `@Customizer(CoherenceReadWriteCustomizer.class)`, as shown in the previous code example, to all the lab entities.

Coherence Cache Configuration To propagate updates to the database, a Coherence entity cache needs to be configured with a **CacheStore**. A CacheStore is configured like a

CacheLoader. The difference is in the implementation which supports write operations in addition to read operations.

```
<distributed-scheme>
  <scheme-name>eclipselink-distributed-readwrite</scheme-name>
    ...
    <cachestore-scheme>
      <class-scheme>
        <class-name>
          oracle.eclipselink.coherence.integrated.EclipseLinkJPACacheStore
        </class-name>
...
```

As in "Lab 2: "Grid Read" Configuration", the **EclipseLinkJPACacheStore** is in the *integrated* package because we are using JPA as our programming API.

Running Lab 3 with Coherence After configuring the your entities with the **CoherenceReadWriteCustomizer** and configuring the caches in the `coherence-cache-config.xml` with a CacheStore, you can run the lab.

1. Stop any CacheServers that may still be running from the previous Lab.
2. If your database connection differs from the defaults, you must update the following EclipseLink JDBC properties in the `META-INF/persistence.xml` file:
   - **eclipselink.jdbc.driver**
   - **eclipselink.jdbc.url**
   - **eclipselink.jdbc.user**
   - **eclipselink.jdbc.password**
3. From the Project Explorer view, right-click the Lab 3 project and select **JPA>Generate Tables from Entities…** from the pop-up menu to reset the database.
4. Run **gridentity.example.StartCacheServer** to start the Coherence cache server.
5. Run **gridentity.example.InsertExample** to create example entities in the database and populate the Coherence cache. In the **InsertExample** console you will see messages showing entities being put into Coherence under their primary key, but no **INSERT** statements.

   ```
   [EL Fine]: Coherence(Employee)::ConditionalPut: 1 value:
   readwrite.model.Employee@b9b618(1: Smith, Bob)
   [EL Fine]: Coherence(PhoneNumber)::ConditionalPut: 3 value:
   readwrite.model.PhoneNumber@800aa1(3: Work: 613-5558812)
   [EL Fine]: Coherence(Address)::ConditionalPut: 2 value:
   readwrite.model.Address@169dd64(Toronto)
   ```

   You will see database queries in the CacheServer console because with **CoherenceReadWrite** configured, all queries in the **QueryExample** client program are being directed to Coherence. When an entity is put into Coherence, the CacheStore determines if the put entity is new, which will produce an **INSERT** statement, or has been updated, which will produce an **UPDATE** statement.
6. Since the Grid Entity configuration has the same read behavior as the "Lab 2: "Grid Read" Configuration" configuration, running **gridentity.example.QueryExample1** will produce the same results as **gridread.example.QueryExample1**. In the console, there are no **SELECT** statements resulting from the JQPL query `select e from Employee e`.

Summary

TopLink Grid provides integration between EclipseLink JPA and Oracle Coherence. This lab demonstrated configurations ranging from using Coherence as a shared L2 cache to using Coherence as data source with JPQL queries translated to Filters executed in the grid.

As these labs illustrated, configuring Coherence with TopLink Grid is simple, straight forward, and introduces limited changes to JPA applications.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at http://www.fcc.gov/cgb/consumerfacts/trs.html, and a list of phone numbers is available at http://www.fcc.gov/cgb/dro/trsphonebk.html.

Integration Guide for Oracle TopLink with Coherence Gird, Volume 1, 11*g* Release 1 (11.1.1)
E16596-01

Copyright © 1997, 2010, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the

additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.