* **Explain the need and benefit of ORM**
  + **ORM (Object-Relational Mapping), makes it easier to develop code that interacts with database, abstracts the database system, transactionality**
    - **ORM Pros and Cons - https://blog.bitsrc.io/what-is-an-orm-and-why-you-should-use-it-b2b6f75f5e2a**
    - **What is ORM? - https://en.wikipedia.org/wiki/Object-relational\_mapping**

**What is ORM?**

**Object-Relational Mapping (ORM)** is a programming technique that helps developers interact with databases using **objects** instead of writing raw SQL queries. It bridges the gap between **object-oriented languages like Java** and **relational databases like MySQL**

**Need for ORM**

**1. Impedance Mismatch Between OOP and RDBMS**

* Java uses **objects, classes, inheritance**, etc.
* Databases use **tables, rows, columns, primary/foreign keys**.
* This mismatch makes direct interaction difficult.
* ORM **maps classes to tables**, attributes to columns.

**2. Reduces Boilerplate Code**

Without ORM:

java

CopyEdit

Connection con = DriverManager.getConnection(...);

PreparedStatement stmt = con.prepareStatement("SELECT \* FROM country WHERE id=?");

With ORM (e.g., JPA):

java

CopyEdit

countryRepository.findById("IN");

**3. Improves Productivity**

* You focus on business logic.
* No need to worry about SQL, JDBC, result sets, etc.

**4. Handles Relationships Easily**

* Supports 1-to-1, 1-to-many, many-to-many using annotations or mapping files.

**5. Transaction Management**

* ORMs like Hibernate manage transaction boundaries using annotations like @Transactional.

**6. Database Independence**

* Change dialects easily (MySQL ➝ PostgreSQL) without rewriting SQL.

* **Demonstrate the need and benefit of Spring Data JPA**
  + **Evolution of ORM solutions, Hibernate XML Configuration, Hibernate Annotation Configuration, Spring Data JPA, Hibernate benefits, open source, light weight, database independent query**
    - **With H2 in memory database - https://www.mkyong.com/spring-boot/spring-boot-spring-data-jpa/**
    - **With MySQL - https://www.mkyong.com/spring-boot/spring-boot-spring-data-jpa-mysql-example/**
    - **XML Configuration Example -https://www.tutorialspoint.com/hibernate/hibernate\_examples.htm**
    - **Hibernate Configuration Example -https://www.tutorialspoint.com/hibernate/hibernate\_annotations.htm**
* Spring Data JPA is a powerful and high-level abstraction built on top of the Java Persistence API (JPA) that simplifies database operations and significantly reduces boilerplate code in Java applications. Traditionally, developers had to write extensive and repetitive code using JDBC to interact with databases, including managing connections, preparing SQL statements, handling transactions, and mapping results. This process was error-prone and hard to maintain. To address this, ORM tools like Hibernate were introduced. Initially, Hibernate used XML-based configuration files (.hbm.xml) to define how Java classes mapped to database tables. Although functional, this method was verbose and difficult to manage in large applications. Over time, Hibernate evolved to support annotations such as @Entity, @Table, and @Id, making the configuration much cleaner and more intuitive by placing it directly within the Java classes.
* The next step in this evolution was the introduction of JPA, which is a Java specification (JSR 338) that defines how to manage relational data using Java objects. However, JPA is only a specification and lacks a concrete implementation, so tools like Hibernate serve as JPA providers. While JPA improved consistency and portability across different ORM tools, developers still had to write data access logic manually. This is where Spring Data JPA provides a significant leap in productivity. It builds on top of JPA and abstracts the entire data access layer, allowing developers to perform CRUD operations without writing any SQL or implementing repository methods. By simply extending interfaces like JpaRepository or CrudRepository, Spring Data JPA automatically provides ready-to-use methods like findAll(), findById(), save(), and deleteById().
* One of the key benefits of Spring Data JPA is its support for query method naming conventions, where methods like findByName() automatically translate into SQL queries, making the code expressive and intuitive. It also provides built-in support for sorting and pagination, manages transactions automatically via annotations like @Transactional, and ensures compatibility with various databases by allowing developers to simply switch the database dialect in configuration. Because of its integration with Spring Boot, Spring Data JPA is lightweight, open source, and easy to integrate in modern applications.
* For testing or lightweight use cases, developers can use an H2 in-memory database, which requires no installation and is perfect for unit testing or prototyping. For real-world scenarios, Spring Data JPA works seamlessly with relational databases like MySQL by setting the appropriate JDBC URL and dialect in the application.properties file. The configuration is minimal and easy to manage. Spring Data JPA’s design allows developers to focus entirely on business logic while the framework handles all underlying data access, transactions, and object-relational mapping concerns. Comprehensive tutorials and examples on implementing Spring Data JPA with H2 or MySQL databases can be found on [Mkyong’s blog for H2](https://www.mkyong.com/spring-boot/spring-boot-spring-data-jpa/) and [for MySQL](https://www.mkyong.com/spring-boot/spring-boot-spring-data-jpa-mysql-example/). Additionally, more foundational Hibernate configuration examples using XML and annotations are available at [TutorialsPoint XML](https://www.tutorialspoint.com/hibernate/hibernate_examples.htm) and [Annotations guide](https://www.tutorialspoint.com/hibernate/hibernate_annotations.htm).

