Mapping JPA in Your Database With the Time API

Throughout application development, you need to make CRUD operations on our stored data. Postgres and the Time API can help you map JPA in your database.

by Tomasz J-ski · Aug. 09, 17 · Database Zone · Tutorial

Many times, we encounter the need to store date and time in our database. Throughout application development, we need to make CRUD operations on that stored data. In my case, I use Spring Boot with Spring Data as a persistence layer. I'm still not sure whether the word “layer” is truly appropriate in modern development techniques, but nevertheless, we'll say that Spring Data is my persistence layer.

As a database, I use Postgres, which has quite a few nice features. And what's most

As a database, I use Postgres, which has quite a few nice features. And what's most important is that it's free!

A detailed technology stack includes:

- PostgreSQL DB.
- Spring Boot 1.5.3.
- Spring Data JPA.
- Lombok (since I do not like to write getters, setters, and so on).
- Angular 2 on the front end (but that is not important in this post).

Problem

The first problem that I encounter is JPA mapping types in my database. I have a Timestamp column type, which fits all my needs at the database level — it stores date and time with the appropriate time zones, so everything that I need is right there.

But JPA (the Hibernate implementation in our case) implements a Postgres SQL timestamp as `java.sql.Timestamp` or `java.sql.Date`, which is not what I want in my Spring Boot application.

I mean, I *could* deal with `ava.sql.*` types, but as of Java 8, we have

`LocalDate` and `LocalDateTime` classes with nice new features like `plusDays` methods.

Generally, I would like to have a new Time API in my application.

So this is what our database table looks like:

```
CREATE TABLE public.tablea
  (
     id      INT4 NOT NULL DEFAULT NEXTVAL('terminykursow_seq'::regclass),
     terma   TIMESTAMP NULL,
     termb   TIMESTAMP NULL,
     idother INT4 NULL
  );
```

And this is the corresponding entity:

```
@Entity
@Data
@Table(name = "tableA", schema = "public")
public class EntityA implements Serializable {
 @Id
 @SequenceGenerator(name = "tableA_seq", sequenceName = "tableA_seq", allocationSiz
 @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "tableA_seq")
 private Integer id;
 private LocalDateTime terminod;
 private LocalDateTime termindo;
 …
 other entity stuff...
}
```

Solution

Solution

Without any changes, the code compiles and the application starts correctly... but there are several errors, like:

```
“2017-08-08 18:32:06.911 ERROR 28428 --- [nio-8472-exec-2] o.a.c.c.C.[.[.[/].[dispa

org.postgresql.util.PSQLException: ERROR: operator does not exist: timestamp withou
```

The error does not tell exactly what the problem is, which is, in my opinion, the weakest characteristic of Spring Boot and Spring Data JPA logging.

Nevertheless, I decided to add some converters as described in the JPA documentation.

Tip 1: Always Use Appropriate Documentation

Please note that `Converter` is a class that implements the `javax.persistence.AttributeConverter` interface. You do not need any annotations on that class.

So my implementation looks like this:

```
import javax.persistence.AttributeConverter;
import java.sql.Timestamp;
import java.time.LocalDateTime;

public class LocalDateTimeConverter implements AttributeConverter < LocalDateTime,
 @Override
 public Timestamp convertToDatabaseColumn(LocalDateTime attribute) {
  return attribute != null ? Timestamp.valueOf(attribute) : null;
 }
```

```
@Override
public LocalDateTime convertToEntityAttribute(Timestamp dbData) {
 return dbData != null ? dbData.toLocalDateTime() : null;
}
}
```

At this point, all I need to add is an annotation to my `Entity` fields that drives to the `Converter` class:

```
@Convert(converter = LocalDateTimeConverter.class)
```

The final Entity looks like this:

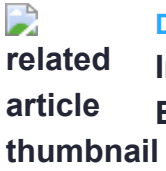
```
@Entity
@Data
@Table(name = "tableA", schema = "public")
public class EntityA implements Serializable {
 @Id
 @SequenceGenerator(name = "tableA_seq", sequenceName = "tableA_seq", allocationSiz
 @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "tableA_seq")
 private Integer id;
 @Convert(converter = LocalDateTimeConverter.class)
 private LocalDateTime terminod;
 @Convert(converter = LocalDateTimeConverter.class)
 private LocalDateTime termindo;
 …
 other entity stuff...
}
```

And *boom*! Everything works as expected.

Summary

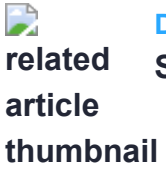
There are many good mechanisms in Java 8 that you can use in your Spring Data application. One of my favorites is the Time API and the `Converter` mechanism. With these, you are no longer sentenced to `javax.sql.*` classes.

Like This Article? Read More From DZone



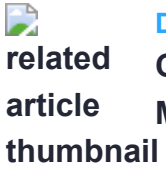
DZone Article

Implement GraphQL With Spring Boot by Connecting to Oracle



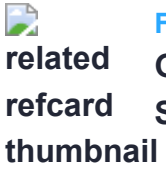
DZone Article

Specifications to the Rescue



DZone Article

Configuring Spring Boot for Microsoft SQL Server



Free DZone Refcard

Getting Started With Distributed SQL

Topics: DATABASE, JPA, SPRING BOOT, TIME API, TUTORIAL

Published at DZone with permission of Tomasz J-ski . See the original article here.
Opinions expressed by DZone contributors are their own.

Database Partner Resources