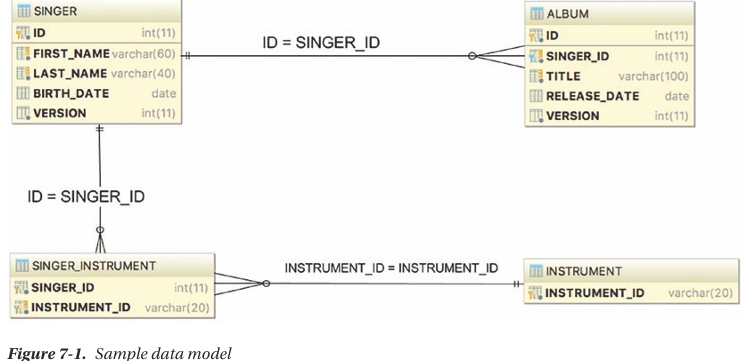
# 6. Spring Data Access with JDBC



## 6.1 Sample Data Model for Example Code

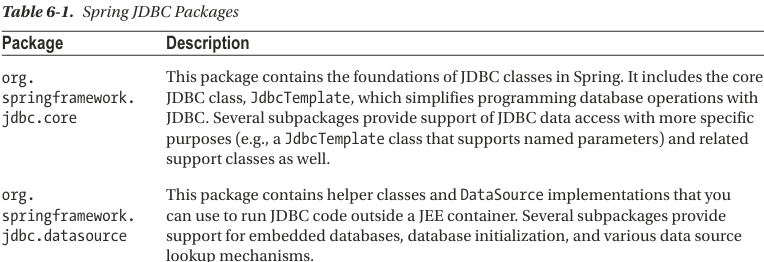
-DAO: data access object, Spring world: repository

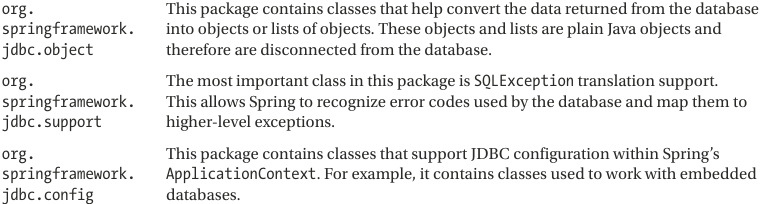
## 6.2 Explore JDBC Infrastructure

## 6.3 Spring JDBC Infrastructure

6.3.1 Overview and used Packages

-JDBC support in Spring is divided into 5 packages



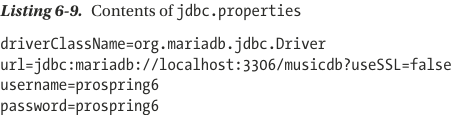


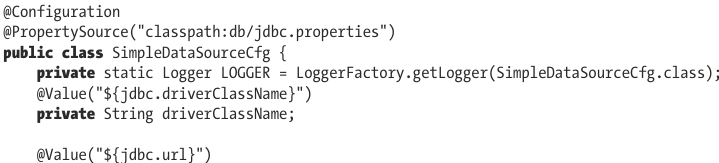
6.3.2 Database Connections and DataSources

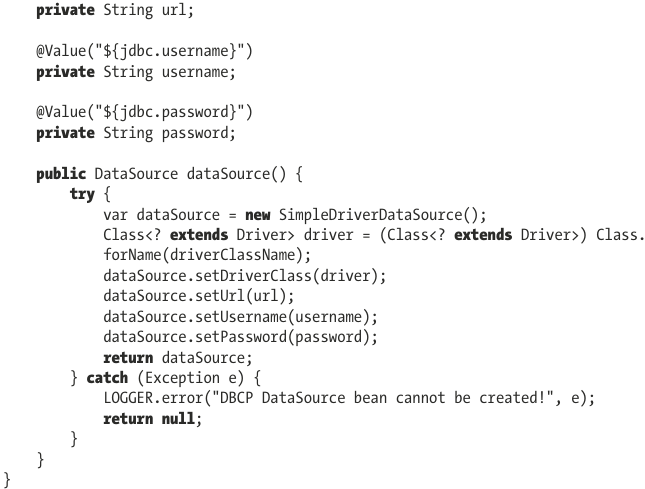
-javax.sql.DataSource: implement a bean that manage database connection

-DriverManagerDataSource is the simplest implementation of a DataSource. But it doesn’t support database connection pooling -> unsuitable

+You need to supply driver class name, connection URL, username, password.



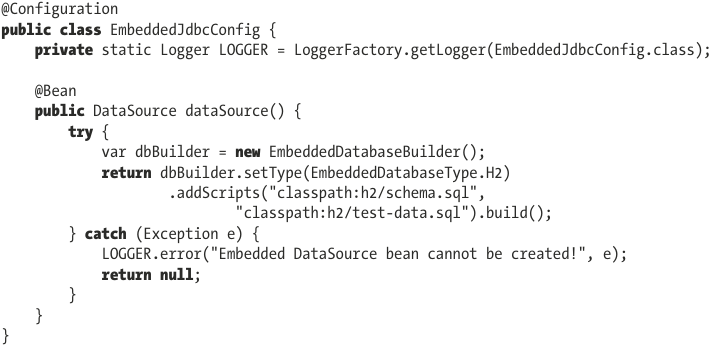




6.3.3 Embedded Database Support

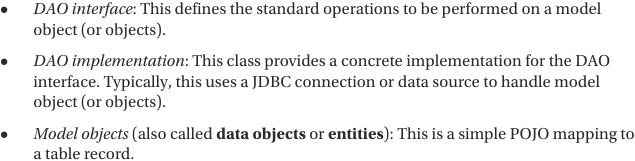
-Spring offers embedded database support, which automatically starts an embedded database and exposes it as a DataSource for application -> useful for local development or unit testing.