* **Explain about core objects of hibernate framework**
  + **Session Factory, Session, Transaction Factory, Transaction, Connection Provider**
    - **Hibernate Architecture Reference - https://www.tutorialspoint.com/hibernate/hibernate\_architecture.htm**

 The Hibernate framework is a powerful and widely used ORM (Object-Relational Mapping) tool in Java that allows seamless interaction between Java applications and relational databases. At the heart of Hibernate are several **core objects** that enable it to manage database operations efficiently and consistently. One of the most fundamental components is the **SessionFactory**. It is a heavyweight object that is created once during application startup and is responsible for producing **Session** objects. The SessionFactory is configured using the Hibernate configuration file (usually hibernate.cfg.xml) and contains database connection information, mappings, dialect settings, and other global properties. It is designed to be thread-safe and shared across the entire application.

The **Session** object is the primary interface used by developers to perform CRUD (Create, Read, Update, Delete) operations. It represents a single unit of work with the database and acts as a bridge between the Java application and the actual database. Each Session is a lightweight and non-thread-safe object that opens a temporary connection with the database. It provides methods like save(), update(), get(), delete(), and createQuery() to interact with the database in an object-oriented way. It also caches objects within a session scope, enabling performance optimizations through first-level caching.

Another key component is the **Transaction** object, which is used to manage the boundaries of database transactions. Transactions ensure that a group of operations are executed atomically—either all succeed or all fail. The Transaction object in Hibernate abstracts away the underlying JDBC transaction management and provides a clean API for beginning, committing, or rolling back transactions using methods like beginTransaction(), commit(), and rollback().

Supporting these components is the **TransactionFactory**, which is responsible for creating Transaction instances. While it is used internally by Hibernate, it plays a crucial role in decoupling the transaction implementation from the core framework, allowing Hibernate to support multiple transaction strategies, including JDBC and JTA (Java Transaction API). Similarly, the **ConnectionProvider** is another behind-the-scenes component that abstracts the management of physical database connections. It handles connection pooling, creation, and release of connections, allowing Hibernate to integrate with different connection pool providers like HikariCP, C3P0, or DBCP.

* **Explain ORM implementation with Hibernate XML Configuration and Annotation Configuration**
  + **XML Configuration - persistence class, mapping xml, configuration xml, loading hibernate configuration xml file; Annotation Configuration - persistence class, @Entity, @Table, @Id, @Column, hibernate configuration xml file Loading hibernate configuration and interacting with database get the session factory, open session, begin transaction, commit transaction, close session**
    - **XML Configuration Example - https://www.tutorialspoint.com/hibernate/hibernate\_examples.htm**
    - **Hibernate Configuration Example -** [**https://www.tutorialspoint.com/hibernate/hibernate\_annotations.htm**](https://www.tutorialspoint.com/hibernate/hibernate_annotations.htm)
* Hibernate supports two primary ways of implementing ORM: **XML configuration** and **annotation configuration**. In the XML-based approach, mappings between Java classes and database tables are defined externally in .hbm.xml files, while database connection details and mapping file references are placed in hibernate.cfg.xml. The application reads these configurations to create a SessionFactory, from which sessions are opened to interact with the database using methods like save(), get(), and delete(). This approach promotes separation of concerns but can be verbose.
* The **annotation-based configuration** simplifies this by embedding mapping metadata directly in the Java class using annotations like @Entity, @Table, @Id, and @Column. This makes the code self-contained and easier to maintain. The hibernate.cfg.xml is still used for DB connection settings, but no separate mapping XML files are needed. In both approaches, database interaction follows the same lifecycle: configure, open session, begin transaction, perform operations, commit, and close session. For detailed examples, refer to [Hibernate XML Example](https://www.tutorialspoint.com/hibernate/hibernate_examples.htm) and [Annotation Example](https://www.tutorialspoint.com/hibernate/hibernate_annotations.htm).

* **Explain the difference between Java Persistence API, Hibernate and Spring Data JPA**
  + **JPA (Java Persistence API), JPA is a specification (JSR 338), JPA does not have implementation, Hibernate is one of the implementation for JPA, Hibernate is a ORM tool, Spring Data JPA is an abstraction above Hibernate to remove boiler plate code when persisting data using Hibernate.**
    - **Difference between Spring Data JPA and Hibernate - https://dzone.com/articles/what-is-the-difference-between-hibernate-and-sprin-1**
    - **Intro to JPA -** [**https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html**](https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html)
* The **Java Persistence API (JPA)** is a **specification** defined under JSR 338 that standardizes the way Java applications interact with relational databases. It provides a set of interfaces and annotations to manage object-relational mapping and data persistence but does **not include any actual implementation**. **Hibernate** is one of the most popular **implementations of the JPA specification**, and it also offers additional features beyond what JPA defines, such as caching and custom query mechanisms. Hibernate is a full-fledged **ORM (Object-Relational Mapping) tool** that handles database connections, mappings, and transactions using either XML or annotations.
* On top of this, **Spring Data JPA** is a **high-level abstraction** built over JPA and its implementations like Hibernate. It reduces boilerplate code significantly by providing out-of-the-box support for standard CRUD operations, pagination, sorting, and custom query derivation through method names (e.g., findByName()). With Spring Data JPA, developers only need to define repository interfaces, and the framework generates the required implementations at runtime. In essence, while JPA defines the rules, Hibernate follows and implements them, and Spring Data JPA simplifies their usage, making data access much easier and faster to develop.

