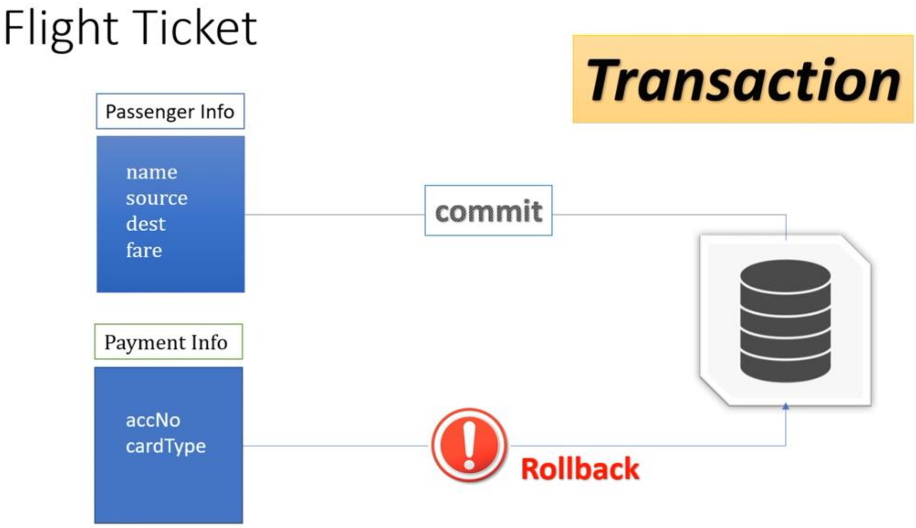
Spring Boot- Spring Data JPA Transaction Management @Transactional

The Java Persistence API (**JPA**) is the standard way of persisting Java objects into relational databases. The **JPA** consists of two parts: a mapping subsystem to map classes onto relational tables as well as an EntityManager API to access the objects, define and execute queries, and more.

Example of scenario :

(https://www.youtube.com/watch?v=95kxPSbHzVg)

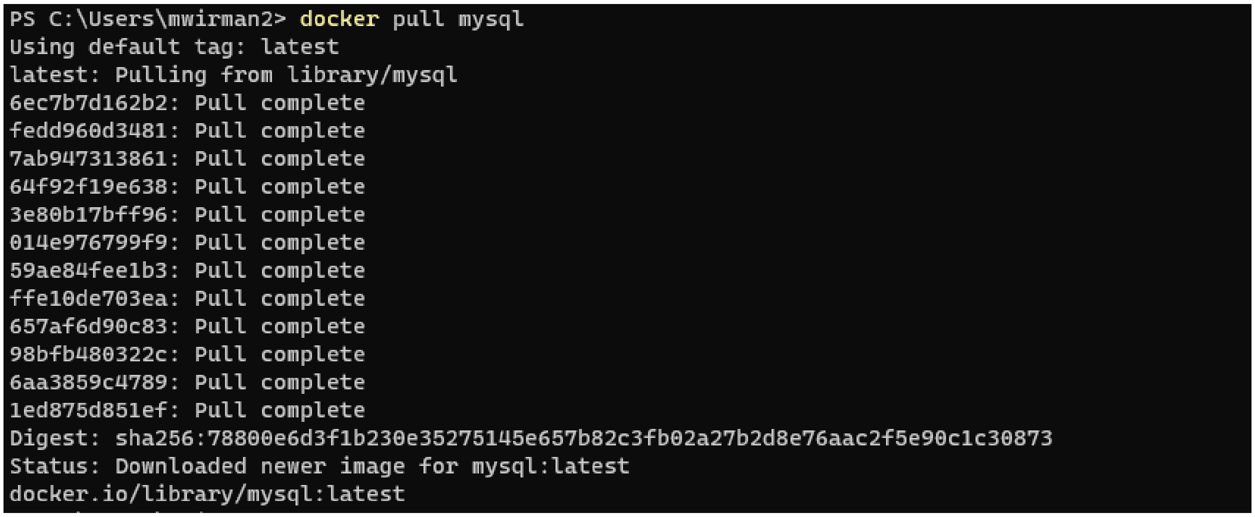
The passenger Info will not be saved until the Payment Info is successfully save



1) Prepare Database

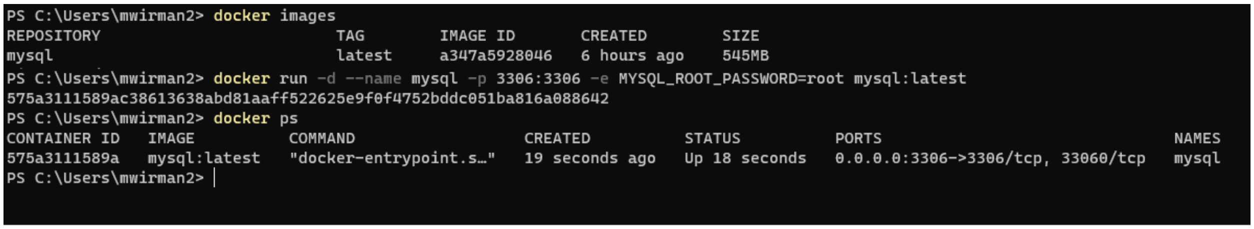
a) Prepare Docker Constainers for mySQL Database

```
$ docker pull mysql
```



```
PS C:\Users\mwirman2> docker pull mysql
Using default tag: latest
latest: Pulling from library/mysql
6ec7b7d162b2: Pull complete
fedd960d3481: Pull complete
7ab947313861: Pull complete
64f92f19e638: Pull complete
3e80b17bff96: Pull complete
014e976799f9: Pull complete
59ae84fee1b3: Pull complete
ffe10de703ea: Pull complete
657af6d90c83: Pull complete
98bfb480322c: Pull complete
6aa3859c4789: Pull complete
1ed875d851ef: Pull complete
Digest: sha256:78800e6d3f1b230e35275145e657b82c3fb02a27b2d8e76aac2f5e90c1c30873
Status: Downloaded newer image for mysql:latest
docker.io/library/mysql:latest
```

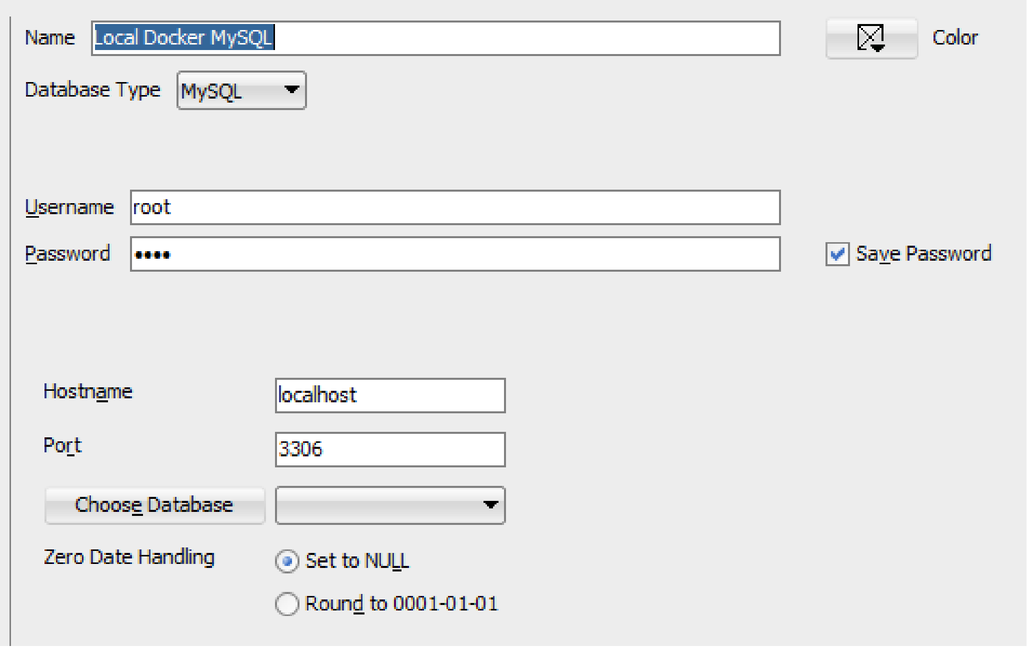
```
$ docker run -d --name mysql -p 3306:3306 mysql:latest
```



```
PS C:\Users\mwirman2> docker images
REPOSITORY                        TAG       IMAGE ID       CREATED       SIZE
mysql                             latest    a347a5928046   6 hours ago   545MB
PS C:\Users\mwirman2> docker run -d --name mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=root mysql:latest
575a3111589ac38613638abd81aaff522625e9f0f4752bddc051ba816a088642
PS C:\Users\mwirman2> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                               NAMES
575a3111589a   mysql:latest   "docker-entrypoint.s…"   19 seconds ago   Up 18 seconds   0.0.0.0:3306->3306/tcp, 33060/tcp   mysql
PS C:\Users\mwirman2> |
```

b) Create Database

Open SQL Developer



Name: Local Docker MySQL

Database Type: MySQL

Username: root

Password: ••••

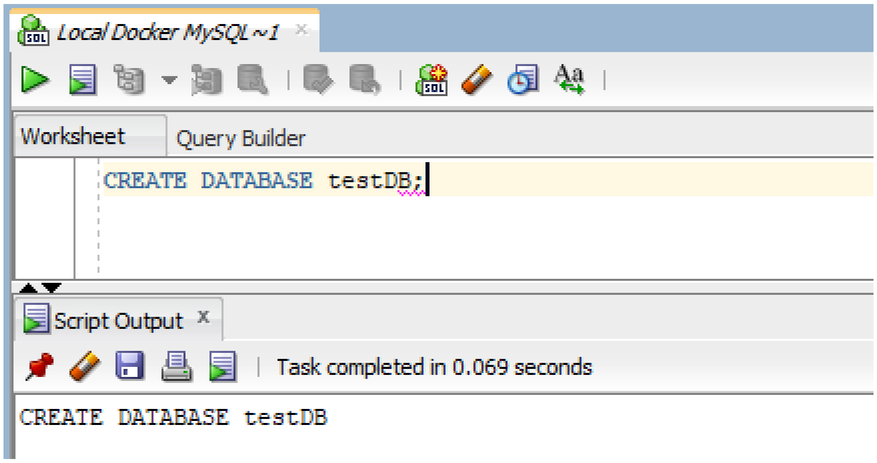
☑ Save Password

Hostname: localhost

Port: 3306

Zero Date Handling: ● Set to NULL ○ Round to 0001-01-01

Create Database



```
CREATE DATABASE testDB;
```

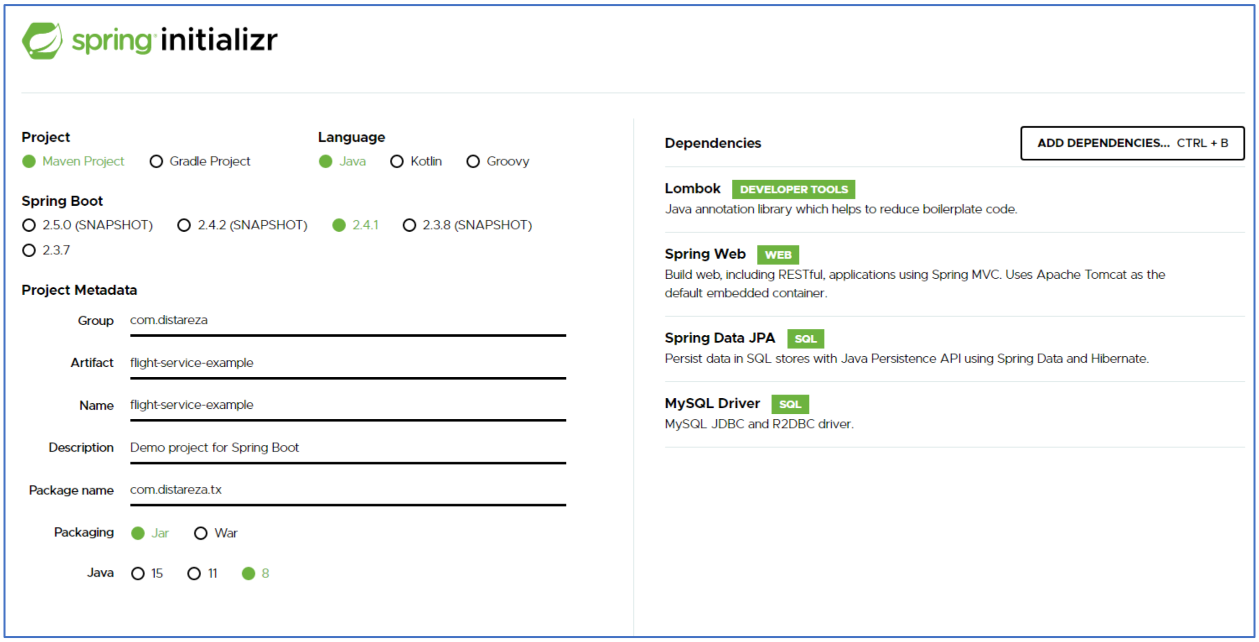
Task completed in 0.069 seconds

```
CREATE DATABASE testDB
```

2) Prepare Spring Boot Application

a) Generate Spring Boot Project

Open url https://start.spring.io to generate Java Project



spring initializr

Project

● Maven Project ○ Gradle Project

Language

● Java ○ Kotlin ○ Groovy

Spring Boot

○ 2.5.0 (SNAPSHOT) ○ 2.4.2 (SNAPSHOT) ● 2.4.1 ○ 2.3.8 (SNAPSHOT) ○ 2.3.7

Project Metadata

Group	com.distareza
Artifact	flight-service-example
Name	flight-service-example
Description	Demo project for Spring Boot
Package name	com.distareza.tx
Packaging	● Jar ○ War
Java	○ 15 ○ 11 ● 8

Dependencies

Lombok DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.