-Set up embedded H2 database:

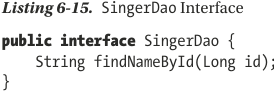


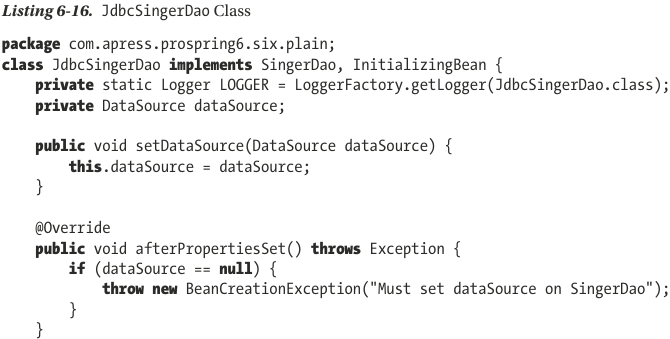
6.3.4 Use DataSources in DAO Classes

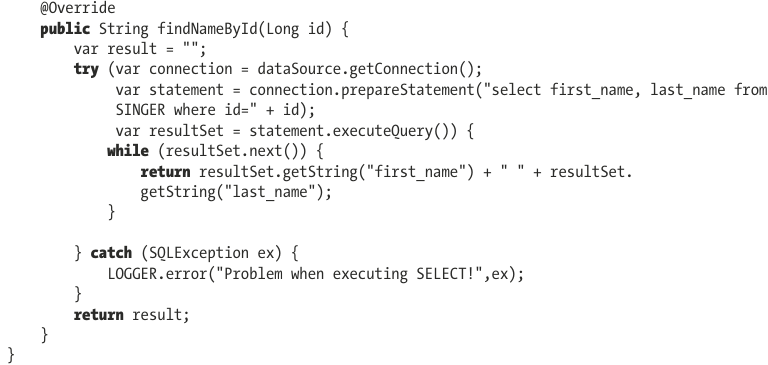
-DAO pattern is used to separate low-level data accessing API or operations from high-level business services

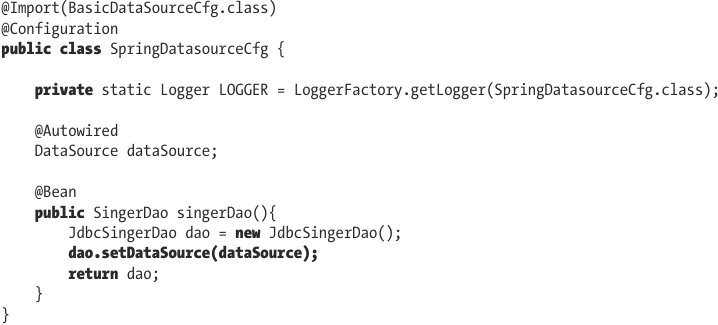


-Example:









6.3.5 Exception Handling

-Spring provides a default implementation of SQLExceptionTranslator interface, which take care of translation generic SQL error codes into Spring JDBC exceptions.

6.3.6 JdbcTemplate Class

-This class represents the core of Spring’s JDBC support.

-Initializing JdbcTemplate in DAO class: construct the class by passing it in data source object

6.3.7 Spring Classes that Model JDBC Operations

6.3.8 Spring Data Project: JDBC Extensions

6.3.9 Spring JDBC Testing Annotations

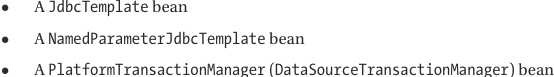
6.3.10 Introduce Testcontainers

6.3.11 Considerations for Using JDBC

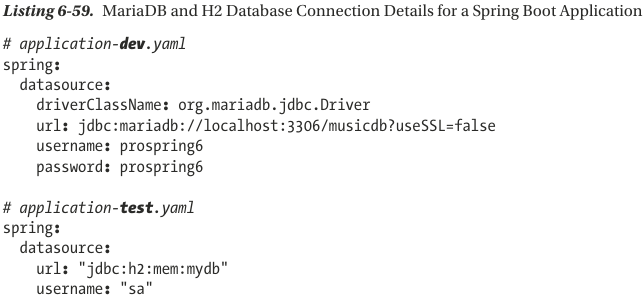
6.3.12 Spring Boot JDBC

-spring-boot-starter-jdbc: not add database driver

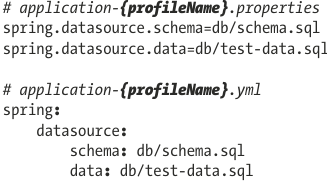
-It uses HikarciCP to configure DataSource bean, which is a production-ready connection poor library. If there’s no DataSource bean explicitly and there’s an embedded database driver in classpath, SB will automatically register DataSource bean using in-memory database setting. SB also registers these beans automatically:



-SB database connection details are configured in: application-{profilename}.properties/yaml



+SB looks for embedded database initialization files under src/main/resources: schema.sql and data.sql. The location and names for these files can be configured:



# 7. Spring with Hibernate

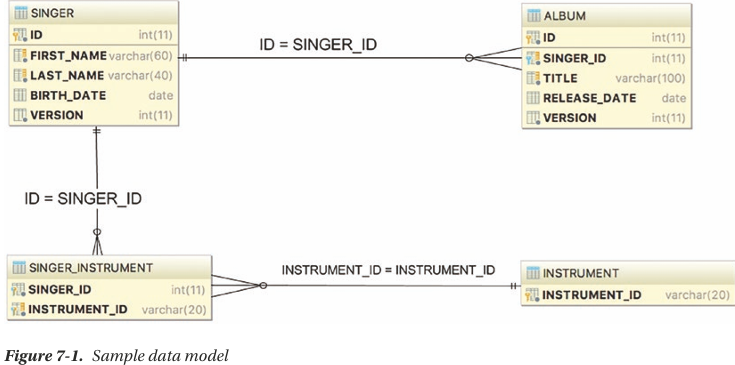
-**Persistent frameworks**: MyBatis, Hibernate

-**jOOQ**: database-mapping library: generates Java code from database and lets you build type-safe SQL queries through its fluent API.

-Main **ff** of **ORM** library: close the gap between relational data structure in RDBMS and OO model in Java so that developers can focus on programming with object model and easily perform actions related to persistence.

-Hibernate provides an implementation of JPA. So when you develop app with Hibernate, you can use either Hibernate’s own API or JPA API with Hibernate as persistence service provider.

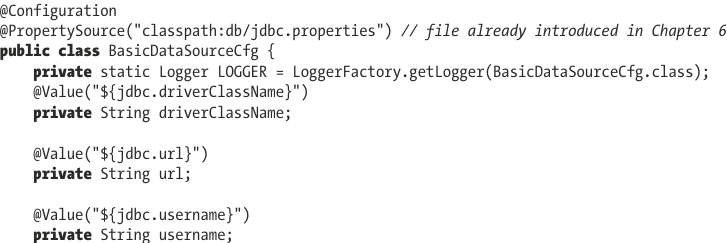
## 7.1 Sample Data Model for Example Code

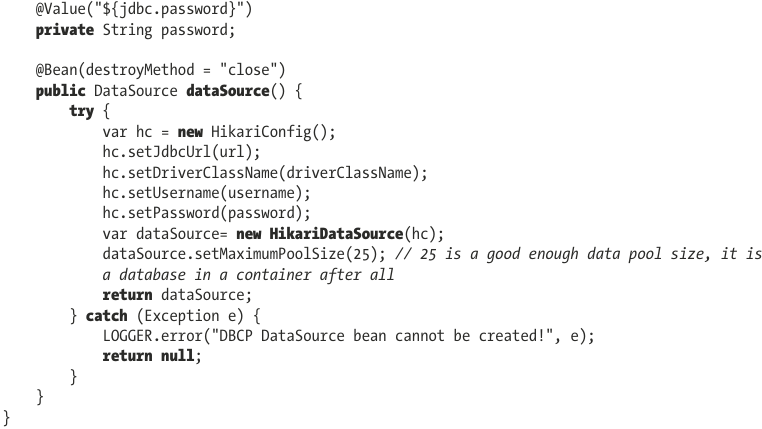


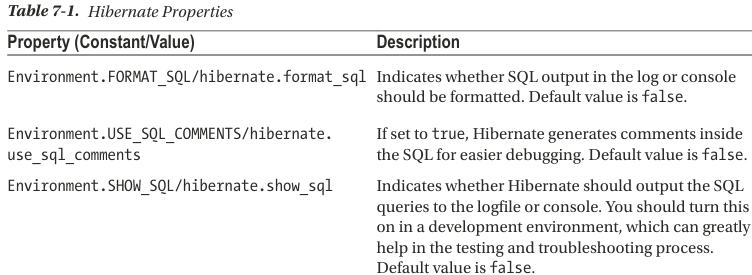
## 7.2 Configure Hibernate’s SessionFactory

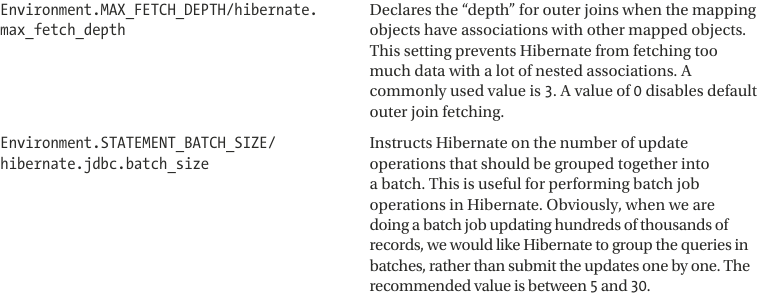
-The core concept of Hibernate is based on **org.hibernate.Session** interface, which is obtained from **org.hibernate.SessionFactory**.