* **Demonstrate implementation of DML using Spring Data JPA on a single database table**
  + **Hibernate log configuration and ddl-auto configuration, JpaRepsitory.findById(), defining Query Methods, JpaRespository.save(), JpaRepository.deleteById()**
    - **Spring Data JPA Ref Repository methods - https://docs.spring.io/spring-data/jpa/docs/2.2.0.RELEASE/reference/html/#repositories.core-concepts**
    - **Query methods -** [**https://docs.spring.io/spring-data/jpa/docs/2.2.0.RELEASE/reference/html/#repositories.query-methods**](https://docs.spring.io/spring-data/jpa/docs/2.2.0.RELEASE/reference/html/#repositories.query-methods)
* Spring Data JPA makes it easy to perform DML operations like **inserting, updating, retrieving, and deleting records** on a database table by using built-in repository methods. To begin with, Hibernate behavior for schema management can be controlled using the spring.jpa.hibernate.ddl-auto property in the application.properties file. Common values include create, update, validate, and create-drop. To view the SQL queries being executed, developers can enable detailed Hibernate logs using properties like logging.level.org.hibernate.SQL=trace.
* For actual database interaction, a repository interface is created by extending JpaRepository, which provides ready-to-use methods such as findById(), save(), and deleteById(). For example, findById(String id) retrieves a record by its primary key, save(entity) is used for both inserting and updating records, and deleteById(id) removes a record. Additionally, developers can define custom query methods by following Spring Data JPA’s method naming conventions—like findByName(String name)—which automatically generates the appropriate query behind the scenes.

**Hands on 1**

**Spring Data JPA - Quick Example**   
  
**Software Pre-requisites**

* MySQL Server 8.0
* MySQL Workbench 8
* Eclipse IDE for Enterprise Java Developers 2019-03 R
* Maven 3.6.2

**Create a Eclipse Project using Spring Initializr**

* Go to <https://start.spring.io/>
* Change Group as “com.cognizant”
* Change Artifact Id as “orm-learn”
* In Options > Description enter "Demo project for Spring Data JPA and Hibernate"
* Click on menu and select "Spring Boot DevTools", "Spring Data JPA" and "MySQL Driver"
* Click Generate and download the project as zip
* Extract the zip in root folder to Eclipse Workspace
* Import the project in Eclipse "File > Import > Maven > Existing Maven Projects > Click Browse and select extracted folder > Finish"
* Create a new schema "ormlearn" in MySQL database. Execute the following commands to open MySQL client and create schema.

> mysql -u root -p

mysql> create schema ormlearn;

* In orm-learn Eclipse project, open src/main/resources/application.properties and include the below database and log configuration.

# Spring Framework and application log

logging.level.org.springframework=info

logging.level.com.cognizant=debug

# Hibernate logs for displaying executed SQL, input and output

logging.level.org.hibernate.SQL=trace

logging.level.org.hibernate.type.descriptor.sql=trace

# Log pattern

logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread %5p %-25.25logger{25} %25M %4L %m%n

# Database configuration

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn

spring.datasource.username=root

spring.datasource.password=root

# Hibernate configuration

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

* Build the project using ‘mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456’ command in command line
* Include logs for verifying if main() method is called.

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);

public static void main(String[] args) {

SpringApplication.run(OrmLearnApplication.class, args);

  LOGGER.info("Inside main");

}

* Execute the OrmLearnApplication and check in log if main method is called.

SME to walk through the following aspects related to the project created:

1. src/main/java - Folder with application code
2. src/main/resources - Folder for application configuration
3. src/test/java - Folder with code for testing the application
4. OrmLearnApplication.java - Walkthrough the main() method.
5. Purpose of @SpringBootApplication annotation
6. pom.xml
   1. Walkthrough all the configuration defined in XML file
   2. Open 'Dependency Hierarchy' and show the dependency tree.

**Country table creation**

* Create a new table country with columns for code and name. For sample, let us insert one country with values 'IN' and 'India' in this table.

create table country(co\_code varchar(2) primary key, co\_name varchar(50));

* Insert couple of records into the table

insert into country values ('IN', 'India');

insert into country values ('US', 'United States of America');

**Persistence Class - com.cognizant.orm-learn.model.Country**

* Open Eclipse with orm-learn project
* Create new package com.cognizant.orm-learn.model
* Create Country.java, then generate getters, setters and toString() methods.
* Include @Entity and @Table at class level
* Include @Column annotations in each getter method specifying the column name.

