JPA and Enums via @Enumerated

Example jpa-enumerated can be browsed at https://github.com/apache/tomee/tree/master/examples/jpa-enumerated

It can sometimes be desirable to have a Java `enum` type to represent a particular column in a database. JPA supports converting database data to and from Java `enum` types via the `@javax.persistence.Enumerated` annotation.

This example will show basic `@Enumerated` usage in a field of an `@Entity` as well as `enum`s as the parameter of a `Query`. We'll also see that the actual database representation can be effectively `String` or `int`.

Enum

For our example we will leverage the familiar `Movie` entity and add a new field to represent the MPAA.org rating of the movie. This is defined via a simple `enum` that requires no JPA specific annotations.

```
public enum Rating {
    UNRATED,
    G,
    PG,
    PG13,
    R,
    NC17
}
```

@Enumerated

In our `Movie` entity, we add a `rating` field of the enum type `Rating` and annotate it with `@Enumerated(EnumType.STRING)` to declare that its value should be converted from what is effectively a `String` in the database to the `Rating` type.

```
@Entity
public class Movie {

    @Id
    @GeneratedValue
    private int id;
    private String director;
    private String title;
    private int year;

    @Enumerated(EnumType.STRING)
    private Rating rating;
```

```
    public Movie() {
    }

    public Movie(String director, String title, int year, Rating rating) {
        this.director = director;
        this.title = title;
        this.year = year;
        this.rating = rating;
    }

    public String getDirector() {
        return director;
    }

    public void setDirector(String director) {
        this.director = director;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public int getYear() {
        return year;
    }

    public void setYear(int year) {
        this.year = year;
    }

    public Rating getRating() {
        return rating;
    }

    public void setRating(Rating rating) {
        this.rating = rating;
    }
}
```

The above is enough and we are effectively done. For the sake of completeness we'll show a sample `Query`

Enum in JPQL Query

Note the `findByRating` method which creates a `Query` with a `rating` named parameter. The key thing

to notice is that the `rating` enum instance itself is passed into the `query.setParameter` method, **not** `rating.name()` or `rating.ordinal()`.

Regardless if you use `EnumType.STRING` or `EnumType.ORDINAL`, you still always pass the enum itself in calls to `query.setParameter`.

```
@Stateful
public class Movies {

    @PersistenceContext(unitName = "movie-unit", type = PersistenceContextType
.EXTENDED)
    private EntityManager entityManager;

    public void addMovie(Movie movie) {
        entityManager.persist(movie);
    }

    public void deleteMovie(Movie movie) {
        entityManager.remove(movie);
    }

    public List<Movie> findByRating(Rating rating) {
        final Query query = entityManager.createQuery("SELECT m FROM Movie as m WHERE
m.rating = :rating");
        query.setParameter("rating", rating);
        return query.getResultList();
    }

    public List<Movie> getMovies() throws Exception {
        Query query = entityManager.createQuery("SELECT m from Movie as m");
        return query.getResultList();
    }

}
```

EnumType.STRING vs EnumType.ORDINAL

It is a matter of style how you would like your `enum` data represented in the database. Either `name()` or `ordinal()` are supported:

- `@Enumerated(EnumType.STRING) Rating rating` the value of `rating.name()` is written and read from the corresponding database column; e.g. `G`, `PG`, `PG13`
- `@Enumerated(EnumType.ORDINAL) Rating rating` the value of `rating.ordinal()` is written and read from the corresponding database column; e.g. `0`, `1`, `2`

The default is `EnumType.ORDINAL`

There are advantages and disadvantages to each.

Disadvantage of EnumType.ORDINAL

A disadvantage of `EnumType.ORDINAL` is the effect of time and the desire to keep `enums` in a logical order. With `EnumType.ORDINAL` any new enum elements must be added to the **end** of the list or you will accidentally change the meaning of all your records.

Let's use our `Rating` enum and see how it would have had to evolve over time to keep up with changes in the MPAA.org ratings system.

1980

```
public enum Rating {
    G,
    PG,
    R,
    UNRATED
}
```

1984 PG-13 is added

```
public enum Rating {
    G,
    PG,
    R,
    UNRATED,
    PG13
}
```

1990 NC-17 is added

```
public enum Rating {
    G,
    PG,
    R,
    UNRATED,
    PG13,
    NC17
}
```

If `EnumType.STRING` was used, then the enum could be reordered at anytime and would instead look as we have defined it originally with ratings starting at `G` and increasing in severity to `NC17` and eventually `UNRATED`. With `EnumType.ORDINAL` the logical ordering would not have withstood the test of time as new values were added.