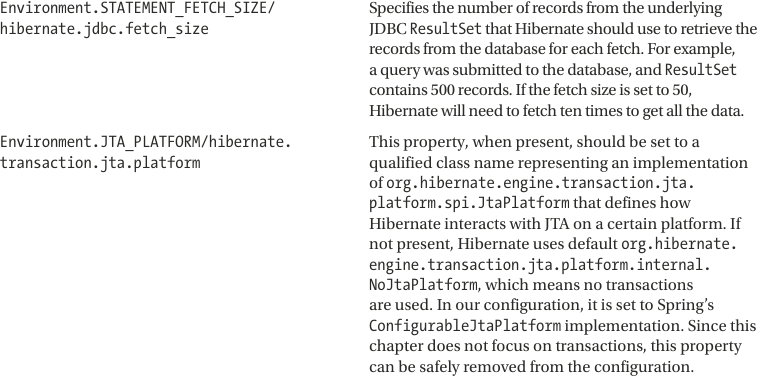
-You must add hibernate-core-jakarta and spring-orm. Spring Hibernate configuration built on top of a DataSource configuration.

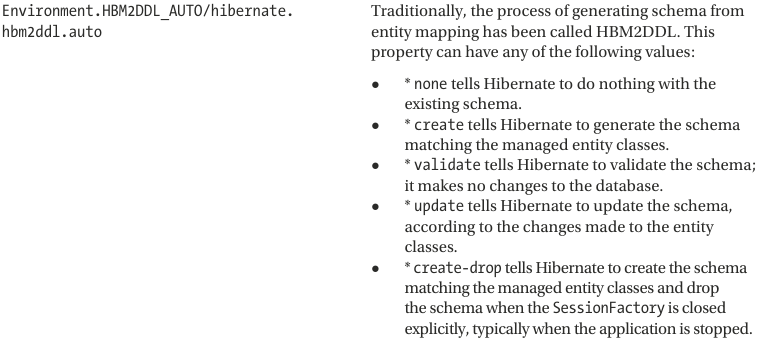


  
-Hibernate properties:









## 7.3 ORM Mapping Using Hibernate Annotations

-2 approaches to mapping:

+Design object model and then generate database scripts->not suitable for production, but rather for test that use-in-memory database

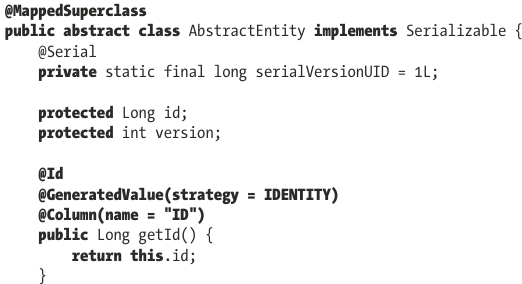
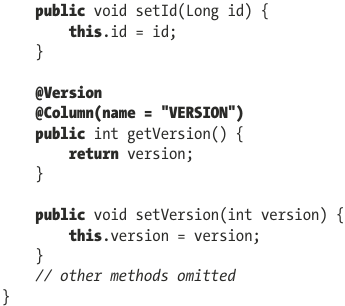
+Start with data model (table) and then model POJOs: recommend for development and production.



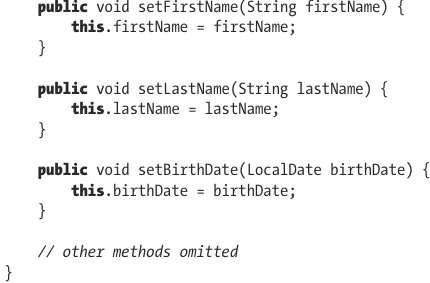
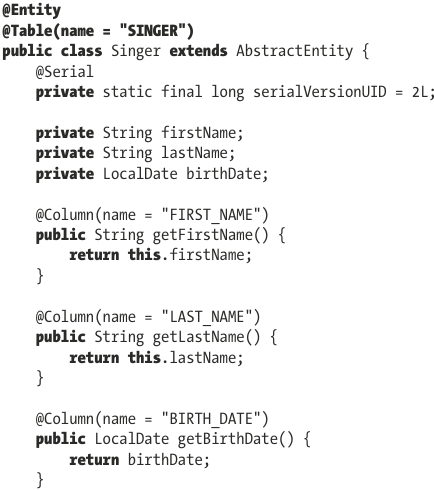
-AbstractEntity: group fields in common in Singer and Album. In production applications, other files could be: createdAt, updatedAt, createdBy, updatedBy… @MappedSuperClass: designates a class whose mapping information is applied to entities that inherit from it.