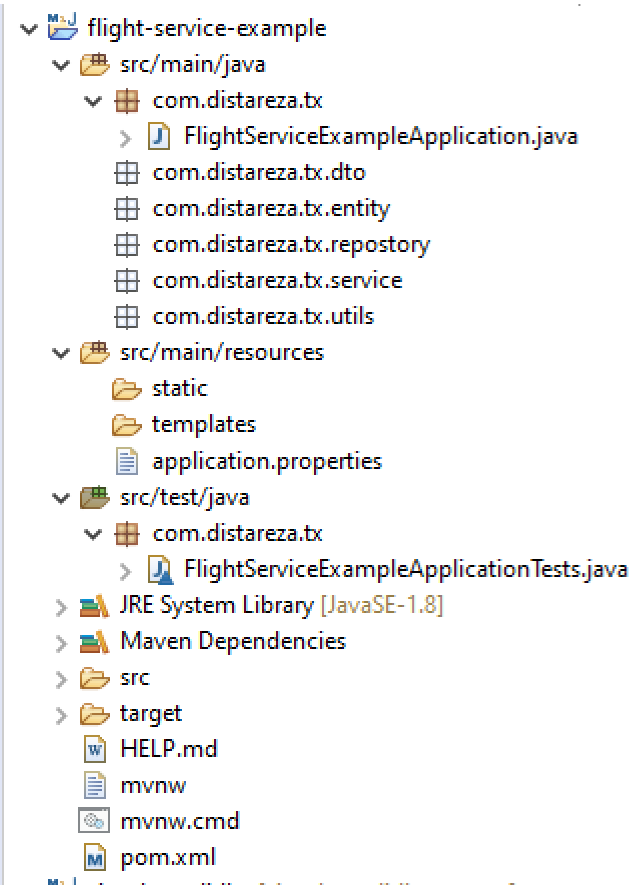
Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

MySQL Driver SQL
MySQL JDBC and R2DBC driver.

Click Generate to Download the Source Code

b) Import The Project source code into Eclipse



```
flight-service-example
  src/main/java
    com.distareza.tx
      FlightServiceExampleApplication.java
    com.distareza.tx.dto
    com.distareza.tx.entity
    com.distareza.tx.repostory
    com.distareza.tx.service
    com.distareza.tx.utils
  src/main/resources
    static
    templates
    application.properties
  src/test/java
    com.distareza.tx
      FlightServiceExampleApplicationTests.java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

Next, Prepare **Entity, Repo, DTO, Exception, Utils, and Service** class as follows
https://github.com/Java-Techie-jt/spring-transaction-example

c) Prepare Entities Class

Entities in JPA are nothing but POJOs representing data that can be persisted to the database. An entity represents a table stored in a database. Every instance of an entity represents a row in the table.

com.distareza.tx.entity.PassengerInfo.java

```
package com.distareza.tx.entity;

import java.util.Date;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

import com.fasterxml.jackson.annotation.JsonFormat;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "PASSENGER_INFO")
public class PassengerInfo {

    @Id
    @GeneratedValue
    private Long pId;
    private String name;
    private String email;
    private String source;
    private String Destination;
    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "dd-MM-yyyy")
    private Date travelDate;
    private String pickupTime;
    private String arrivalTime;
    private double fare;

}
```

com.distareza.tx.entity.PaymentInfo.java

```
package com.distareza.tx.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

import org.hibernate.annotations.GenericGenerator;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "PAYMENT_INFO")
public class PaymentInfo {

    @Id
    @GeneratedValue(generator = "uuid2")
    @GenericGenerator(name = "uuid2", strategy = "org.hibernate.id.UUIDGenerator")
    private String paymentId;
    private String accountNo;
    private double amount;
    private String cardType;
    private Long passengerId;

}
```

Annotation @Data , @AllArgsConstructor and @NoArgsConstructor are the annotation provided by Lombok project (https://projectlombok.org/features/Data)

@**Data is** a convenient shortcut annotation that bundles the features of **@ToString** , **@EqualsAndHashCode** , **@Getter** / **@Setter** and **@RequiredArgsConstructor** together: In other words, @**Data** generates all the boilerplate that **is** normally associated with simple POJOs (Plain Old Java Objects) and beans: getters for all fields, ...

To enable it on Eclipse you need to install it on your system. Upgrade **Lombok** as a dependency and as a IDE plugin (IntelliJ, NetBeans, Eclipse) and enable Annotation Processing in IDEs settings. Latest version of **Lombok** and/or IntelliJ plugin perfectly supports **Java 11**.

Starting with a fresh eclipse installation you, in fact, need to "install" Lombok before being able to use it.

How to install lombok:
Go where you Lombok jar is (e.g. (e.g. you can find in ~/.m2/repository/org/projectlombok/lombok/1.16.10/lombok-1.16.10.jar),
run it (Example: java -jar lombok-1.16.10.jar). A window should appear, browse to your eclipse.exe location.
Click on install.
Launch Eclipse, update project configuration on all projects and you ready to go.

`@Entity` annotation defines that a class can be mapped to a table.

When you create a new entity you have to do at least two things

1. annotated it with `@Entity`
2. create an id field and annotate it with `@Id`

Anything else is optional, for example table name is derived from entity class name (and therefore `@Table` annotation can be optional), table's columns are derived from entities variables (and therefore `@Column` annotation can be optional), and so on ...

JPA is trying to provide a fast and easy start to developers who want to learn/use this API, and giving developers option to configure as few things as possible to make something functional is one of the ways how this API wants to achieve this "easy to use/learn" goal. Hence the `@Entity` annotation (together with `@Id` annotation) is the minimum you have to do in order to create an entity.
The `@Id` annotation is inherited from `javax.persistence.Id`, indicating the member field below is the primary key of current entity. Hence your Hibernate and spring framework as well as you can do some `reflect` works based on this annotation. for details please check javadoc for Id

The `@GeneratedValue` annotation is to configure the way of increment of the specified column(field). For example when using `Mysql`, you may specify `auto_increment` in the definition of table to make it self-incremental, and then use

@GeneratedValue(strategy = GenerationType.IDENTITY) in the Java code to denote that you also acknowledged to use this database server side strategy. Also, you may change the value in this annotation to fit different requirements.

1. Define Sequence in database

For instance, Oracle has to use `sequence` as increment method, say we create a sequence in Oracle:

```
create sequence oracle_seq;
```

2. Refer the database sequence

Now that we have the sequence in database, but we need to establish the relation between Java and DB, by using `@SequenceGenerator`:

```
@SequenceGenerator(name="seq",sequenceName="oracle_seq")
```

`sequenceName` is the real name of a sequence in Oracle, `name` is what you want to call it in Java. You need to specify `sequenceName` if it is different from `name`, otherwise just use `name`. I usually ignore `sequenceName` to save my time.

3. Use sequence in Java

Finally, it is time to make use this sequence in Java. Just add `@GeneratedValue`:

```
@GeneratedValue(strategy=GenerationType.SEQUENCE, generator="seq")
```

The `generator` field refers to which sequence generator you want to use. Notice it is not the real sequence name in DB, but the name you specified in `name` field of `SequenceGenerator`.

4. Complete

So the complete version should be like this:

```
public class MyTable
{
   @Id
   @SequenceGenerator(name="seq",sequenceName="oracle_seq")
   @GeneratedValue(strategy=GenerationType.SEQUENCE, generator="seq")
   private Integer pid;
}
```

Now start using these annotations to make your JavaWeb development easier.

@JsonFormat is a Jackson annotation that is used to specify how to format fields and/or properties for JSON output.This annotation allows you to specify how to format *Date* and *Calendar* values according to a *SimpleDateFormat* format.

d) Prepare Repository Class and Database Connection

Repositories are **Java** interfaces that allow you as the developer to define a **data** access contract. The **Spring Data JPA** framework can then inspect that contract, and automatically build the interface implementation under the covers for you.

JpaRepository is **JPA** specific extension of **Repository**. It contains the full API of CrudRepository and PagingAndSortingRepository . So it contains API for basic CRUD operations and also API for pagination and sorting

Spring Data JPA focuses on using JPA to store data in a relational database. Its most compelling feature is the ability to create repository implementations automatically, at runtime, from a repository interface.

com.distareza.tx.repostiory.PassengerInfoRepository.java

```java
package com.distareza.tx.repostory;

import org.springframework.data.jpa.repository.JpaRepository;

import com.distareza.tx.entity.PassengerInfo;

public interface PassengerInfoRepository extends JpaRepository<PassengerInfo,Long> {

}
```

com.distareza.tx.repository.PaymentInfoRepository.java

```java
package com.distareza.tx.repostory;

import org.springframework.data.jpa.repository.JpaRepository;

import com.distareza.tx.entity.PaymentInfo;

public interface PaymentInfoRepository extends JpaRepository<PaymentInfo,String> {

}
```

e) Enable Persistance Data base in application Properties file

define the connection attributes in the `application.properties` file

application.properties

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url = jdbc:mysql://localhost:3306/testDB
spring.datasource.username = root
spring.datasource.password = root
spring.jpa.show-sql = true
spring.jpa.hibernate.ddl-auto = update
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect
server.port=9090
```

Here, `spring.jpa.hibernate.ddl-auto` can be `none`, `update`, `create`, or `create-drop`. See the [Hibernate documentation]() for details.

- `none`: The default for `MySQL`. No change is made to the database structure.
- `update`: Hibernate changes the database according to the given entity structures.
- `create`: Creates the database every time but does not drop it on close.
- `create-drop`: Creates the database and drops it when `SessionFactory` closes.

You must begin with either `create` or `update`, because you do not yet have the database structure. After the first run, you can switch it to `update` or `none`, according to program requirements. Use `update` when you want to make some change to the database structure.

