Fineract 1.x - ORM Migration from OpenJPA to EclipseLink

OpenJpa and EclispeLink are both free opensource JPA service providers. Over the years Fineract has been using openJpa as their JPA provider to handle various transactions with mysql database which is what we are currently using.

OpenJpa has worked well in Fineract for ages but for some reasons, we had to switch it. The reasons are as follows:

- OpenJpa is reaching it's end of life cycle as there are less and less community involvement.
- We have noticed a lot of users while in production tend to switch back to hibernate and switching to eclipselink will hopefully save them that trouble as EclipseLink has increasing performance and optimisation properties.
- EclipseLink is the reference implementation for JPA and is hence more standard compliant.
- There are fewer trade-offs between EclipseLink and Hibernate

What's New/Different?

Migrating from OpenJPA a few things were removed and couple of adjustment and configurations made for EclispeLink.

1. **MySQLDB Dictionary Support:** With OpenJPA there was no native support for MySQL DB, so we had to setup DB dictionary. EclipseLink comes natively with support for MySQL and other DBs.
2. **Enhancement / Weaving:** Just like OpenJPA, EclipseLink does not support out of the box runtime enhancement on tomcat server. For this, there a couple of options which were tried but we settled for a static or compile time weaving. We able to achieve this through a gradle task which handle this manually. Here is a peek of the task:

```
configurations {
    weaver.extendsFrom implementation
    integrationTestCompile.extendsFrom testImplementation
    integrationTestRuntime.extendsFrom testRuntime
}

task weave(type: JavaExec, dependsOn: [
    compileJava,
    processResources
]) {
    main = 'org.eclipse.persistence.tools.weaving.jpa.StaticWeave'
    classpath configurations.weaver
    args '-persistenceinfo'
    args processResources.destinationDir.absolutePath
    args '-classpath'
    args configurations.compile.incoming.files.asPath
    args '-loglevel'
    args 'INFO' // logging level at "FINE" shows alot of output to console.
    args sourceSets.main.java.outputDir.absolutePath
    args sourceSets.main.java.outputDir.absolutePath

    inputs.files fileTree(processResources.destinationDir).matching({pattern -> pattern.include('**/META-
INF/persistence.xml')})
}
classes.dependsOn weave
```

3. **Entity Persistence:** Unlike OpenJPA, EclipseLink has a bit of trouble persisting entities especially in a cases of OneToMany or ManyToOne or self referencing entities as most often transactions are not committed until the very end of the transaction. To overcome this, "saveAndFlush" was used in some parts of the code instead of "save" as saveAndFlush committed changes immediately.
4. **Default Constructors:** Unlike OpenJPA, EclispeLink is not tolerant to the absence of a default no-arg constructor in entity classes. OpenJPA enhancement handled that but EclipseLink does not have a thing for that. This should be watch for when creating a new entity.
5. **Common negligence errors:** EclipseLink with it's high compliance with standards tends to non-tolerant of some mistakes which sometimes escape our attention, like duplicate column names, wrong SQL syntax in case of native queries. We picked up some of these during the migration.

Some Flip Flops

- Some many to one relationships fail to add foreign key to owning side of the relationship. This is a current issue that is being looked at.
- There seems to be some performance issue at the level of schedular jobs where queries sometime take longer than expected.
- Reverting to OpenJPA might not be as easy as a lot of helper code for OpenJPA has been discarded.
- Column "*isEqualAmortization*" in the *LoanProduct* entity class has been made transient because it is duplicated in it embedded entity *LoanProduct RelatedDetail* which also has a column *isEqualAmortization*. Checking the database and the data stored, I judged alien column is the one in *Loan Product*. ***This could use extra pair of eyes.***

Coming up next

- JPA optimisations for improved performance.
- Increase test coverage to assess the reliability of the migration.