## 7.4 Simple Mappings

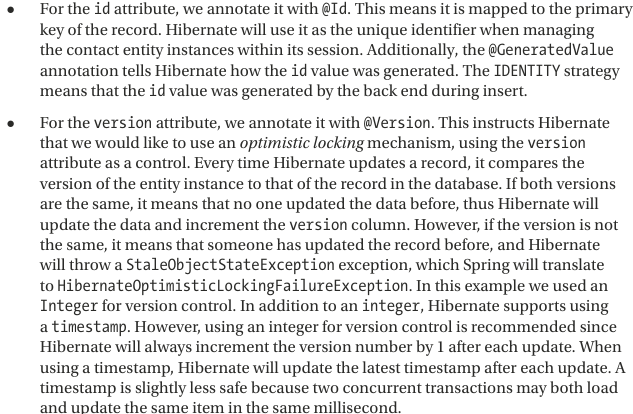
-AbstractEntity:

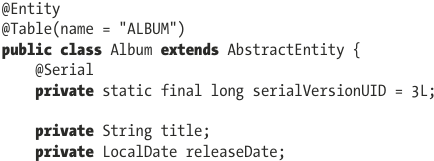
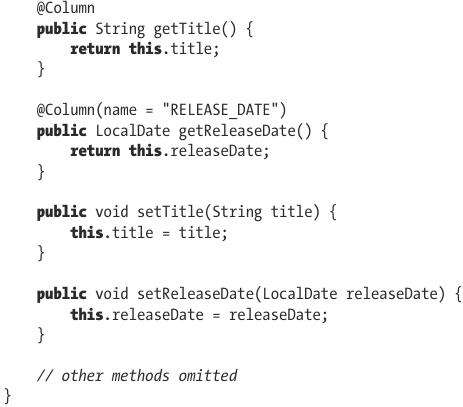
-Singer:



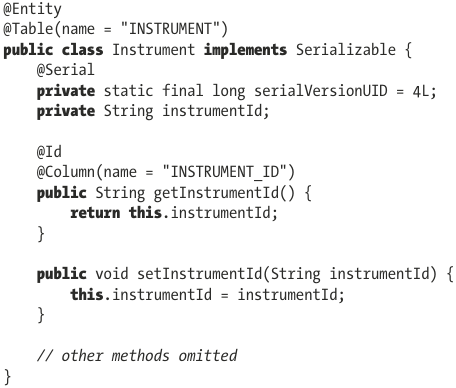
+Few highlights:



-Album:

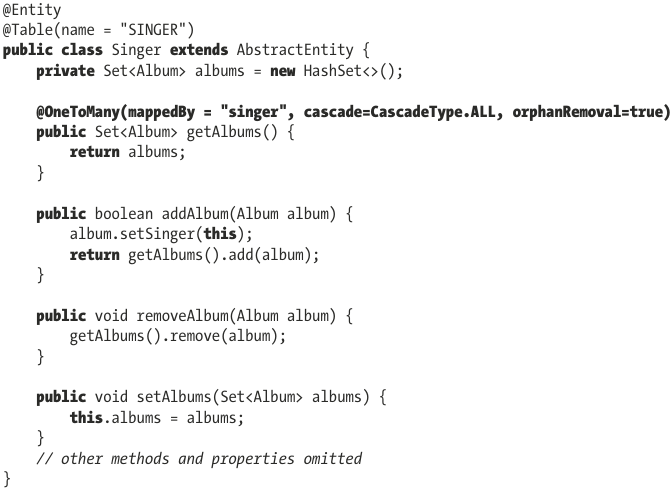
 

-Instrument:



### 7.4.1 One-to-Many Mappings

-Singer:

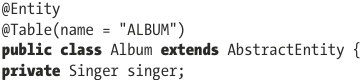


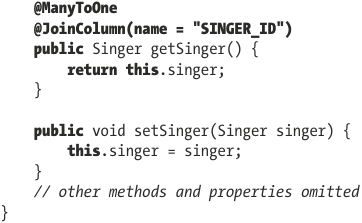
+**mappedBy**: indicate the property for “one” class

+**CascadeType.ALL**: any operation done on Singer are propagated to albums linked to it.

+**orphanRemoval**: after albums set has been updated, Album entries that no longer exist in set should be deleted from database.

-Album:

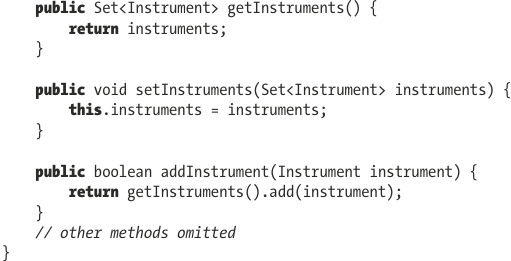
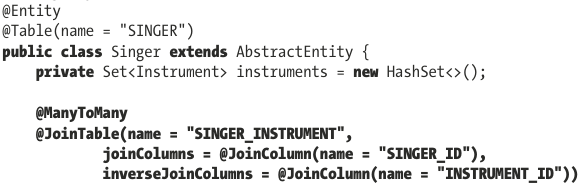




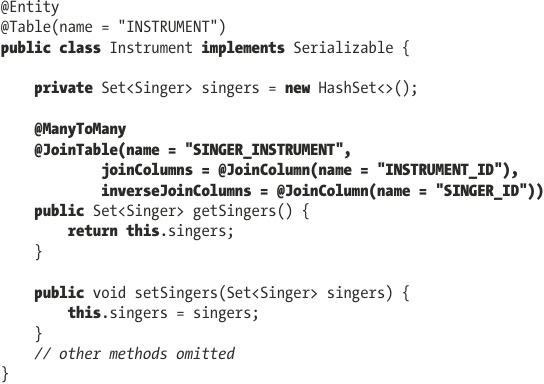
+@ManyToOne: @JoinColumn: specify the foreign-key column name.