The default for `H2` and other embedded databases is `create-drop`. For other databases, such as `MySQL`, the default is `none`.

com.distareza.tx.repository.TestPassangerInfoRepository.java

```
@PersistenceContext
private EntityManager em;

@Autowired
private PassengerInfoRepository repo;

@Test
public void testQueryPassengerInfo() throws Exception {
        SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
        PassengerInfo passengerInfo =
                new PassengerInfo(
                        null,
                        "Elton John",
                        "eltonjon@gmail.com",
                        "London",
                        "New York",
                        sdf.parse("01-02-2020"),
                        "09.00 AM",
                        "13.00 PM",
                        9000.0);
        repo.save(passengerInfo);

        Optional<PassengerInfo> passenger = repo.findById(1l);

        List<PassengerInfo> list = repo.findAll();

        TypedQuery<String> query =
                em.createQuery("SELECT email FROM PassengerInfo WHERE p_id = ?1", String.class);
        query.setParameter(1, 1l);

        List<String> listResult = query.getResultList();

}
```

f) Prepare Validation and Utility Class

com.distareza.tx.utils.PaymentUtils.java

```
package com.distareza.tx.utils;

import java.util.HashMap;
import java.util.Map;

import com.distareza.tx.exception.InsufficientAmountException;

public class PaymentUtils {

    private static Map<String, Double> paymentMap = new HashMap<>();

    static {
        paymentMap.put("acc1", 12000.0);
        paymentMap.put("acc2", 10000.0);
        paymentMap.put("acc3", 5000.0);
        paymentMap.put("acc4", 8000.0);
    }


    public static boolean validateCreditLimit(String accNo, double paidAmount) {
        if (paidAmount > paymentMap.get(accNo)) {
            throw new InsufficientAmountException("insufficient fund..!");
        } else {
            return true;
        }
    }

}
```

com.distareza.tx.exception.InsufficientAmountException.java

```
package com.distareza.tx.exception;

public class InsufficientAmountException extends RuntimeException {

    public InsufficientAmountException(String msg){
        super(msg);
    }

}
```

g) Prepare Request and Response DTO Class

com.dista.reza.tx.dto.FlightBookingRequest.java

```
package com.distareza.tx.dto;

import com.distareza.tx.entity.PassengerInfo;
import com.distareza.tx.entity.PaymentInfo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;


@Data
@AllArgsConstructor
@NoArgsConstructor
```

```
public class FlightBookingRequest {

    private PassengerInfo passengerInfo;
    private PaymentInfo paymentInfo;

}
```

com.distareza.tx.dto.FlightBookingAcknowledgement.java

```
package com.distareza.tx.dto;

import com.distareza.tx.entity.PassengerInfo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class FlightBookingAcknowledgement {

    private String status;
    private double totalFare;
    private String pnrNo;
    private PassengerInfo passengerInfo;

}
```

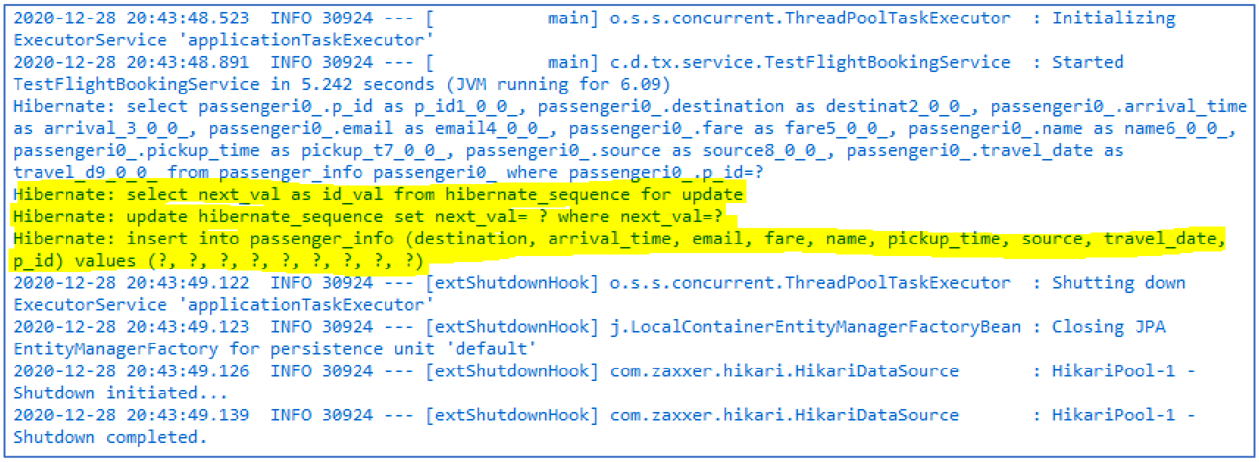
h) Prepare Service Class (Default / Transactional not enabled)

com.distareza.tx.service.FlightBookingService.java

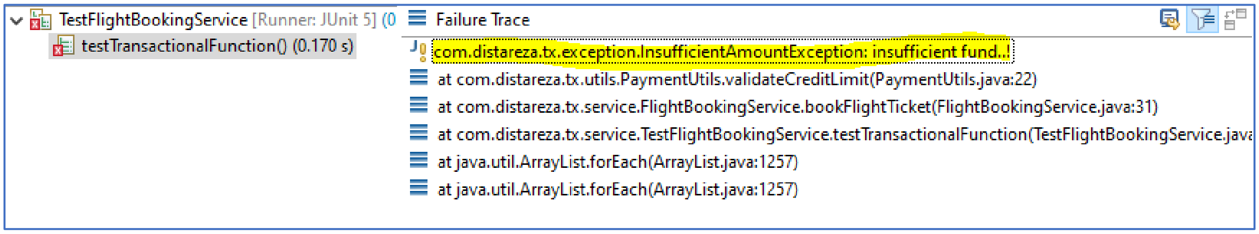
```
package com.distareza.tx.service;

import java.util.UUID;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.distareza.tx.dto.FlightBookingAcknowledgement;
import com.distareza.tx.dto.FlightBookingRequest;
import com.distareza.tx.entity.PassengerInfo;
import com.distareza.tx.entity.PaymentInfo;
import com.distareza.tx.repostory.PassengerInfoRepository;
import com.distareza.tx.repostory.PaymentInfoRepository;
import com.distareza.tx.utils.PaymentUtils;

@Service
public class FlightBookingService {

    @Autowired
    private PassengerInfoRepository passengerInfoRepository;
    @Autowired
    private PaymentInfoRepository paymentInfoRepository;

    public FlightBookingAcknowledgement bookFlightTicket(FlightBookingRequest request) {

        PassengerInfo passengerInfo = request.getPassengerInfo();
        passengerInfo = passengerInfoRepository.save(passengerInfo);
        PaymentInfo paymentInfo = request.getPaymentInfo();

        PaymentUtils.validateCreditLimit(paymentInfo.getAccountNo(), passengerInfo.getFare());

        paymentInfo.setPassengerId(passengerInfo.getPId());
        paymentInfo.setAmount(passengerInfo.getFare());
        paymentInfoRepository.save(paymentInfo);
        return new FlightBookingAcknowledgement(
                "SUCCESS",
                passengerInfo.getFare(),
                UUID.randomUUID().toString().split("-")[0],
                passengerInfo);

    }

}
```

Create and Run Test Class

```
package com.distareza.tx.service;

import static org.junit.jupiter.api.Assertions.assertTrue;

import java.text.SimpleDateFormat;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import com.distareza.tx.dto.FlightBookingRequest;
import com.distareza.tx.entity.PassengerInfo;
import com.distareza.tx.entity.PaymentInfo;

@SpringBootTest
public class TestFlightBookingService {

        @Autowired
        private FlightBookingService service;

        @Test
        public void testTransactionalFunction() throws Exception {
                SimpleDateFormat sdf  = new SimpleDateFormat("dd-MM-yyyy");

                FlightBookingRequest request = new FlightBookingRequest(
                                new PassengerInfo(-1l,
                                                "Jhon Doe",
                                                "doe@gmail.com",
                                                "Tokyo",
                                                "San fransisco",
                                                sdf.parse("31-01-2021"),
                                                "12.00 PM",
                                                "15.00 PM",
                                                50000.0),
                                new PaymentInfo(null, "acc1", 0.0, "Debit", null)
                                );
                service.bookFlightTicket(request );

                assertTrue(true);
        }

}
```



```
2020-12-28 20:43:48.523  INFO 30924 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing
ExecutorService 'applicationTaskExecutor'
2020-12-28 20:43:48.891  INFO 30924 --- [           main] c.d.tx.service.TestFlightBookingService  : Started
TestFlightBookingService in 5.242 seconds (JVM running for 6.09)
Hibernate: select passengeri0_.p_id as p_id1_0_0_, passengeri0_.destination as destinat2_0_0_, passengeri0_.arrival_time
as arrival_3_0_0_, passengeri0_.email as email4_0_0_, passengeri0_.fare as fare5_0_0_, passengeri0_.name as name6_0_0_,
passengeri0_.pickup_time as pickup_t7_0_0_, passengeri0_.source as source8_0_0_, passengeri0_.travel_date as
travel_d9_0_0_ from passenger_info passengeri0_ where passengeri0_.p_id=?
Hibernate: select next_val as id_val from hibernate_sequence for update
Hibernate: update hibernate_sequence set next_val= ? where next_val=?
Hibernate: insert into passenger_info (destination, arrival_time, email, fare, name, pickup_time, source, travel_date,
p_id) values (?, ?, ?, ?, ?, ?, ?, ?, ?)
2020-12-28 20:43:49.122  INFO 30924 --- [extShutdownHook] o.s.s.concurrent.ThreadPoolTaskExecutor  : Shutting down
ExecutorService 'applicationTaskExecutor'
2020-12-28 20:43:49.123  INFO 30924 --- [extShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA
EntityManagerFactory for persistence unit 'default'
2020-12-28 20:43:49.126  INFO 30924 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 -
Shutdown initiated...
2020-12-28 20:43:49.139  INFO 30924 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 -
Shutdown completed.
```

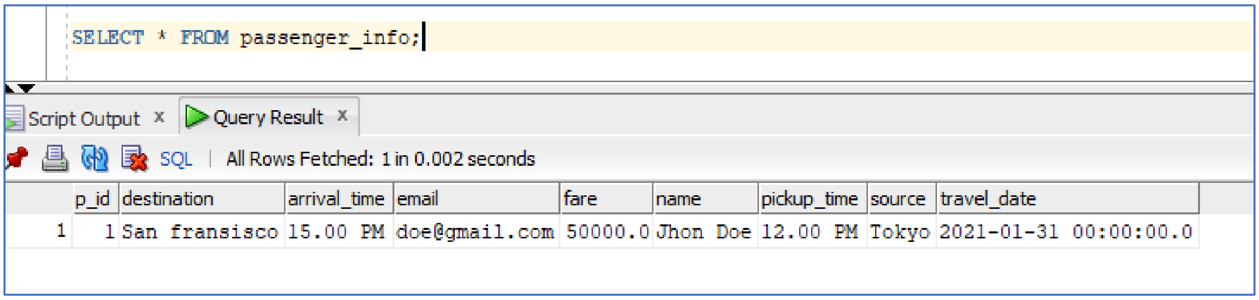


TestFlightBookingService [Runner: JUnit 5] (0

testTransactionalFunction() (0.170 s)

Failure Trace

```
com.distareza.tx.exception.InsufficientAmountException: insufficient fund..!
at com.distareza.tx.utils.PaymentUtils.validateCreditLimit(PaymentUtils.java:22)
at com.distareza.tx.service.FlightBookingService.bookFlightTicket(FlightBookingService.java:31)
at com.distareza.tx.service.TestFlightBookingService.testTransactionalFunction(TestFlightBookingService.java
at java.util.ArrayList.forEach(ArrayList.java:1257)
at java.util.ArrayList.forEach(ArrayList.java:1257)
```

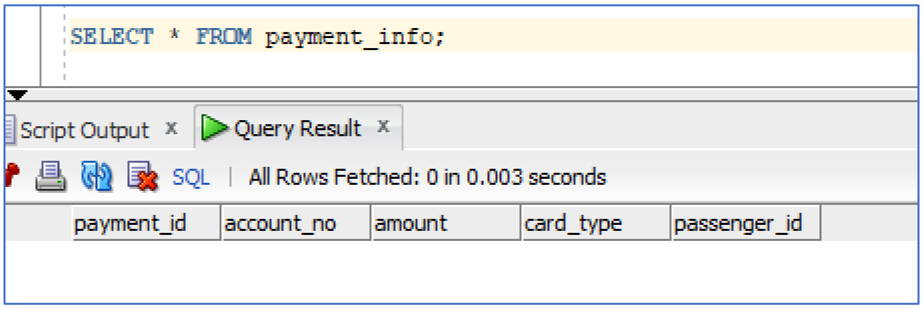


```
SELECT * FROM passenger_info;
```

Script Output × Query Result ×

SQL | All Rows Fetched: 1 in 0.002 seconds

	p_id	destination	arrival_time	email	fare	name	pickup_time	source	travel_date
1	1	San fransisco	15.00 PM	doe@gmail.com	50000.0	Jhon Doe	12.00 PM	Tokyo	2021-01-31 00:00:00.0

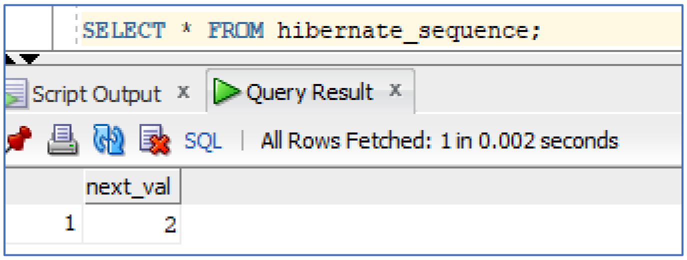


```
SELECT * FROM payment_info;
```

Script Output × Query Result ×

SQL | All Rows Fetched: 0 in 0.003 seconds

payment_id	account_no	amount	card_type	passenger_id



```
SELECT * FROM hibernate_sequence;
```