If the order of the enum values is significant to your code, avoid `EnumType.ORDINAL`

Unit Testing the JPA @Enumerated

```
public class MoviesTest extends TestCase {

    public void test() throws Exception {

        final Properties p = new Properties();
        p.put("movieDatabase", "new://Resource?type=DataSource");
        p.put("movieDatabase.JdbcDriver", "org.hsqldb.jdbcDriver");
        p.put("movieDatabase.JdbcUrl", "jdbc:hsqldb:mem:moviedb");

        EJBContainer container = EJBContainer.createEJBContainer(p);
        final Context context = container.getContext();

        final Movies movies = (Movies) context.lookup("java:global/jpa-scratch/Movies
");

        movies.addMovie(new Movie("James Frawley", "The Muppet Movie", 1979, Rating.G
));
        movies.addMovie(new Movie("Jim Henson", "The Great Muppet Caper", 1981,
Rating.G));
        movies.addMovie(new Movie("Frank Oz", "The Muppets Take Manhattan", 1984,
Rating.G));
        movies.addMovie(new Movie("James Bobin", "The Muppets", 2011, Rating.PG));

        assertEquals("List.size()", 4, movies.getMovies().size());

        assertEquals("List.size()", 3, movies.findByRating(Rating.G).size());

        assertEquals("List.size()", 1, movies.findByRating(Rating.PG).size());

        assertEquals("List.size()", 0, movies.findByRating(Rating.R).size());

        container.close();
    }
}
```

Running

To run the example via maven:

```
cd jpa-enumerated
mvn clean install
```

Which will generate output similar to the following:

```
```

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running org.superbiz.jpa.enums.MoviesTest
Apache OpenEJB 4.0.0-beta-2    build: 20120115-08:26
http://tomee.apache.org/
INFO - openejb.home = /Users/dblevins/openejb/examples/jpa-enumerated
INFO - openejb.base = /Users/dblevins/openejb/examples/jpa-enumerated
INFO - Using 'javax.ejb.embeddable.EJBContainer=true'
INFO - Configuring Service(id=Default Security Service, type=SecurityService,
provider-id=Default Security Service)
INFO - Configuring Service(id=Default Transaction Manager, type=TransactionManager,
provider-id=Default Transaction Manager)
INFO - Configuring Service(id=movieDatabase, type=Resource, provider-id=Default JDBC
Database)
INFO - Found EjbModule in classpath: /Users/dblevins/openejb/examples/jpa-
enumerated/target/classes
INFO - Beginning load: /Users/dblevins/openejb/examples/jpa-enumerated/target/classes
INFO - Configuring enterprise application: /Users/dblevins/openejb/examples/jpa-
enumerated
INFO - Configuring Service(id=Default Stateful Container, type=Container, provider-
id=Default Stateful Container)
INFO - Auto-creating a container for bean Movies: Container(type=STATEFUL, id=Default
Stateful Container)
INFO - Configuring Service(id=Default Managed Container, type=Container, provider-
id=Default Managed Container)
INFO - Auto-creating a container for bean org.superbiz.jpa.enums.MoviesTest:
Container(type=MANAGED, id=Default Managed Container)
INFO - Configuring PersistenceUnit(name=movie-unit)
INFO - Auto-creating a Resource with id 'movieDatabaseNonJta' of type 'DataSource for
'movie-unit'.
INFO - Configuring Service(id=movieDatabaseNonJta, type=Resource, provider-
id=movieDatabase)
INFO - Adjusting PersistenceUnit movie-unit <non-jta-data-source> to Resource ID
'movieDatabaseNonJta' from 'movieDatabaseUnmanaged'
INFO - Enterprise application "/Users/dblevins/openejb/examples/jpa-enumerated"
loaded.
INFO - Assembling app: /Users/dblevins/openejb/examples/jpa-enumerated
INFO - PersistenceUnit(name=movie-unit,
provider=org.apache.openjpa.persistence.PersistenceProviderImpl) - provider time 406ms
INFO - Jndi(name="java:global/jpa-enumerated/Movies!org.superbiz.jpa.enums.Movies")
INFO - Jndi(name="java:global/jpa-enumerated/Movies")
INFO - Created Ejb(deployment-id=Movies, ejb-name=Movies, container=Default Stateful
Container)
INFO - Started Ejb(deployment-id=Movies, ejb-name=Movies, container=Default Stateful
Container)
INFO - Deployed Application(path=/Users/dblevins/openejb/examples/jpa-enumerated)
INFO - Undeploying app: /Users/dblevins/openejb/examples/jpa-enumerated
INFO - Closing DataSource: movieDatabase
INFO - Closing DataSource: movieDatabaseNonJta
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.831 sec
```

```
Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```