### 7.4.2 Many-to-Many Mappings

-Singer:



-Instrument:



+getters is annotated with @ManyToMany.

+@JoinTable: indicate the underlying join table that Hibernate should look for. Name is join table’s name, joinColumns defines the column that is the foreign key in this table. inverseJoinColumns defines the column that is the foreign key to other side.

### 7.4.3 The Hibernate Session Interface

-In Hibernate, when interacting with database, the main interface to deal with is Session which obtained from an instance of SessionFactory.

-A SessionFactory instance is created and added to application context by configuring LocalSessionFactory bean in hibernate config. This istance can then be autowired in repositoriy class to create Hibernate Session used to communicate with database.

## 7.5 Query Data by Using Hibernate Query Language

### 7.5.1 Simple Querying with Lazy Fetching

### 7.5.2 Querying with Associations Fetching

## 7.6 Insert Data

## 7.7 Update Data

## 7.8 Delete Data

## 7.9 Execute SQL Native Queries

## 7.10 Execute Projections with Hibernate

## 7.11 Call Stored Functions with Hibernate

## 7.12 Configure Hibernate to Generate Tables from Entities

## 7.13 Annotate Methods or Fields?

## 7.14 Considerations when using Hibernate

## 7.15 Introduce jOOQ

# 8. Spring with JPA

8.1 Introduce JPA 3.1

8.1.1 Sample Data Model for Example Code

8.1.2 Configure JPA’s EntityManagerFactory

8.1.3 Use JPA Annotations for ORM Mapping

8.2 Perform Database Operations with JPA

8.2.1 Use Java Persistence Query Language to Query Data

8.2.2 Query with Untyped Results

8.2.3 Query for a Custom Result Type with a Constructor Expression

8.2.4 Insert Data

8.2.5 Update Data

8.2.6 Delete Data

8.3 Use a Native Query

8.3.1 Use a Simple native query

8.3.2 Native Querying with SQL ResultSet Mapping

8.3.3 Execute Stored Functions and Procedures

8.4 Using JPA Criteria API for a Criteria Query

# 9. Spring Transaction Management

-Transaction management is a crosscutting concern and shouldn’t be coded within business code.

## 9.1 Explore Spring Transaction Abstraction Layer

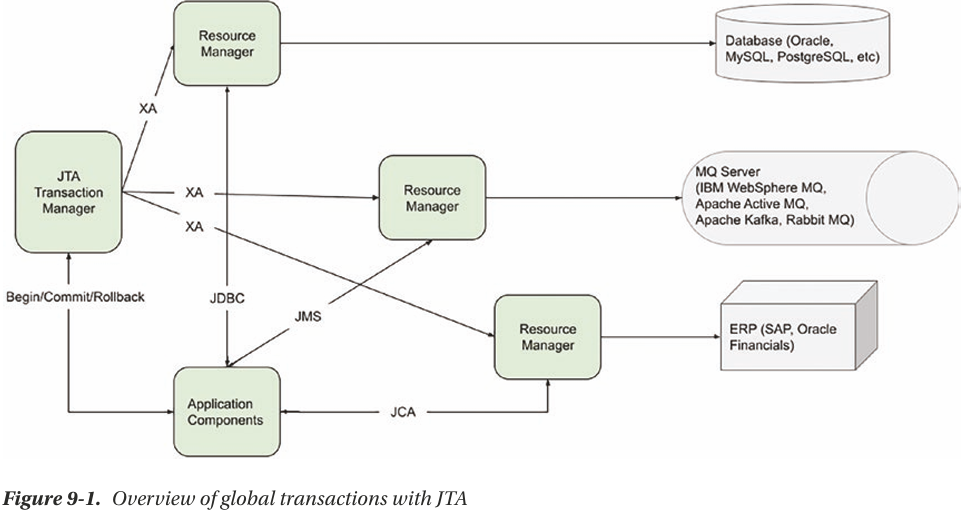
-Local transactions: single transactional resource (a JDBC connection).

-Global transaction: managed by container and can span multiple transactional resources

-Transaction Types

+Local transactions are easy to manage, all operations need to interact with just one transactional resource.

+Global transactions are implemented with JTA (Java/Jakarta Transaction API).