Script Output × Query Result ×

SQL | All Rows Fetched: 1 in 0.002 seconds

	next_val
1	2

i) Update Service Class with Transactional Enabled

com.distareza.tx.service.FlightBookingService.java

```
package com.distareza.tx.service;

...
import javax.transaction.Transactional;
import org.springframework.transaction.annotation.EnableTransactionManagement;
...

@Service
@EnableTransactionManagement
public class FlightBookingService {

    @Autowired
    private PassengerInfoRepository passengerInfoRepository;
    @Autowired
    private PaymentInfoRepository paymentInfoRepository;

    @Transactional
    public FlightBookingAcknowledgement bookFlightTicket(FlightBookingRequest request) {
        ...
    }

}
```

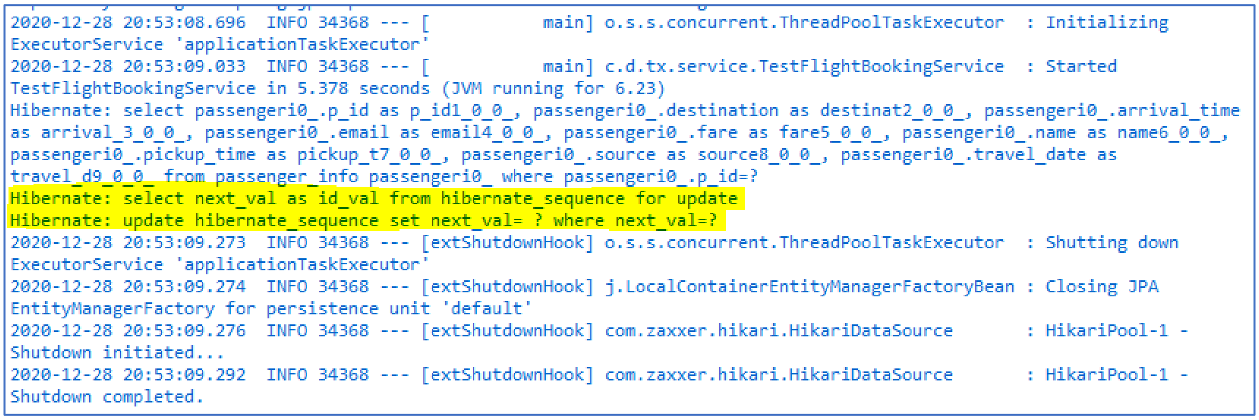
The annotation @EnableTransactionManagement tells Spring that classes with the @Transactional annotation should be wrapped with the Transactional Aspect. With this the @Transactional is now ready to be used.

The *@Transactional* annotation supports further configuration as well:

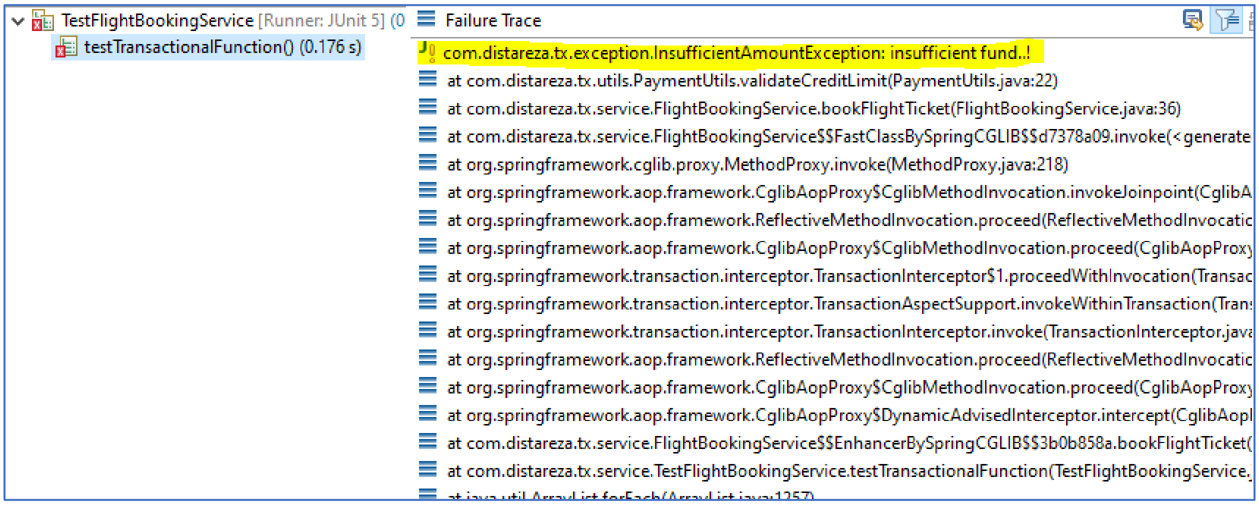
- the *Propagation Type* of the transaction
- the *Isolation Level* of the transaction
- a *Timeout* for the operation wrapped by the transaction
- a *readOnly flag* – a hint for the persistence provider that the transaction should be read only
- the *Rollback* rules for the transaction

Note that – by default, rollback happens for runtime, unchecked exceptions only. The checked exception does not trigger a rollback of the transaction. We can, of course, configure this behavior with the *rollbackFor* and *noRollbackFor* annotation parameters.

Run with the same test class

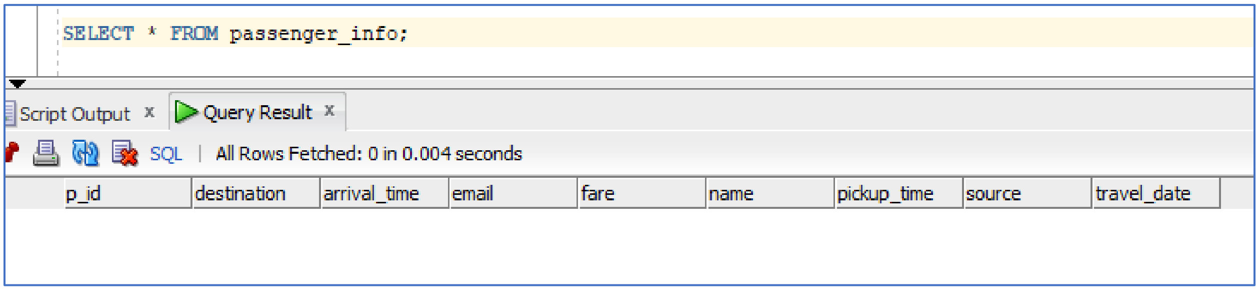


```
2020-12-28 20:53:08.696  INFO 34368 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing
ExecutorService 'applicationTaskExecutor'
2020-12-28 20:53:09.033  INFO 34368 --- [           main] c.d.tx.service.TestFlightBookingService  : Started
TestFlightBookingService in 5.378 seconds (JVM running for 6.23)
Hibernate: select passengeri0_.p_id as p_id1_0_0_, passengeri0_.destination as destinat2_0_0_, passengeri0_.arrival_time
as arrival_3_0_0_, passengeri0_.email as email4_0_0_, passengeri0_.fare as fare5_0_0_, passengeri0_.name as name6_0_0_,
passengeri0_.pickup_time as pickup_t7_0_0_, passengeri0_.source as source8_0_0_, passengeri0_.travel_date as
travel_d9_0_0_ from passenger_info passengeri0_ where passengeri0_.p_id=?
Hibernate: select next_val as id_val from hibernate_sequence for update
Hibernate: update hibernate_sequence set next_val= ? where next_val=?
2020-12-28 20:53:09.273  INFO 34368 --- [extShutdownHook] o.s.s.concurrent.ThreadPoolTaskExecutor  : Shutting down
ExecutorService 'applicationTaskExecutor'
2020-12-28 20:53:09.274  INFO 34368 --- [extShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA
EntityManagerFactory for persistence unit 'default'
2020-12-28 20:53:09.276  INFO 34368 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 -
Shutdown initiated...
2020-12-28 20:53:09.292  INFO 34368 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 -
Shutdown completed.
```



TestFlightBookingService [Runner: JUnit 5] (0
testTransactionalFunction() (0.176 s)
Failure Trace

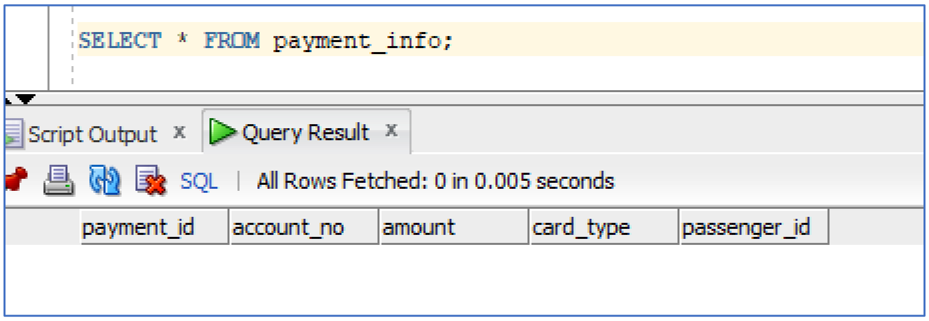
```
com.distareza.tx.exception.InsufficientAmountException: insufficient fund..!
at com.distareza.tx.utils.PaymentUtils.validateCreditLimit(PaymentUtils.java:22)
at com.distareza.tx.service.FlightBookingService.bookFlightTicket(FlightBookingService.java:36)
at com.distareza.tx.service.FlightBookingService$$FastClassBySpringCGLIB$$d7378a09.invoke(<generate
at org.springframework.cglib.proxy.MethodProxy.invoke(MethodProxy.java:218)
at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.invokeJoinpoint(CglibA
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocatic
at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy
at org.springframework.transaction.interceptor.TransactionInterceptor$1.proceedWithInvocation(Transac
at org.springframework.transaction.interceptor.TransactionAspectSupport.invokeWithinTransaction(Tran
at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocatic
at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy
at org.springframework.aop.framework.CglibAopProxy$DynamicAdvisedInterceptor.intercept(CglibAopl
at com.distareza.tx.service.FlightBookingService$$EnhancerBySpringCGLIB$$3b0b858a.bookFlightTicket(
at com.distareza.tx.service.TestFlightBookingService.testTransactionalFunction(TestFlightBookingService.
at java.util.ArrayList.forEach(ArrayList.java:1257)
```



```sql
SELECT * FROM passenger_info;
```

All Rows Fetched: 0 in 0.004 seconds

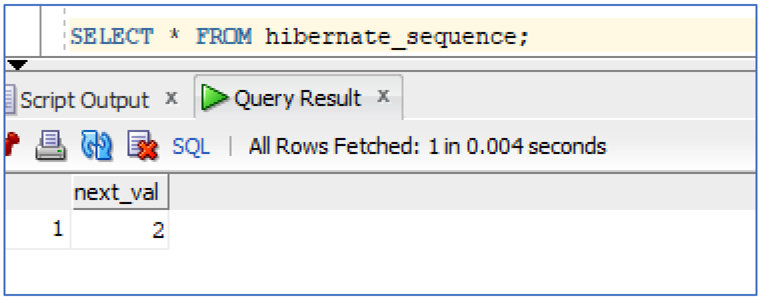
	p_id	destination	arrival_time	email	fare	name	pickup_time	source	travel_date



```sql
SELECT * FROM payment_info;
```

All Rows Fetched: 0 in 0.005 seconds

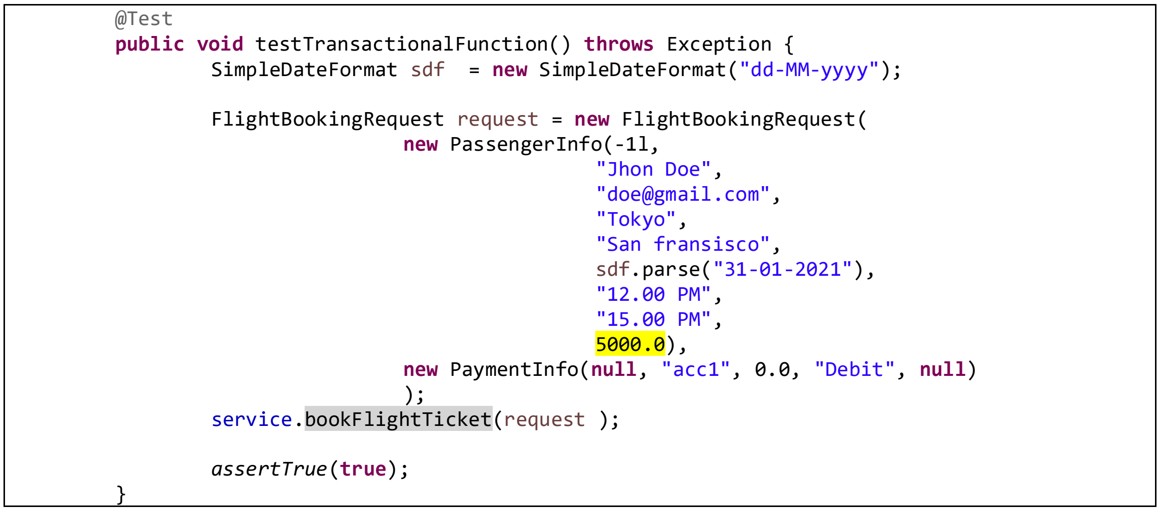
	payment_id	account_no	amount	card_type	passenger_id