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name="country")

public class Country {

  @Id

    @Column(name="code")

    private String code;

    @Column(name="name")

    private String name;

// getters and setters

  // toString()

}

*Notes:*

* @Entity is an indicator to Spring Data JPA that it is an entity class for the application
* @Table helps in defining the mapping database table
* @Id helps is defining the primary key
* @Column helps in defining the mapping table column

**Repository Class - com.cognizant.orm-learn.CountryRepository**

* Create new package com.cognizant.orm-learn.repository
* Create new interface named CountryRepository that extends JpaRepository<Country, String>
* Define @Repository annotation at class level

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

import org.springframework.stereotype.Repository;

import com.cognizant.ormlearn.model.Country;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

**Service Class - com.cognizant.orm-learn.service.CountryService**

* Create new package com.cognizant.orm-learn.service
* Create new class CountryService
* Include @Service annotation at class level
* Autowire CountryRepository in CountryService
* Include new method getAllCountries() method that returns a list of countries.
* Include @Transactional annotation for this method
* In getAllCountries() method invoke countryRepository.findAll() method and return the result

**Testing in OrmLearnApplication.java**

* Include a static reference to CountryService in OrmLearnApplication class

private static CountryService countryService;

* Define a test method to get all countries from service.

    private static void testGetAllCountries() {

        LOGGER.info("Start");

        List<Country> countries = countryService.getAllCountries();

        LOGGER.debug("countries={}", countries);

        LOGGER.info("End");

    }

* Modify SpringApplication.run() invocation to set the application context and the CountryService reference from the application context.

        ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);

        countryService = context.getBean(CountryService.class);

        testGetAllCountries();

* Execute main method to check if data from ormlearn database is retrieved.

**Hands on 2**

**Hibernate XML Config implementation walk through**   
  
SME to provide explanation on the sample Hibernate implementation available in the link below:  
https://www.tutorialspoint.com/hibernate/hibernate\_examples.htm  
  
Explanation Topics

* Explain how object to relational database mapping done in hibernate xml configuration file
* Explain about following aspects of implementing the end to end operations in Hibernate:
  + SessionFactory
  + Session
  + Transaction
  + beginTransaction()
  + commit()
  + rollback()
  + session.save()
  + session.createQuery().list()
  + session.get()
  + session.delete()

**Hands on 3**

**Hibernate Annotation Config implementation walk through**   
  
SME to provide explanation on the sample Hibernate implementation available in the link below:  
https://www.tutorialspoint.com/hibernate/hibernate\_annotations.htm  
  
Explanation Topics

* Explain how object to relational database mapping done in persistence class file Employee
* Explain about following aspects of implementing the end to end operations in Hibernate:
  + @Entity
  + @Table
  + @Id
  + @GeneratedValue
  + @Column
  + Hibernate Configuration (hibernate.cfg.xml)
    - Dialect
    - Driver
    - Connection URL
    - Username
    - Password

**Hands on 4**

**Difference between JPA, Hibernate and Spring Data JPA**   
  
Java Persistence API (JPA)

* JSR 338 Specification for persisting, reading and managing data from Java objects
* Does not contain concrete implementation of the specification
* Hibernate is one of the implementation of JPA

Hibernate

* ORM Tool that implements JPA

Spring Data JPA

* Does not have JPA implementation, but reduces boiler plate code
* This is another level of abstraction over JPA implementation provider like Hibernate
* Manages transactions

**Refer code snippets below on how the code compares between Hibernate and Spring Data JPA  
Hibernate**

   /\* Method to CREATE an employee in the database \*/

   public Integer addEmployee(Employee employee){

      Session session = factory.openSession();

      Transaction tx = null;

      Integer employeeID = null;

      try {

         tx = session.beginTransaction();

         employeeID = (Integer) session.save(employee);

         tx.commit();

      } catch (HibernateException e) {

         if (tx != null) tx.rollback();

         e.printStackTrace();

      } finally {

         session.close();

      }

      return employeeID;

   }

**Spring Data JPA**  
EmployeeRespository.java

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

EmployeeService.java

@Autowire

  private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

  employeeRepository.save(employee);

  }

​​​​​​​   
  
**Reference Links:**   
<https://dzone.com/articles/what-is-the-difference-between-hibernate-and-sprin-1>   
<https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html>

**Hands on 5**

**Implement services for managing Country**   
  
An application requires for features to be implemented with regards to country. These features needs to be supported by implementing them as service using Spring Data JPA.

* Find a country based on country code
* Add new country
* Update country
* Delete country
* Find list of countries matching a partial country name

Before starting the implementation of the above features, there are few configuration and data population that needs to be incorporated. Please refer each topic below and implement the same.   
  
**Explanation for Hibernate table creation configuration**

* Moreover the ddl-auto defines how hibernate behaves if a specific table or column is not present in the database.
  + create - drops existing tables data and structure, then creates new tables
  + validate - check if the table and columns exist or not, throws an exception if a matching table or column is not found
  + update - if a table does not exists, it creates a new table; if a column does not exists, it creates a new column
  + create-drop - creates the table, once all operations are completed, the table is dropped

# Hibernate ddl auto (create, create-drop, update, validate)

spring.jpa.hibernate.ddl-auto=validate

Populate country table

* Delete all the records in Country table and then use the below script to create the actual list of all countries in our world.