+The 1st party is back-end resource (RDBMS, messaging middleware, enterprise resource planning system…)

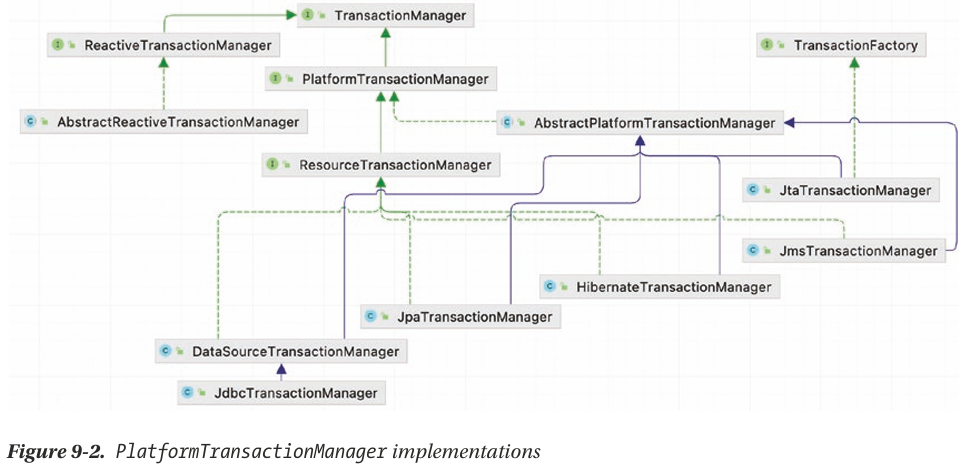
+The 2nd party is resource manager, provided by back-end resource vender is responsible for interacting with back-end resource.

+The 3rt party is JTA transaction manager: manage, coordinate, synchronize the transaction status with all resource managers that are participating in transaction.

+The final component is application. It manages transaction (begin, commit, rollback). Application interacts with back-end resources various standards defined by JEE.

## 9.2 Implementations of PlatformTransactionManager

-In Spring, PlatformTransactionManager interface uses TransactionDefinition and TransactionStatus interfaces to create and manage transaction.



+DataSourceTransactionManager is for generic JDBC conenctions.

+For ORM: JpaTransactionManager, HibernateTransactionManager.

## 9.3 Analyze Transaction Properties

-We discuss transaction properties that Spring supports, focusing on interacting with RDBMS as back-end resource.

-Transactions have 4 ACID properties (atomicity, consistency, isolation, durability)

### 9.3.1 TransactionDefinition Interface

-Controls the properties of transaction.

A screenshot of a computer code

AI-generated content may be incorrect.A computer code with black text

AI-generated content may be incorrect.

+getTimeout(): return the time the transaction must complete

+isReadOnly(): indicate whether the transaction is read-only.

+getName(): return the name of transaction

+getIsolationLevel(): controls what change to data other transactions see. Isolation levels are represented as static value defined

A screenshot of a computer

AI-generated content may be incorrect.

+getPropagationBehavior(): specifies what happens to a transaction call, depending on whether there is an active transaction.

A screenshot of a computer

AI-generated content may be incorrect.

### 9.3.2 TransactionStatus Interface

-Allows a transaction manager to control the transaction execution. The code can check whether the transaction is a new one or whether it’s a read-only transaction, and it can initiate a rollback.

-TransactionStatus behavior is split between 2 interfaces:

+TransactionExecution: declares basic transaction operations.

+SavepointManager: declares methods related to savepoints.

A screen shot of a computer code

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.A computer code with black text

AI-generated content may be incorrect.

+setRollbackOnly(): causes a rollback and ends the active transaction

+hasSavePoint(): returns whether the transaction internally carries a savepoint

+flush():flushes the session to a data store if applicable.

+isCompleted() returns whether the transaction has ended (committed or rolled back)

## 9.4 Sample Data Model and Infrastructure for Example Code

-@EnableTransactionManagement: enabling annotation-driven transaction capability. It’s responsible for registering all infrastructure Spring beans that support transaction management (TransctionInterceptor, methods annotated with @Transactional)

-Configuration with Spring Core:

+Basic Data Source:

A screen shot of a computer program

AI-generated content may be incorrect.

+Transaction Config:

A screen shot of a computer program

AI-generated content may be incorrect.