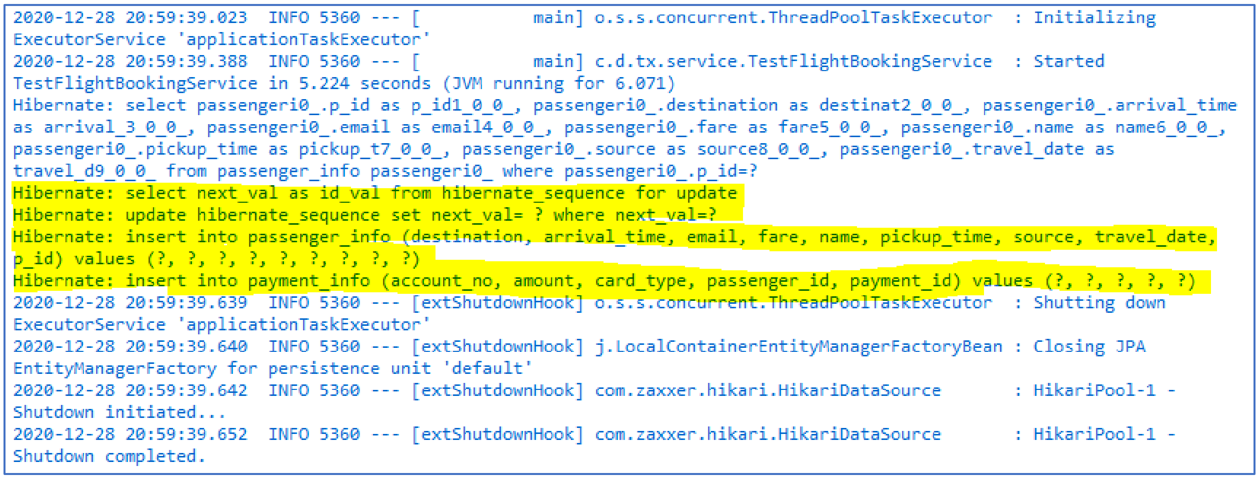
```
SELECT * FROM hibernate_sequence;
```

	next_val
1	2

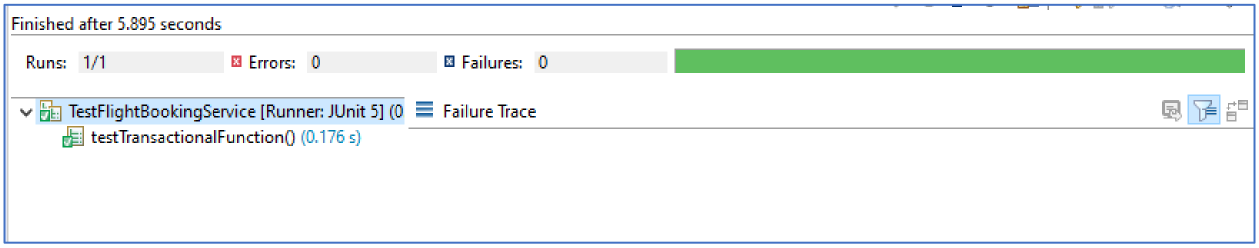
Test with Valid Data



```
@Test
public void testTransactionalFunction() throws Exception {
        SimpleDateFormat sdf  = new SimpleDateFormat("dd-MM-yyyy");

        FlightBookingRequest request = new FlightBookingRequest(
                        new PassengerInfo(-1l,
                                        "Jhon Doe",
                                        "doe@gmail.com",
                                        "Tokyo",
                                        "San fransisco",
                                        sdf.parse("31-01-2021"),
                                        "12.00 PM",
                                        "15.00 PM",
                                        5000.0),
                        new PaymentInfo(null, "acc1", 0.0, "Debit", null)
                        );
        service.bookFlightTicket(request );

        assertTrue(true);
}
```



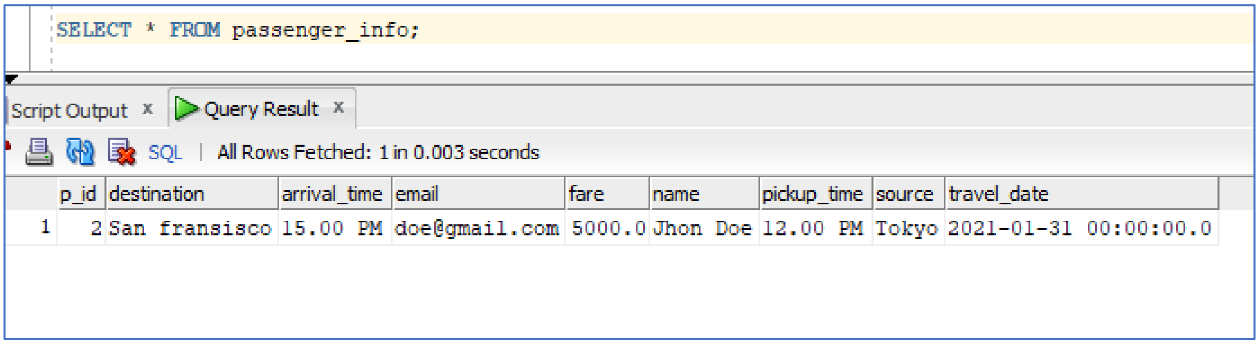
```
2020-12-28 20:59:39.023  INFO 5360 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing
ExecutorService 'applicationTaskExecutor'
2020-12-28 20:59:39.388  INFO 5360 --- [           main] c.d.tx.service.TestFlightBookingService  : Started
TestFlightBookingService in 5.224 seconds (JVM running for 6.071)
Hibernate: select passengeri0_.p_id as p_id1_0_0_, passengeri0_.destination as destinat2_0_0_, passengeri0_.arrival_time
as arrival_3_0_0_, passengeri0_.email as email4_0_0_, passengeri0_.fare as fare5_0_0_, passengeri0_.name as name6_0_0_,
passengeri0_.pickup_time as pickup_t7_0_0_, passengeri0_.source as source8_0_0_, passengeri0_.travel_date as
travel_d9_0_0_ from passenger_info passengeri0_ where passengeri0_.p_id=?
Hibernate: select next_val as id_val from hibernate_sequence for update
Hibernate: update hibernate_sequence set next_val= ? where next_val=?
Hibernate: insert into passenger_info (destination, arrival_time, email, fare, name, pickup_time, source, travel_date,
p_id) values (?, ?, ?, ?, ?, ?, ?, ?, ?)
Hibernate: insert into payment_info (account_no, amount, card_type, passenger_id, payment_id) values (?, ?, ?, ?, ?)
2020-12-28 20:59:39.639  INFO 5360 --- [extShutdownHook] o.s.s.concurrent.ThreadPoolTaskExecutor  : Shutting down
ExecutorService 'applicationTaskExecutor'
2020-12-28 20:59:39.640  INFO 5360 --- [extShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA
EntityManagerFactory for persistence unit 'default'
2020-12-28 20:59:39.642  INFO 5360 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 -
Shutdown initiated...
2020-12-28 20:59:39.652  INFO 5360 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 -
Shutdown completed.
```



Finished after 5.895 seconds

Runs: 1/1 ☒ Errors: 0 ☒ Failures: 0

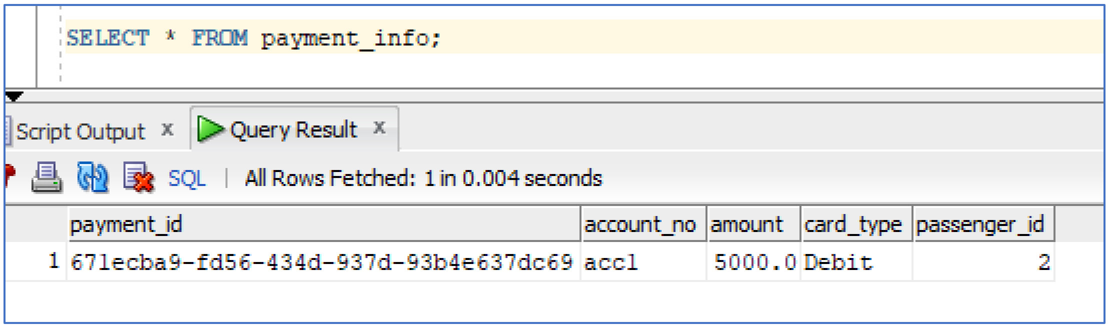
TestFlightBookingService [Runner: JUnit 5] (0
Failure Trace
testTransactionalFunction() (0.176 s)



```
SELECT * FROM passenger_info;
```

All Rows Fetched: 1 in 0.003 seconds

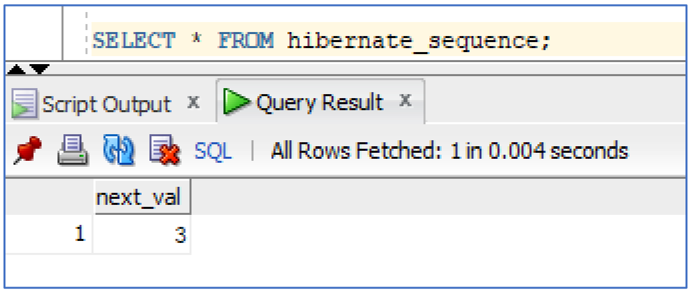
	p_id	destination	arrival_time	email	fare	name	pickup_time	source	travel_date
1	2	San fransisco	15.00 PM	doe@gmail.com	5000.0	Jhon Doe	12.00 PM	Tokyo	2021-01-31 00:00:00.0



```
SELECT * FROM payment_info;
```

All Rows Fetched: 1 in 0.004 seconds

	payment_id	account_no	amount	card_type	passenger_id
1	671ecba9-fd56-434d-937d-93b4e637dc69	acc1	5000.0	Debit	2



```
SELECT * FROM hibernate_sequence;
```

All Rows Fetched: 1 in 0.004 seconds

	next_val
1	3

j) Publish an Endpoint of controller class

com.distareza.tx.FlightServiceExampleApplication.java

```
package com.distareza.tx;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.transaction.annotation.EnableTransactionManagement;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import com.distareza.tx.dto.FlightBookingAcknowledgement;
import com.distareza.tx.dto.FlightBookingRequest;
import com.distareza.tx.service.FlightBookingService;

@SpringBootApplication
@RestController
@EnableTransactionManagement
public class FlightServiceExampleApplication {


        @Autowired
        private FlightBookingService service;
```

```
	@PostMapping("/bookFlightTicket")
	public FlightBookingAcknowledgement bookFlightTicket
	(@RequestBody FlightBookingRequest request){
		return service.bookFlightTicket(request);
	}

	public static void main(String[] args) {
		SpringApplication.run(FlightServiceExampleApplication.class, args);
	}

}
```