insert into country (co\_code, co\_name) values ("AF", "Afghanistan");

insert into country (co\_code, co\_name) values ("AL", "Albania");

insert into country (co\_code, co\_name) values ("DZ", "Algeria");

insert into country (co\_code, co\_name) values ("AS", "American Samoa");

insert into country (co\_code, co\_name) values ("AD", "Andorra");

insert into country (co\_code, co\_name) values ("AO", "Angola");

insert into country (co\_code, co\_name) values ("AI", "Anguilla");

insert into country (co\_code, co\_name) values ("AQ", "Antarctica");

insert into country (co\_code, co\_name) values ("AG", "Antigua and Barbuda");

insert into country (co\_code, co\_name) values ("AR", "Argentina");

insert into country (co\_code, co\_name) values ("AM", "Armenia");

insert into country (co\_code, co\_name) values ("AW", "Aruba");

insert into country (co\_code, co\_name) values ("AU", "Australia");

insert into country (co\_code, co\_name) values ("AT", "Austria");

insert into country (co\_code, co\_name) values ("AZ", "Azerbaijan");

insert into country (co\_code, co\_name) values ("BS", "Bahamas");

insert into country (co\_code, co\_name) values ("BH", "Bahrain");

insert into country (co\_code, co\_name) values ("BD", "Bangladesh");

insert into country (co\_code, co\_name) values ("BB", "Barbados");

insert into country (co\_code, co\_name) values ("BY", "Belarus");

insert into country (co\_code, co\_name) values ("BE", "Belgium");

insert into country (co\_code, co\_name) values ("BZ", "Belize");

insert into country (co\_code, co\_name) values ("BJ", "Benin");

insert into country (co\_code, co\_name) values ("BM", "Bermuda");

insert into country (co\_code, co\_name) values ("BT", "Bhutan");

insert into country (co\_code, co\_name) values ("BO", "Bolivia, Plurinational State of");

insert into country (co\_code, co\_name) values ("BQ", "Bonaire, Sint Eustatius and Saba");

insert into country (co\_code, co\_name) values ("BA", "Bosnia and Herzegovina");

insert into country (co\_code, co\_name) values ("BW", "Botswana");

insert into country (co\_code, co\_name) values ("BV", "Bouvet Island");

insert into country (co\_code, co\_name) values ("BR", "Brazil");

insert into country (co\_code, co\_name) values ("IO", "British Indian Ocean Territory");

insert into country (co\_code, co\_name) values ("BN", "Brunei Darussalam");

insert into country (co\_code, co\_name) values ("BG", "Bulgaria");

insert into country (co\_code, co\_name) values ("BF", "Burkina Faso");

insert into country (co\_code, co\_name) values ("BI", "Burundi");

insert into country (co\_code, co\_name) values ("KH", "Cambodia");

insert into country (co\_code, co\_name) values ("CM", "Cameroon");

insert into country (co\_code, co\_name) values ("CA", "Canada");

insert into country (co\_code, co\_name) values ("CV", "Cape Verde");

insert into country (co\_code, co\_name) values ("KY", "Cayman Islands");

insert into country (co\_code, co\_name) values ("CF", "Central African Republic");

insert into country (co\_code, co\_name) values ("TD", "Chad");

insert into country (co\_code, co\_name) values ("CL", "Chile");

insert into country (co\_code, co\_name) values ("CN", "China");

insert into country (co\_code, co\_name) values ("CX", "Christmas Island");

insert into country (co\_code, co\_name) values ("CC", "Cocos (Keeling) Islands");

insert into country (co\_code, co\_name) values ("CO", "Colombia");

insert into country (co\_code, co\_name) values ("KM", "Comoros");

insert into country (co\_code, co\_name) values ("CG", "Congo");

insert into country (co\_code, co\_name) values ("CD", "Congo, the Democratic Republic of the");

insert into country (co\_code, co\_name) values ("CK", "Cook Islands");

insert into country (co\_code, co\_name) values ("CR", "Costa Rica");

insert into country (co\_code, co\_name) values ("HR", "Croatia");

insert into country (co\_code, co\_name) values ("CU", "Cuba");

insert into country (co\_code, co\_name) values ("CW", "Curaçao");

insert into country (co\_code, co\_name) values ("CY", "Cyprus");

insert into country (co\_code, co\_name) values ("CZ", "Czech Republic");

insert into country (co\_code, co\_name) values ("CI", "Côte d'Ivoire");

insert into country (co\_code, co\_name) values ("DK", "Denmark");

insert into country (co\_code, co\_name) values ("DJ", "Djibouti");

insert into country (co\_code, co\_name) values ("DM", "Dominica");

insert into country (co\_code, co\_name) values ("DO", "Dominican Republic");

insert into country (co\_code, co\_name) values ("EC", "Ecuador");

insert into country (co\_code, co\_name) values ("EG", "Egypt");

insert into country (co\_code, co\_name) values ("SV", "El Salvador");

insert into country (co\_code, co\_name) values ("GQ", "Equatorial Guinea");

insert into country (co\_code, co\_name) values ("ER", "Eritrea");

insert into country (co\_code, co\_name) values ("EE", "Estonia");

insert into country (co\_code, co\_name) values ("ET", "Ethiopia");

insert into country (co\_code, co\_name) values ("FK", "Falkland Islands (Malvinas)");

insert into country (co\_code, co\_name) values ("FO", "Faroe Islands");

insert into country (co\_code, co\_name) values ("FJ", "Fiji");

insert into country (co\_code, co\_name) values ("FI", "Finland");