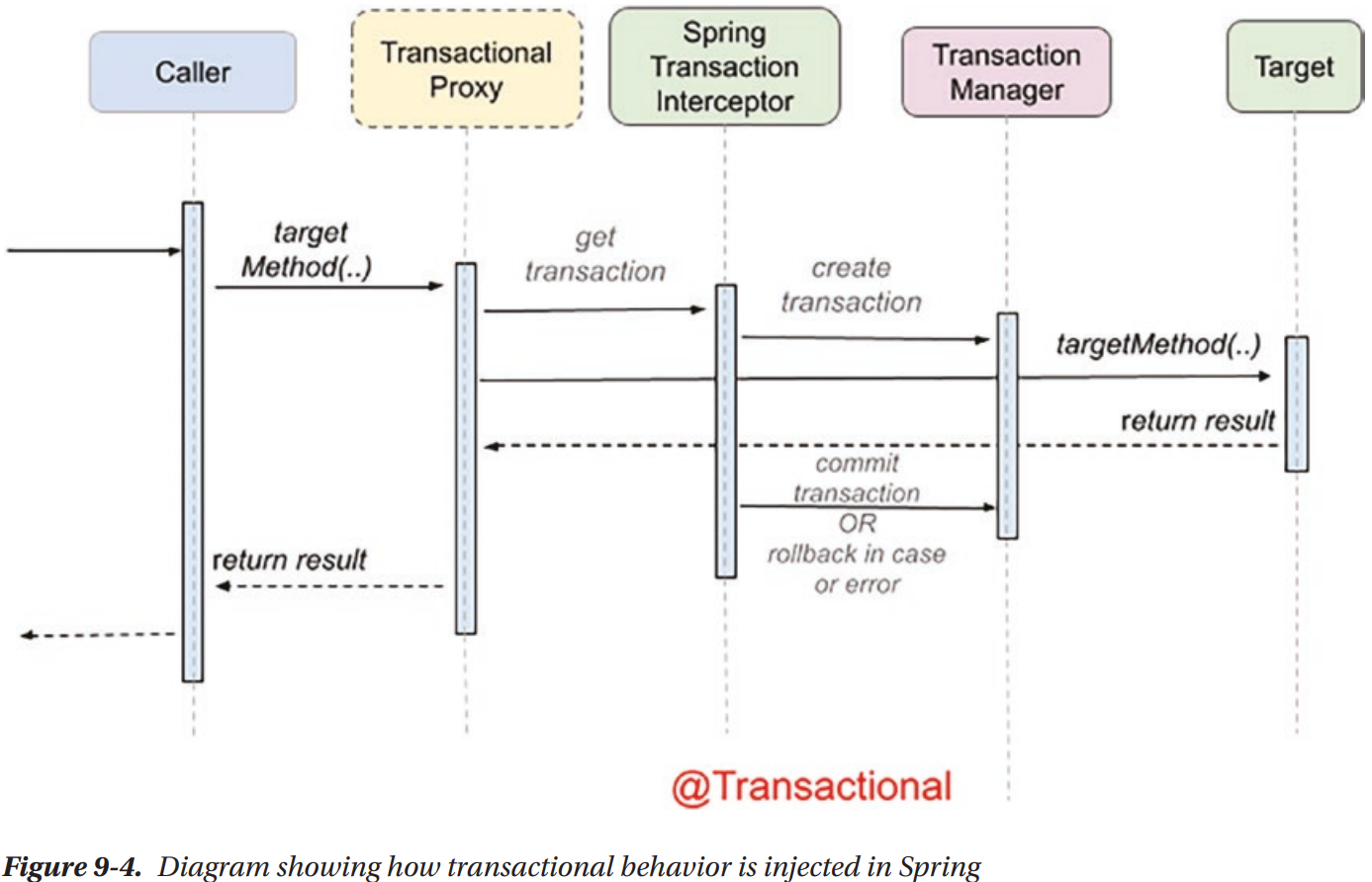
A screen shot of a computer code

AI-generated content may be incorrect.

### 9.4.1 Use Declarative Transactions

-Declarative: the behavior is declared using annotation on targeted method. There is no code to be written to create, start and end a transaction.

-transactionManager bean picks up @Transactional and wraps method within the desired transactional behavior. Bean declaration or methods annotated @Transactional is transformed at runtime into a bean that is wrapped into a proxy that injects transactional behavior for every targeted method.



-Service methods are configured to be executed in transactions, provide atomic behavior for methods that involve multiple database operations.

-@Transactional in class level: transaction is present before the execution of each method within the class.

A screenshot of a computer

AI-generated content may be incorrect.

A close up of a document

AI-generated content may be incorrect.

-@Transactional without any attribute: transaction propagation mode is required, default isolation level of datastore, default timeout of transaction system, the mode is read-write.

### 9.4.2 Roll Back Transactions

-**rollback**: an operation that returns the database to some previous state. It’s important for database integrity.

-By default, a transaction will be rolled back on RuntimeException and Error but not on checked exceptions (business exceptions).

### 9.4.3 Use Programmatic Transactions

-(See in book)

### 9.4.4 Considerations on Transaction Management

-The declarative approach is recommended in all cases.

## 9.5 Transactional Configuration with Spring Boot

-Main dependencies is spring-boot-starter-jdbc and add transactional-specific components requires: spring-orm and hibernate-core.

-Configuration file:

A computer code with black text

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

+spring.datasource: configure data source: connection URL, credentials, connection pooling

+spring. jpa: persistence unit.

-Spring Boot autoconfigure persistence layer by spring.jpa: it configures jpaVendorAdapter bean of type HibernateJpaVenderAdapter, a transactionManager bean of type JpaTransactionManager, Properties beans for Hibernate and JPa… but not for LocalSessionFactoryBean

A screen shot of a computer program

AI-generated content may be incorrect.

### 9.5.1 Transactional Tests

### 9.5.2 Considerations on Transaction Management

# 10. Spring Data with SQL and NoSQL Database

10.1 Introduce Spring Data JPA

10.1.1 Use Spring Data JPA Repository Abstraction for Database Operations

10.1.2 Use JpaRepository

10.1.3 Spring Data JPA with Custom Queries

10.1.4 Projection Queries

10.2 Keep track of changes on Entity Class

10.3 Keep Entity Versions by Using Hibernate Envers

10.3.1 Add Tables for Entity Versioning

10.3.2 Configure EntityManagerFactory for Entity Versioning

10.3.3 Enable Entity Versioning and History Retrieval

10.3.4 Custom Implementation for Spring Data Repositories

10.4 Spring Boot Data JPA

10.5 Spring Data with MongoDB

10.6 Considerations using Spring Data