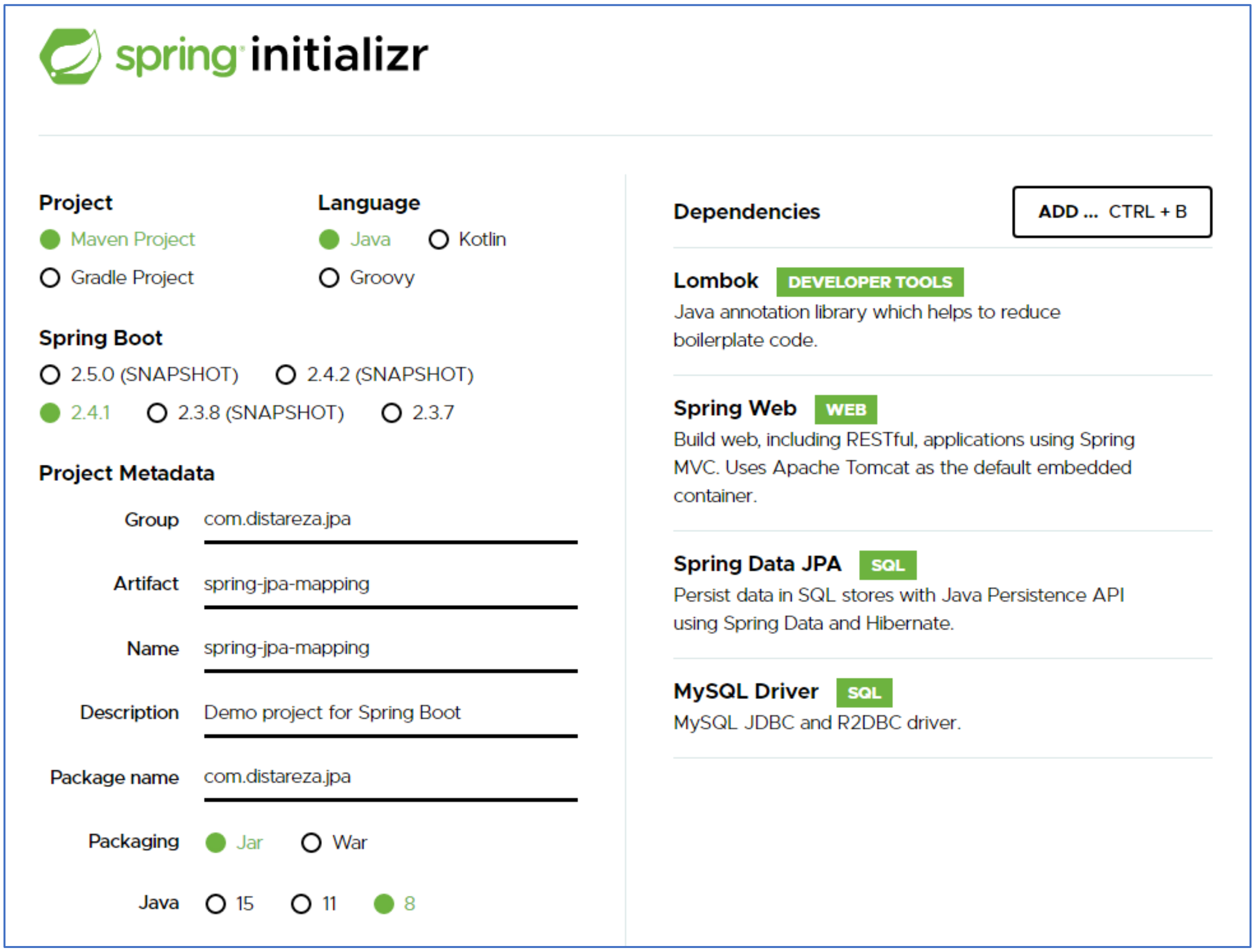
k) Start application

Request body (JSON/application) :

```
{
    "passengerInfo": {
        "name": "reza",
        "email": "distareza@gmail.com",
        "source": "KL",
        "destination": "Tokyo",
        "travelDate": "14-12-2020",
        "pickupTime": "4.0 PM",
        "arrivalTime": "6.PM",
        "fare": 18000.0
    },
    "paymentInfo": {
        "accountNo": "acc1",
        "cardType": "DEBIT"
    }
}
```

3) Spring JPA Mapping

a) Generate Spring Boot Application



spring initializr

Project

● Maven Project
○ Gradle Project

Language

● Java ○ Kotlin
○ Groovy

Spring Boot

○ 2.5.0 (SNAPSHOT) ○ 2.4.2 (SNAPSHOT)
● 2.4.1 ○ 2.3.8 (SNAPSHOT) ○ 2.3.7

Project Metadata

Group	com.distareza.jpa
Artifact	spring-jpa-mapping
Name	spring-jpa-mapping
Description	Demo project for Spring Boot
Package name	com.distareza.jpa
Packaging	● Jar ○ War
Java	○ 15 ○ 11 ● 8

Dependencies ADD ... CTRL + B

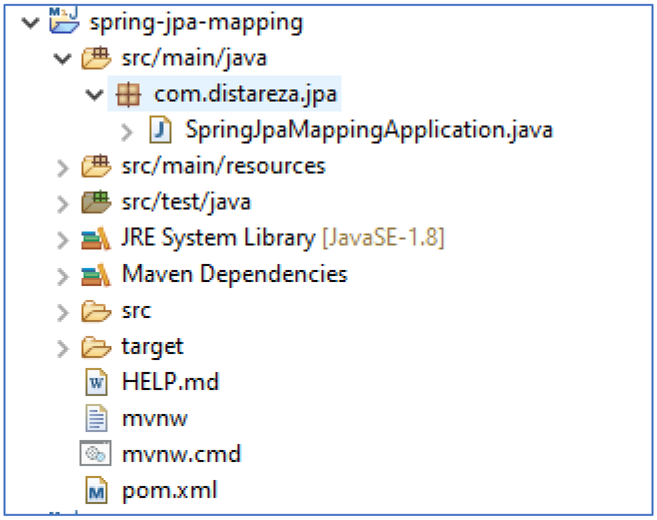
Lombok DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

MySQL Driver SQL
MySQL JDBC and R2DBC driver.

b) Import Project to Eclipse



```
spring-jpa-mapping
  src/main/java
    com.distareza.jpa
      SpringJpaMappingApplication.java
  src/main/resources
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

c) Prepare Entity Class

Customer Entity contains a customer detail information with Autogenerated Id when it created, Lombok.Data annotation will going to generate “Getter and Setter”, so no need to declare in this Class

```
package com.distareza.jpa.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

import  lombok.AllArgsConstructor;
import  lombok.Data;
import  lombok.NoArgsConstructor;
import  lombok.ToString;

@Data
@AllArgsConstructor
@NoArgsConstructor
@ToString
@Entity
public class Customer {

        @Id
        @GeneratedValue
        private int id;
        private String name;
        private String email;
        private String gender;
}
```

Product Entity contains product Information, id is not autogenerated

```
package com.distareza.jpa.entity;

import javax.persistence.Entity;
import javax.persistence.Id;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

@Data
@AllArgsConstructor
@NoArgsConstructor
@ToString
@Entity
public class Product {

        @Id
        private int pid;
        private String productName;
        private int quantity;
        private double price;

}
```

In order to make “One to Many” relation to both entity, the customer entity should declare the mapping as follows inside Customer Class

```
import java.util.List;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;

public class Customer {
...
        @OneToMany(
                        targetEntity = Product.class,
                        cascade = CascadeType.ALL,
                        fetch= FetchType.EAGER)
        @JoinColumn(
                        name = "cp_fk",
                        referencedColumnName = "id")
        private List<Product> products;

}
```

FetchType.EAGER : Fetch child entity eagerly then it will available even after the session will get close, to prevent error of org.hibernate.lazyinitializationexception: could not initialize proxy - no session .

All to-one relationships use FetchType.EAGER and all to-many relationships FetchType.LAZY (DEFAULT).
The *FetchType.EAGER* tells Hibernate to get all elements of a relationship when selecting the root entity.
The *FetchType.LAZY* tells Hibernate to only fetch the related entities from the database when you use the relationship.

d) Prepare Repository Interface

Create Corresponding Repository to each available Entity as follows

```
package com.distareza.jpa.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.distareza.jpa.entity.Customer;

public interface CustomerRepository extends JpaRepository<Customer, Integer>{

}
```

```
package com.distareza.jpa.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.distareza.jpa.entity.Customer;

public interface CustomerRepository extends JpaRepository<Customer, Integer>{

}
```

e) Enable Persistence Database in application properties

Declare the database persistence information on application properties as follows

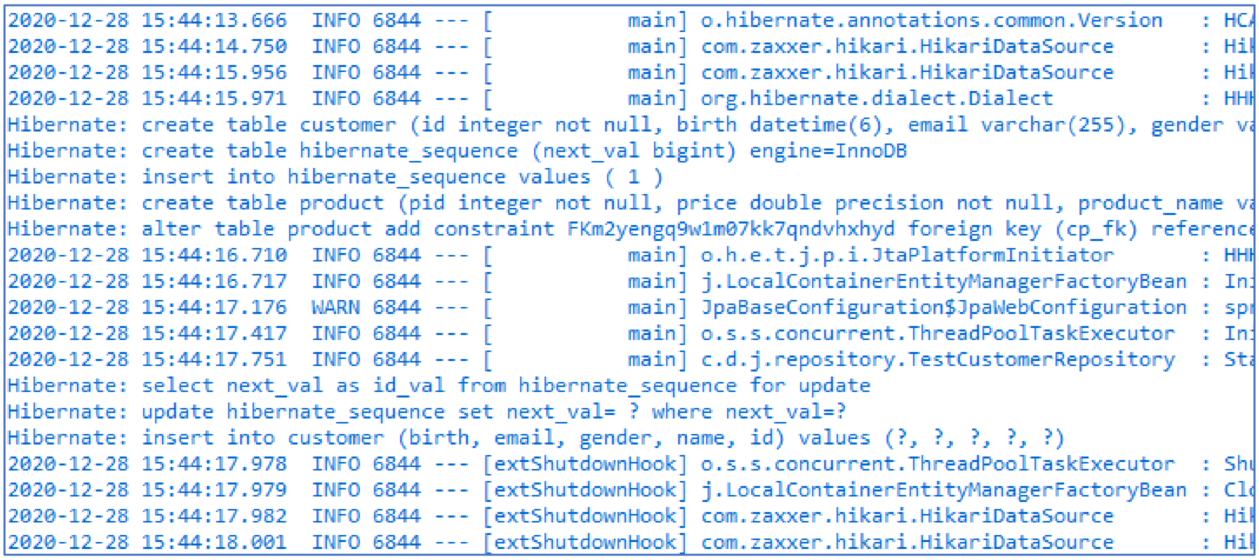
```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url = jdbc:mysql://localhost:3306/testDB
spring.datasource.username = root
spring.datasource.password = root

spring.jpa.show-sql = true
spring.jpa.hibernate.ddl-auto = update
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect

server.port = 8081
```

f) Test Repository

Create Test Class Customer Repository and Product Repository

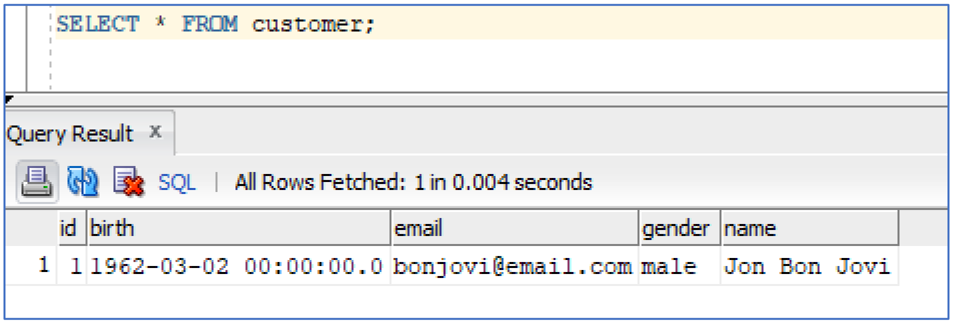
```
package com.distareza.jpa.repository;

import static org.junit.jupiter.api.Assertions.assertTrue;

import java.text.SimpleDateFormat;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import com.distareza.jpa.entity.Customer;

@SpringBootTest
public class TestCustomerRepository {

        @Autowired
        private CustomerRepository repo;

        @Test
        public void testInsertCustomer() throws Exception {
                SimpleDateFormat sdf  = new SimpleDateFormat("dd-MM-yyyy");

                Customer customer = new Customer();
                customer.setName("Jon Bon Jovi");
                customer.setEmail("bonjovi@email.com");
                customer.setBirth(sdf.parse("02-03-1962"));
                customer.setGender("male");

                repo.save(customer);

                assertTrue(true);
        }

}
```

When this test function is running, Table “Customer” and “Product” will be created if they are not yet available on Database