insert into country (co\_code, co\_name) values ("FR", "France");

insert into country (co\_code, co\_name) values ("GF", "French Guiana");

insert into country (co\_code, co\_name) values ("PF", "French Polynesia");

insert into country (co\_code, co\_name) values ("TF", "French Southern Territories");

insert into country (co\_code, co\_name) values ("GA", "Gabon");

insert into country (co\_code, co\_name) values ("GM", "Gambia");

insert into country (co\_code, co\_name) values ("GE", "Georgia");

insert into country (co\_code, co\_name) values ("DE", "Germany");

insert into country (co\_code, co\_name) values ("GH", "Ghana");

insert into country (co\_code, co\_name) values ("GI", "Gibraltar");

insert into country (co\_code, co\_name) values ("GR", "Greece");

insert into country (co\_code, co\_name) values ("GL", "Greenland");

insert into country (co\_code, co\_name) values ("GD", "Grenada");

insert into country (co\_code, co\_name) values ("GP", "Guadeloupe");

insert into country (co\_code, co\_name) values ("GU", "Guam");

insert into country (co\_code, co\_name) values ("GT", "Guatemala");

insert into country (co\_code, co\_name) values ("GG", "Guernsey");

insert into country (co\_code, co\_name) values ("GN", "Guinea");

insert into country (co\_code, co\_name) values ("GW", "Guinea-Bissau");

insert into country (co\_code, co\_name) values ("GY", "Guyana");

insert into country (co\_code, co\_name) values ("HT", "Haiti");

insert into country (co\_code, co\_name) values ("HM", "Heard Island and McDonald Islands");

insert into country (co\_code, co\_name) values ("VA", "Holy See (Vatican City State)");

insert into country (co\_code, co\_name) values ("HN", "Honduras");

insert into country (co\_code, co\_name) values ("HK", "Hong Kong");

insert into country (co\_code, co\_name) values ("HU", "Hungary");

insert into country (co\_code, co\_name) values ("IS", "Iceland");

insert into country (co\_code, co\_name) values ("IN", "India");

insert into country (co\_code, co\_name) values ("ID", "Indonesia");

insert into country (co\_code, co\_name) values ("IR", "Iran, Islamic Republic of");

insert into country (co\_code, co\_name) values ("IQ", "Iraq");

insert into country (co\_code, co\_name) values ("IE", "Ireland");

insert into country (co\_code, co\_name) values ("IM", "Isle of Man");

insert into country (co\_code, co\_name) values ("IL", "Israel");

insert into country (co\_code, co\_name) values ("IT", "Italy");

insert into country (co\_code, co\_name) values ("JM", "Jamaica");

insert into country (co\_code, co\_name) values ("JP", "Japan");

insert into country (co\_code, co\_name) values ("JE", "Jersey");

insert into country (co\_code, co\_name) values ("JO", "Jordan");

insert into country (co\_code, co\_name) values ("KZ", "Kazakhstan");

insert into country (co\_code, co\_name) values ("KE", "Kenya");

insert into country (co\_code, co\_name) values ("KI", "Kiribati");

insert into country (co\_code, co\_name) values ("KP", "Democratic People's Republic of Korea");

insert into country (co\_code, co\_name) values ("KR", "Republic of Korea");

insert into country (co\_code, co\_name) values ("KW", "Kuwait");

insert into country (co\_code, co\_name) values ("KG", "Kyrgyzstan");

insert into country (co\_code, co\_name) values ("LA", "Lao People's Democratic Republic");

insert into country (co\_code, co\_name) values ("LV", "Latvia");

insert into country (co\_code, co\_name) values ("LB", "Lebanon");

insert into country (co\_code, co\_name) values ("LS", "Lesotho");

insert into country (co\_code, co\_name) values ("LR", "Liberia");

insert into country (co\_code, co\_name) values ("LY", "Libya");

insert into country (co\_code, co\_name) values ("LI", "Liechtenstein");

insert into country (co\_code, co\_name) values ("LT", "Lithuania");

insert into country (co\_code, co\_name) values ("LU", "Luxembourg");

insert into country (co\_code, co\_name) values ("MO", "Macao");

insert into country (co\_code, co\_name) values ("MK", "Macedonia, the Former Yugoslav Republic of");

insert into country (co\_code, co\_name) values ("MG", "Madagascar");

insert into country (co\_code, co\_name) values ("MW", "Malawi");

insert into country (co\_code, co\_name) values ("MY", "Malaysia");

insert into country (co\_code, co\_name) values ("MV", "Maldives");

insert into country (co\_code, co\_name) values ("ML", "Mali");

insert into country (co\_code, co\_name) values ("MT", "Malta");

insert into country (co\_code, co\_name) values ("MH", "Marshall Islands");

insert into country (co\_code, co\_name) values ("MQ", "Martinique");

insert into country (co\_code, co\_name) values ("MR", "Mauritania");

insert into country (co\_code, co\_name) values ("MU", "Mauritius");

insert into country (co\_code, co\_name) values ("YT", "Mayotte");

insert into country (co\_code, co\_name) values ("MX", "Mexico");

insert into country (co\_code, co\_name) values ("FM", "Micronesia, Federated States of");