```
2020-12-28 15:44:13.666  INFO 6844 --- [           main] o.hibernate.annotations.common.Version   : HC
2020-12-28 15:44:14.750  INFO 6844 --- [           main] com.zaxxer.hikari.HikariDataSource       : Hi
2020-12-28 15:44:15.956  INFO 6844 --- [           main] com.zaxxer.hikari.HikariDataSource       : Hi
2020-12-28 15:44:15.971  INFO 6844 --- [           main] org.hibernate.dialect.Dialect            : HH
Hibernate: create table customer (id integer not null, birth datetime(6), email varchar(255), gender v
Hibernate: create table hibernate_sequence (next_val bigint) engine=InnoDB
Hibernate: insert into hibernate_sequence values ( 1 )
Hibernate: create table product (pid integer not null, price double precision not null, product_name v
Hibernate: alter table product add constraint FKm2yengq9w1m07kk7qndvhxhyd foreign key (cp_fk) referenc
2020-12-28 15:44:16.710  INFO 6844 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HH
2020-12-28 15:44:16.717  INFO 6844 --- [           main] j.LocalContainerEntityManagerFactoryBean : In
2020-12-28 15:44:17.176  WARN 6844 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : sp
2020-12-28 15:44:17.417  INFO 6844 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : In
2020-12-28 15:44:17.751  INFO 6844 --- [           main] c.d.j.repository.TestCustomerRepository  : St
Hibernate: select next_val as id_val from hibernate_sequence for update
Hibernate: update hibernate_sequence set next_val= ? where next_val=?
Hibernate: insert into customer (birth, email, gender, name, id) values (?, ?, ?, ?, ?)
2020-12-28 15:44:17.978  INFO 6844 --- [extShutdownHook] o.s.s.concurrent.ThreadPoolTaskExecutor  : Sh
2020-12-28 15:44:17.979  INFO 6844 --- [extShutdownHook] j.LocalContainerEntityManagerFactoryBean : Cl
2020-12-28 15:44:17.982  INFO 6844 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource       : Hi
2020-12-28 15:44:18.001  INFO 6844 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource       : Hi
```

And the result should be as follows :

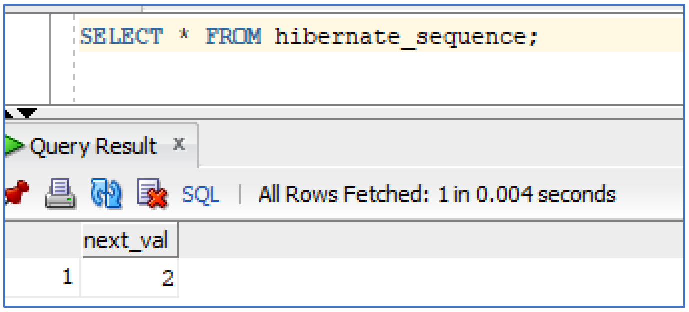


```
SELECT * FROM customer;
```

All Rows Fetched: 1 in 0.004 seconds

	id	birth	email	gender	name
1	1	1962-03-02 00:00:00.0	bonjovi@email.com	male	Jon Bon Jovi

Default Sequence is created “hibernate_sequence”



```
SELECT * FROM hibernate_sequence;
```

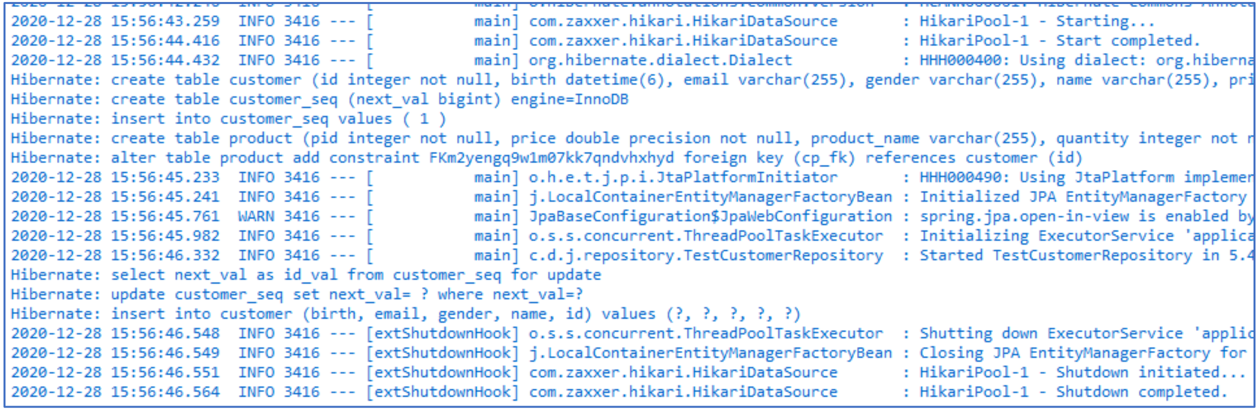
All Rows Fetched: 1 in 0.004 seconds

	next_val
1	2

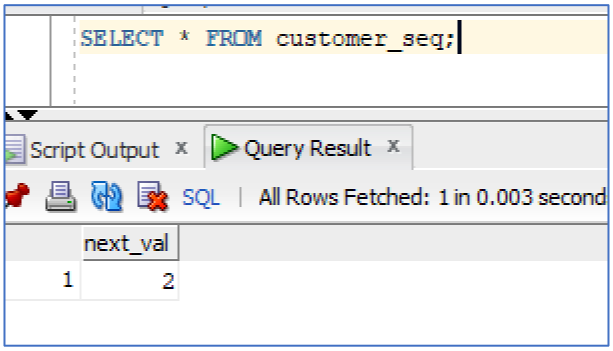
To using custom Sequence use following code :

```
@Data
@AllArgsConstructor
@NoArgsConstructor
```

```
@ToString
@Entity
public class Customer {
...
        @Id
        @GeneratedValue(strategy = GenerationType.AUTO,
                        generator = "custumer_seq")
        @SequenceGenerator(
                        name = "customer_seq",
                        sequenceName = "customer_seq",
                        initialValue = 1,
                        allocationSize = 1)
        private int id;
        private String name;
        private String email;
        private String gender;
        private Date birth;
...
}
```



```
2020-12-28 15:56:43.259  INFO 3416 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2020-12-28 15:56:44.416  INFO 3416 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2020-12-28 15:56:44.432  INFO 3416 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hiberna
Hibernate: create table customer (id integer not null, birth datetime(6), email varchar(255), gender varchar(255), name varchar(255), pri
Hibernate: create table customer_seq (next_val bigint) engine=InnoDB
Hibernate: insert into customer_seq values ( 1 )
Hibernate: create table product (pid integer not null, price double precision not null, product_name varchar(255), quantity integer not n
Hibernate: alter table product add constraint FKm2yengq9w1m07kk7qndvhxhyd foreign key (cp_fk) references customer (id)
2020-12-28 15:56:45.233  INFO 3416 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implemen
2020-12-28 15:56:45.241  INFO 3416 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory
2020-12-28 15:56:45.761  WARN 3416 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by
2020-12-28 15:56:45.982  INFO 3416 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applica
2020-12-28 15:56:46.332  INFO 3416 --- [           main] c.d.j.repository.TestCustomerRepository  : Started TestCustomerRepository in 5.4
Hibernate: select next_val as id_val from customer_seq for update
Hibernate: update customer_seq set next_val= ? where next_val=?
Hibernate: insert into customer (birth, email, gender, name, id) values (?, ?, ?, ?, ?)
2020-12-28 15:56:46.548  INFO 3416 --- [extShutdownHook] o.s.s.concurrent.ThreadPoolTaskExecutor  : Shutting down ExecutorService 'applic
2020-12-28 15:56:46.549  INFO 3416 --- [extShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for
2020-12-28 15:56:46.551  INFO 3416 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown initiated...
2020-12-28 15:56:46.564  INFO 3416 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown completed.
```

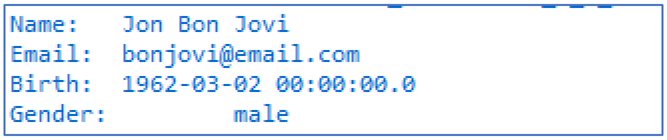


```
SELECT * FROM customer_seq;
```

	next_val
1	2

Add another test case

```
@Test
public void testSelectOneCustomerById() throws Exception {
        Optional<Customer> customer = repo.findById(1);
        if (customer.isPresent()) {

                System.out.println("Name: \t" + customer.get().getName());
                System.out.println("Email: \t" + customer.get().getEmail());
                System.out.println("Birth: \t" + customer.get().getBirth());
                System.out.println("Gender: \t" + customer.get().getGender());

        }
        assertTrue(true);
}
```

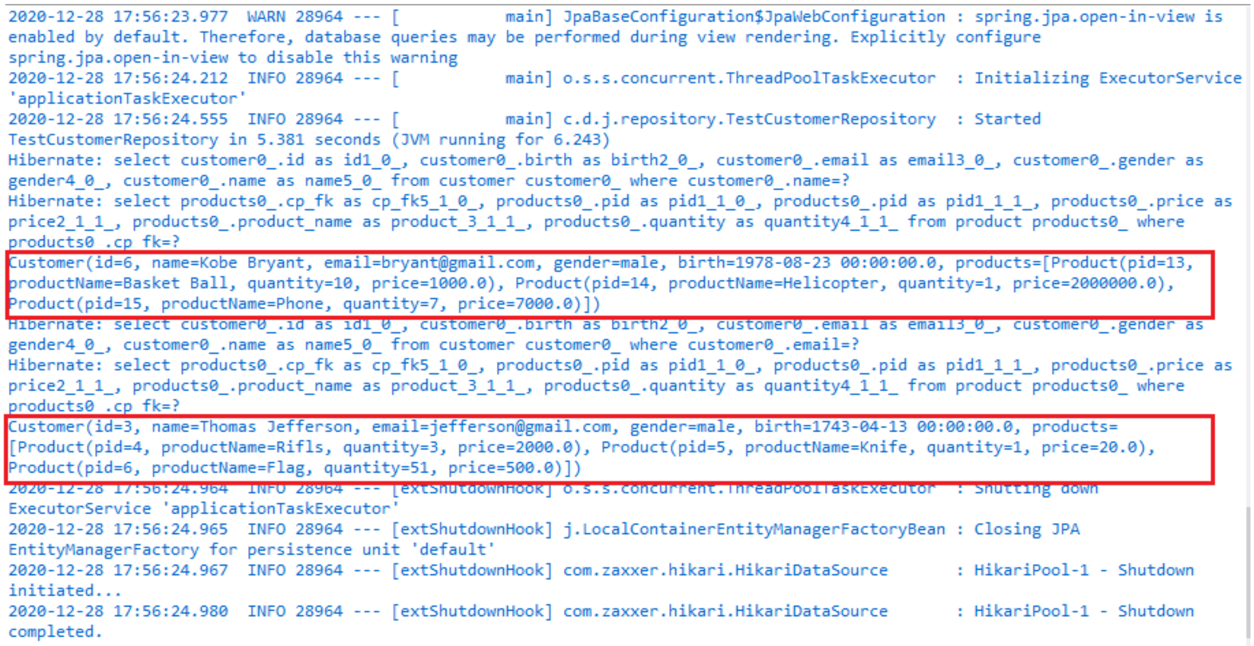


```
Name:    Jon Bon Jovi
Email:   bonjovi@email.com
Birth:   1962-03-02 00:00:00.0
Gender:          male
```

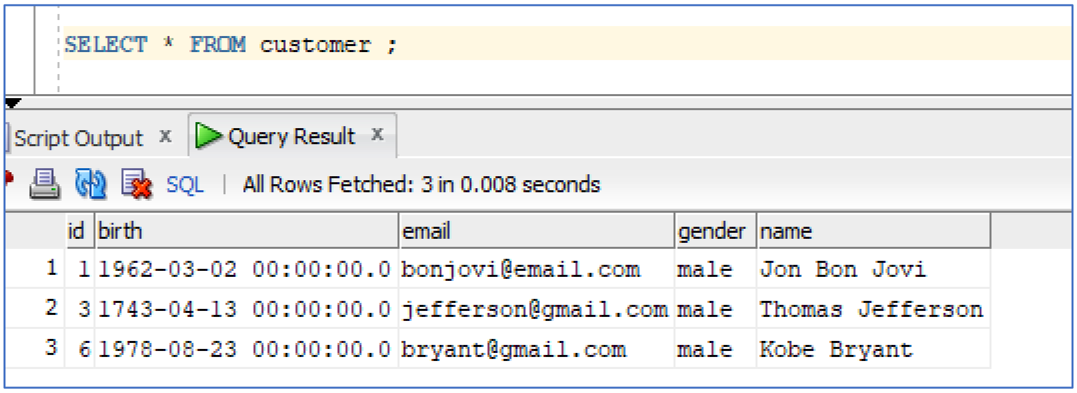
Update Customer Repository Class to add findByName and findByEmail method as follows

```java
public interface CustomerRepository extends JpaRepository<Customer, Integer>{

      Customer findByName(String user);
      Customer findByEmail(String email);

}
```

```java
@Test
public void testSelectCustomerByName() throws Exception {
      SimpleDateFormat sdf  = new SimpleDateFormat("dd-MM-yyyy");