insert into country (co\_code, co\_name) values ("MD", "Moldova, Republic of");

insert into country (co\_code, co\_name) values ("MC", "Monaco");

insert into country (co\_code, co\_name) values ("MN", "Mongolia");

insert into country (co\_code, co\_name) values ("ME", "Montenegro");

insert into country (co\_code, co\_name) values ("MS", "Montserrat");

insert into country (co\_code, co\_name) values ("MA", "Morocco");

insert into country (co\_code, co\_name) values ("MZ", "Mozambique");

insert into country (co\_code, co\_name) values ("MM", "Myanmar");

insert into country (co\_code, co\_name) values ("NA", "Namibia");

insert into country (co\_code, co\_name) values ("NR", "Nauru");

insert into country (co\_code, co\_name) values ("NP", "Nepal");

insert into country (co\_code, co\_name) values ("NL", "Netherlands");

insert into country (co\_code, co\_name) values ("NC", "New Caledonia");

insert into country (co\_code, co\_name) values ("NZ", "New Zealand");

insert into country (co\_code, co\_name) values ("NI", "Nicaragua");

insert into country (co\_code, co\_name) values ("NE", "Niger");

insert into country (co\_code, co\_name) values ("NG", "Nigeria");

insert into country (co\_code, co\_name) values ("NU", "Niue");

insert into country (co\_code, co\_name) values ("NF", "Norfolk Island");

insert into country (co\_code, co\_name) values ("MP", "Northern Mariana Islands");

insert into country (co\_code, co\_name) values ("NO", "Norway");

insert into country (co\_code, co\_name) values ("OM", "Oman");

insert into country (co\_code, co\_name) values ("PK", "Pakistan");

insert into country (co\_code, co\_name) values ("PW", "Palau");

insert into country (co\_code, co\_name) values ("PS", "Palestine, State of");

insert into country (co\_code, co\_name) values ("PA", "Panama");

insert into country (co\_code, co\_name) values ("PG", "Papua New Guinea");

insert into country (co\_code, co\_name) values ("PY", "Paraguay");

insert into country (co\_code, co\_name) values ("PE", "Peru");

insert into country (co\_code, co\_name) values ("PH", "Philippines");

insert into country (co\_code, co\_name) values ("PN", "Pitcairn");

insert into country (co\_code, co\_name) values ("PL", "Poland");

insert into country (co\_code, co\_name) values ("PT", "Portugal");

insert into country (co\_code, co\_name) values ("PR", "Puerto Rico");

insert into country (co\_code, co\_name) values ("QA", "Qatar");

insert into country (co\_code, co\_name) values ("RO", "Romania");

insert into country (co\_code, co\_name) values ("RU", "Russian Federation");

insert into country (co\_code, co\_name) values ("RW", "Rwanda");

insert into country (co\_code, co\_name) values ("RE", "Réunion");

insert into country (co\_code, co\_name) values ("BL", "Saint Barthélemy");

insert into country (co\_code, co\_name) values ("SH", "Saint Helena, Ascension and Tristan da Cunha");

insert into country (co\_code, co\_name) values ("KN", "Saint Kitts and Nevis");

insert into country (co\_code, co\_name) values ("LC", "Saint Lucia");

insert into country (co\_code, co\_name) values ("MF", "Saint Martin (French part)");

insert into country (co\_code, co\_name) values ("PM", "Saint Pierre and Miquelon");

insert into country (co\_code, co\_name) values ("VC", "Saint Vincent and the Grenadines");

insert into country (co\_code, co\_name) values ("WS", "Samoa");

insert into country (co\_code, co\_name) values ("SM", "San Marino");

insert into country (co\_code, co\_name) values ("ST", "Sao Tome and Principe");

insert into country (co\_code, co\_name) values ("SA", "Saudi Arabia");

insert into country (co\_code, co\_name) values ("SN", "Senegal");

insert into country (co\_code, co\_name) values ("RS", "Serbia");

insert into country (co\_code, co\_name) values ("SC", "Seychelles");

insert into country (co\_code, co\_name) values ("SL", "Sierra Leone");

insert into country (co\_code, co\_name) values ("SG", "Singapore");

insert into country (co\_code, co\_name) values ("SX", "Sint Maarten (Dutch part)");

insert into country (co\_code, co\_name) values ("SK", "Slovakia");

insert into country (co\_code, co\_name) values ("SI", "Slovenia");

insert into country (co\_code, co\_name) values ("SB", "Solomon Islands");

insert into country (co\_code, co\_name) values ("SO", "Somalia");

insert into country (co\_code, co\_name) values ("ZA", "South Africa");

insert into country (co\_code, co\_name) values ("GS", "South Georgia and the South Sandwich Islands");

insert into country (co\_code, co\_name) values ("SS", "South Sudan");

insert into country (co\_code, co\_name) values ("ES", "Spain");

insert into country (co\_code, co\_name) values ("LK", "Sri Lanka");

insert into country (co\_code, co\_name) values ("SD", "Sudan");

insert into country (co\_code, co\_name) values ("SR", "Suriname");

insert into country (co\_code, co\_name) values ("SJ", "Svalbard and Jan Mayen");

insert into country (co\_code, co\_name) values ("SZ", "Swaziland");

insert into country (co\_code, co\_name) values ("SE", "Sweden");

insert into country (co\_code, co\_name) values ("CH", "Switzerland");

insert into country (co\_code, co\_name) values ("SY", "Syrian Arab Republic");

insert into country (co\_code, co\_name) values ("TW", "Taiwan, Province of China");

insert into country (co\_code, co\_name) values ("TJ", "Tajikistan");

insert into country (co\_code, co\_name) values ("TZ", "Tanzania, United Republic of");

insert into country (co\_code, co\_name) values ("TH", "Thailand");

insert into country (co\_code, co\_name) values ("TL", "Timor-Leste");

insert into country (co\_code, co\_name) values ("TG", "Togo");

insert into country (co\_code, co\_name) values ("TK", "Tokelau");

insert into country (co\_code, co\_name) values ("TO", "Tonga");

insert into country (co\_code, co\_name) values ("TT", "Trinidad and Tobago");

insert into country (co\_code, co\_name) values ("TN", "Tunisia");

insert into country (co\_code, co\_name) values ("TR", "Turkey");

insert into country (co\_code, co\_name) values ("TM", "Turkmenistan");

insert into country (co\_code, co\_name) values ("TC", "Turks and Caicos Islands");

insert into country (co\_code, co\_name) values ("TV", "Tuvalu");

insert into country (co\_code, co\_name) values ("UG", "Uganda");

insert into country (co\_code, co\_name) values ("UA", "Ukraine");

insert into country (co\_code, co\_name) values ("AE", "United Arab Emirates");

insert into country (co\_code, co\_name) values ("GB", "United Kingdom");

insert into country (co\_code, co\_name) values ("US", "United States");

insert into country (co\_code, co\_name) values ("UM", "United States Minor Outlying Islands");

insert into country (co\_code, co\_name) values ("UY", "Uruguay");

insert into country (co\_code, co\_name) values ("UZ", "Uzbekistan");

insert into country (co\_code, co\_name) values ("VU", "Vanuatu");

insert into country (co\_code, co\_name) values ("VE", "Venezuela, Bolivarian Republic of");

insert into country (co\_code, co\_name) values ("VN", "Viet Nam");

insert into country (co\_code, co\_name) values ("VG", "Virgin Islands, British");

insert into country (co\_code, co\_name) values ("VI", "Virgin Islands, U.S.");

insert into country (co\_code, co\_name) values ("WF", "Wallis and Futuna");

insert into country (co\_code, co\_name) values ("EH", "Western Sahara");

insert into country (co\_code, co\_name) values ("YE", "Yemen");

insert into country (co\_code, co\_name) values ("ZM", "Zambia");

insert into country (co\_code, co\_name) values ("ZW", "Zimbabwe");

insert into country (co\_code, co\_name) values ("AX", "Åland Islands");

Refer subsequent hands on exercises to implement the features related to country.

**Hands on 6**

**Find a country based on country code** 

* Create new exception class CountryNotFoundException in com.cognizant.spring-learn.service.exception
* Create new method findCountryByCode() in CountryService with @Transactional annotation
* In findCountryByCode() method, perform the following steps:
  + Method signature

@Transactional

public Country findCountryByCode(String countryCode) throws CountryNotFoundException

* Get the country based on findById() built in method

Optional<Country> result = countryRepository.findById(countryCode);

* From the result, check if a country is found. If not found, throw CountryNotFoundException

if (!result.isPresent())

* Use get() method to return the country fetched.

Country country = result.get();

* Include new test method in OrmLearnApplication to find a country based on country code and compare the country name to check if it is valid.

    private static void getAllCountriesTest() {

        LOGGER.info("Start");

        Country country = countryService.findCountryByCode("IN");

  LOGGER.debug("Country:{}", country);

        LOGGER.info("End");

    }

* Invoke the above method in main() method and test it.

**NOTE:** SME to explain the importance of @Transactional annotation. Spring takes care of creating the Hibernate session and manages the transactionality when executing the service method.

**Hands on 7**

**Add a new country** 

* Create new method in CountryService.

@Transactional

public void addCountry(Country country)

* Invoke save() method of repository to get the country added.

countryRepository.save(country)

* Include new testAddCountry() method in OrmLearnApplication. Perform steps below:
  + Create new instance of country with a new code and name
  + Call countryService.addCountry() passing the country created in the previous step.
  + Invoke countryService.findCountryByCode() passing the same code used when adding a new country
  + Check in the database if the country is added

**Hands on 8**

**Update a country based on code** 

* Create a new method updateCountry() in CountryService with parameters code and name. Annotate this method with @Transactional. Implement following steps in this method.
  + Get the reference of the country using findById() method in repository
  + In the country reference obtained, update the name of country using setter method
  + Call countryRepository.save() method to update the name
* Include new test method in OrmLearnApplication, which invokes updateCountry() method in CountryService passing a country's code and different name for the country.
* Check in database table if name is modified.

**Hands on 9**

**Delete a country based on code** 

* Create new method deleteCountry() in CountryService. Annotate this method with @Transactional.
* In deleteCountry() method call deleteById() method of repository.
* Include new test method in OrmLearnApplication with following steps
  + Call the delete method based on the country code during the add country hands on
* Check in database if the country is deleted