      @SuppressWarnings("unchecked")
      List<Customer> listOfCustomer = Stream.of(
              new Customer(0, "Thomas Jefferson",
                              "jefferson@gmail.com",
                              "male",
                              sdf.parse("13-04-1743"),
              (List<Product>)(Object) Arrays.asList( new Product[]{
                      new Product(4, "Rifls", 3, 2000.0) ,
                      new Product(5, "Knife", 1, 20.0) ,
                      new Product(6, "Flag", 51, 500.0)
              })),
              new Customer(0,
                      "Kobe Bryant",
                      "bryant@gmail.com",
                      "male",
                      sdf.parse("23-08-1978"),
               (List<Product>)(Object) Arrays.asList( new Product[]{
                      new Product(13, "Basket Ball", 10, 1000.0) ,
                      new Product(14, "Helicopter", 1, 2000000.0) ,
                      new Product(15, "Phone", 7, 7000.0)
              }))
      ).collect(Collectors.toList());

      repo.saveAll(listOfCustomer);


      System.out.println(repo.findByName("Kobe Bryant"));
      System.out.println(repo.findByEmail("jefferson@gmail.com"));
      assertTrue(true);
}
```

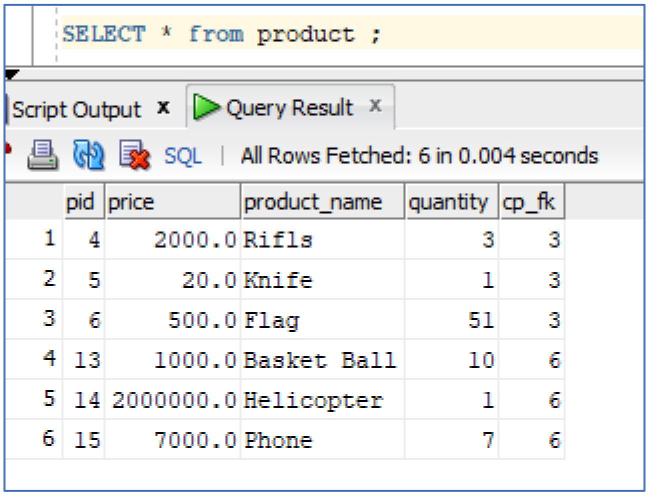


```
2020-12-28 17:56:23.977  WARN 28964 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is
enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure
spring.jpa.open-in-view to disable this warning
2020-12-28 17:56:24.212  INFO 28964 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService
'applicationTaskExecutor'
2020-12-28 17:56:24.555  INFO 28964 --- [           main] c.d.j.repository.TestCustomerRepository  : Started
TestCustomerRepository in 5.381 seconds (JVM running for 6.243)
Hibernate: select customer0_.id as id1_0_, customer0_.birth as birth2_0_, customer0_.email as email3_0_, customer0_.gender as
gender4_0_, customer0_.name as name5_0_ from customer customer0_ where customer0_.name=?
Hibernate: select products0_.cp_fk as cp_fk5_1_0_, products0_.pid as pid1_1_0_, products0_.pid as pid1_1_1_, products0_.price as
price2_1_1_, products0_.product_name as product_3_1_1_, products0_.quantity as quantity4_1_1_ from product products0_ where
products0_.cp_fk=?
Customer(id=6, name=Kobe Bryant, email=bryant@gmail.com, gender=male, birth=1978-08-23 00:00:00.0, products=[Product(pid=13,
productName=Basket Ball, quantity=10, price=1000.0), Product(pid=14, productName=Helicopter, quantity=1, price=2000000.0),
Product(pid=15, productName=Phone, quantity=7, price=7000.0)])
Hibernate: select customer0_.id as id1_0_, customer0_.birth as birth2_0_, customer0_.email as email3_0_, customer0_.gender as
gender4_0_, customer0_.name as name5_0_ from customer customer0_ where customer0_.email=?
Hibernate: select products0_.cp_fk as cp_fk5_1_0_, products0_.pid as pid1_1_0_, products0_.pid as pid1_1_1_, products0_.price as
price2_1_1_, products0_.product_name as product_3_1_1_, products0_.quantity as quantity4_1_1_ from product products0_ where
products0_.cp_fk=?
Customer(id=3, name=Thomas Jefferson, email=jefferson@gmail.com, gender=male, birth=1743-04-13 00:00:00.0, products=
[Product(pid=4, productName=Rifls, quantity=3, price=2000.0), Product(pid=5, productName=Knife, quantity=1, price=20.0),
Product(pid=6, productName=Flag, quantity=51, price=500.0)])
2020-12-28 17:56:24.964  INFO 28964 --- [extShutdownHook] o.s.s.concurrent.ThreadPoolTaskExecutor  : Shutting down
ExecutorService 'applicationTaskExecutor'
2020-12-28 17:56:24.965  INFO 28964 --- [extShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA
EntityManagerFactory for persistence unit 'default'
2020-12-28 17:56:24.967  INFO 28964 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown
initiated...
2020-12-28 17:56:24.980  INFO 28964 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown
completed.
```



```
SELECT * FROM customer ;
```

	id	birth	email	gender	name
1	1	1962-03-02 00:00:00.0	bonjovi@email.com	male	Jon Bon Jovi
2	3	1743-04-13 00:00:00.0	jefferson@gmail.com	male	Thomas Jefferson
3	6	1978-08-23 00:00:00.0	bryant@gmail.com	male	Kobe Bryant



```
SELECT * from product ;
```

	pid	price	product_name	quantity	cp_fk
1	4	2000.0	Rifls	3	3
2	5	20.0	Knife	1	3
3	6	500.0	Flag	51	3
4	13	1000.0	Basket Ball	10	6
5	14	2000000.0	Helicopter	1	6
6	15	7000.0	Phone	7	6

g) Custom Join Query

Create DTO Class

```
package com.distareza.jpa.dto;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.annotation.JsonInclude;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

@Data
@AllArgsConstructor
@NoArgsConstructor
@ToString
@JsonIgnoreProperties(ignoreUnknown = true)
@JsonInclude(JsonInclude.Include.NON_DEFAULT)
public class OrderCustomerProduct {

	private String name;
	private String productName;
	private int price;

	public OrderCustomerProduct(String name, String productName) {
		this.name = name;
		this.productName = productName;
	}
}
```

Update Customer Repository

```
public interface CustomerRepository extends JpaRepository<Customer, Integer>{

@Query("SELECT new com.distareza.jpa.dto.OrderCustomerProduct(c.name, p.productName) " +
      "from Customer c JOIN c.products p " +
      "WHERE c.email = ?1")
public List<OrderCustomerProduct> getJoinInformation(String email);

}
```

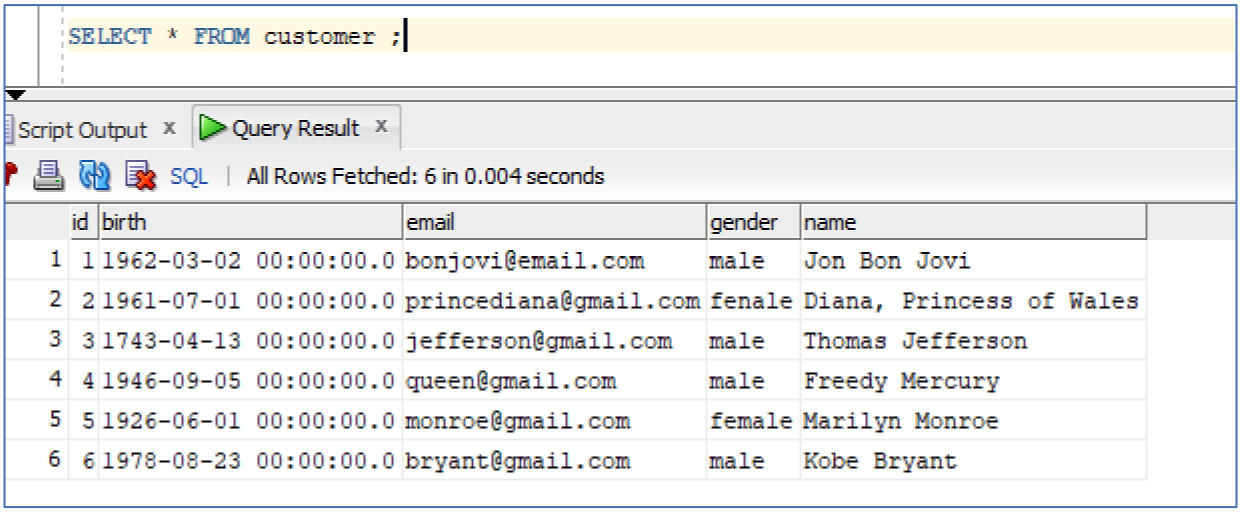
Or update as follows

```
public interface CustomerRepository extends JpaRepository<Customer, Integer>{

@Query("SELECT new com.distareza.jpa.dto.OrderCustomerProduct(c.name, p.productName) " +
      "from Customer c JOIN c.products p " +
      "WHERE c.email = :email ")
public List<OrderCustomerProduct> getJoinInformation(@Param("email") String email);
}
```

Test Class

```
@Test
public void testQueryJoin() throws Exception {

      List<OrderCustomerProduct> list = repo.getJoinInformation("queen@gmail.com");
      ObjectMapper objectMapper = new ObjectMapper();
      String jsonText = objectMapper
              .writerWithDefaultPrettyPrinter().writeValueAsString(list);

      System.out.println(jsonText);
```

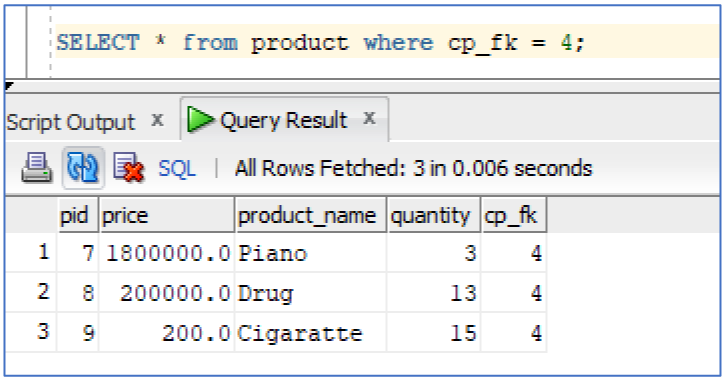
```java
        assertTrue(true);
    }
```



```sql
SELECT * FROM customer ;
```

All Rows Fetched: 6 in 0.004 seconds

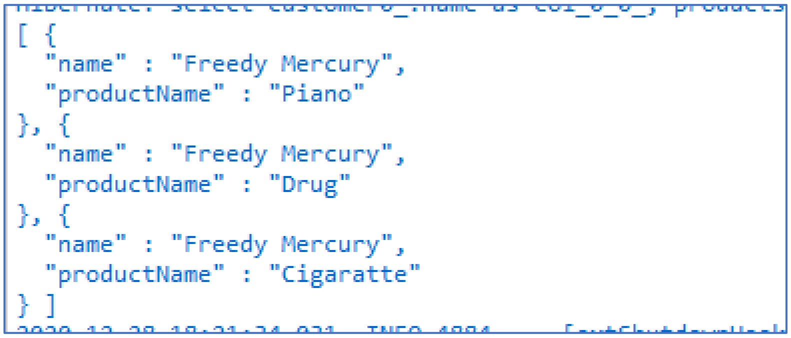
	id	birth	email	gender	name
1	1	1962-03-02 00:00:00.0	bonjovi@email.com	male	Jon Bon Jovi
2	2	1961-07-01 00:00:00.0	princediana@gmail.com	fenale	Diana, Princess of Wales
3	3	1743-04-13 00:00:00.0	jefferson@gmail.com	male	Thomas Jefferson
4	4	1946-09-05 00:00:00.0	queen@gmail.com	male	Freedy Mercury
5	5	1926-06-01 00:00:00.0	monroe@gmail.com	female	Marilyn Monroe
6	6	1978-08-23 00:00:00.0	bryant@gmail.com	male	Kobe Bryant



```sql
SELECT * from product where cp_fk = 4;
```

All Rows Fetched: 3 in 0.006 seconds

	pid	price	product_name	quantity	cp_fk
1	7	1800000.0	Piano	3	4
2	8	200000.0	Drug	13	4
3	9	200.0	Cigaratte	15	4



```
Hibernate: select customer0_.name as col_0_0_, products
[ {
  "name" : "Freedy Mercury",
  "productName" : "Piano"
}, {
  "name" : "Freedy Mercury",
  "productName" : "Drug"
}, {
  "name" : "Freedy Mercury",
  "productName" : "Cigaratte"
} ]
2020-12-28 18:21:34.031  INFO 4884 --- [extShutdownHook
```