3. Working with data

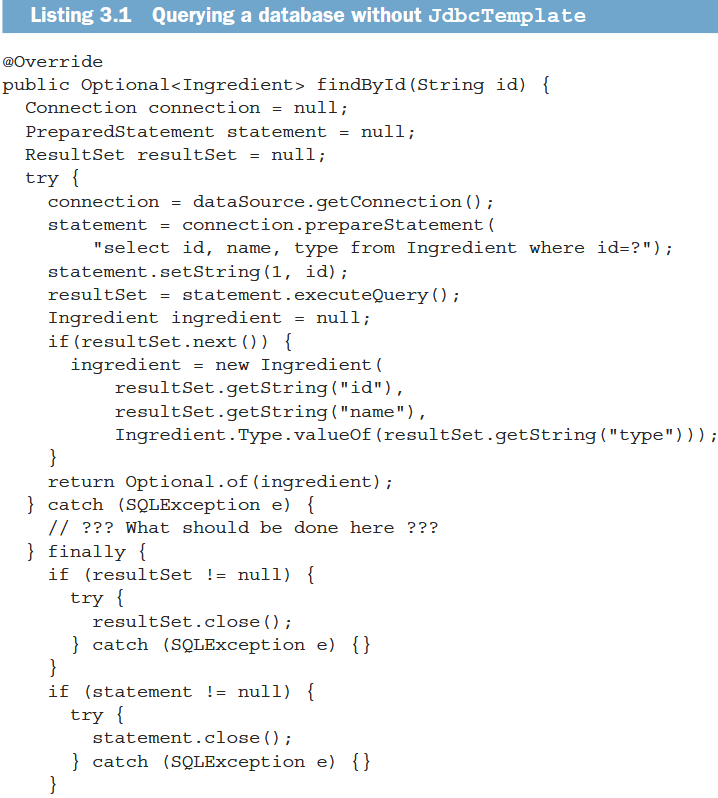
# 3.1 Reading and writing data with JDBC

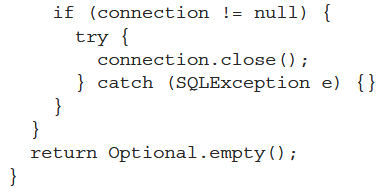
-Relational database and SQL are leading choice for data persistence.

-Working with relational data: JDBC and JPA

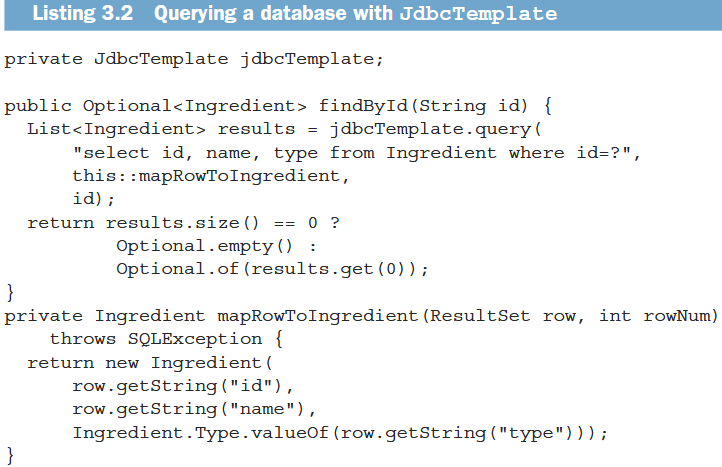
-Spring JDBC support in **JdbcTemplate** class: provide means that developer can perform SQL operations against relational database without preparing configuration JDBC

-Query a database without JdbcTemplate:





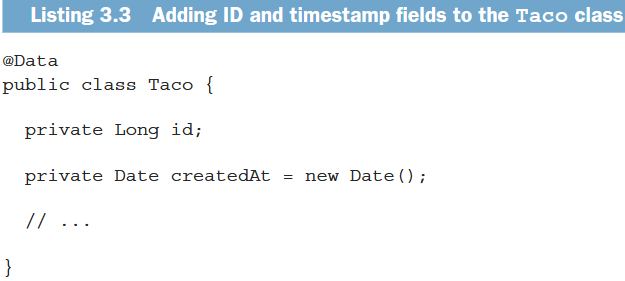
-Use JdbcTemplate

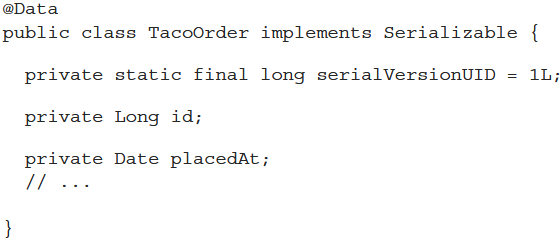


## 3.1.1 Adapting the domain for persistence

-Persisting objects to a database, it’s a good idea to have one field that uniquely identifies the object.

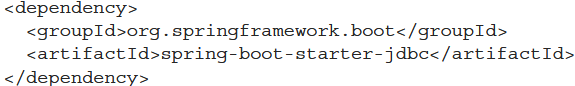
-It’s useful to know when Taco is created and TacoOrder is placed. Also add a field to each object to capture the data and time that the objects are saved.



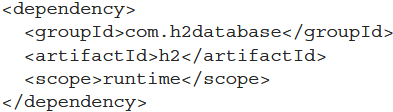


## 3.1.2 Working with JdbcTemplate

-Add JdbcTemplate to the project classpath.



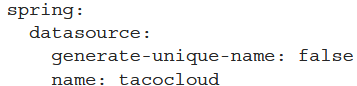
-For development purposes, use H2 embedded database



+By default, the database name is automatically generated. It’s ideal to pin down the database name:



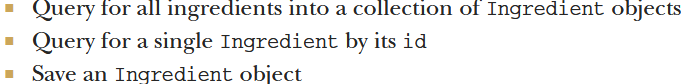
Or change to application.yml by YAML format:



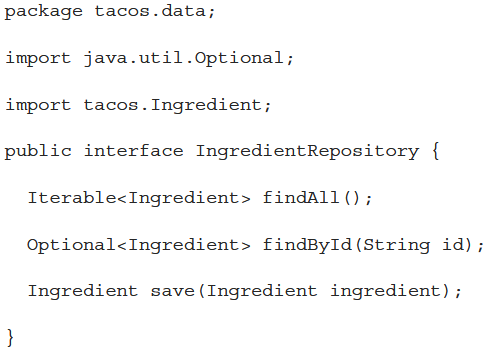
+The database URL is “**jdbc:h2:mem:tacocloud**”

-Defining JDBC Repositories

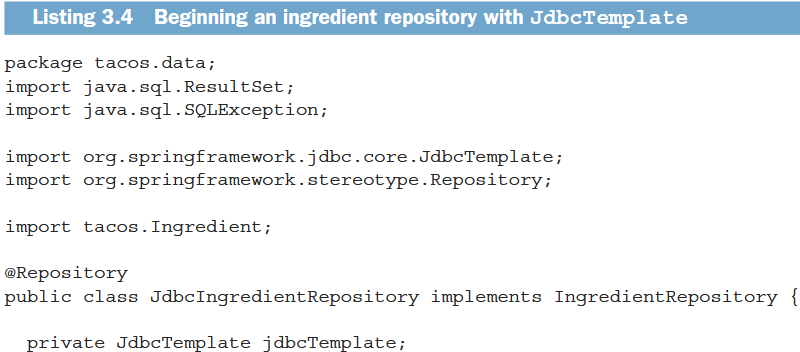
+Ingredient repository performs operations:

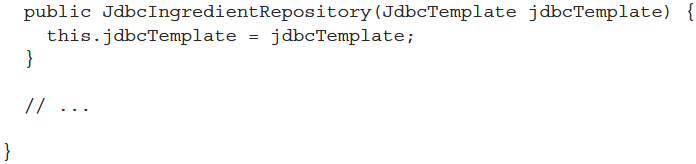


+**IngredientRepository** interface defines 3 operations:



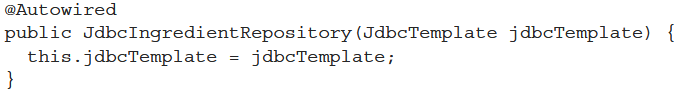
+You still need to write an **implementation** of IngredientRepository using **JdbcTemplate** to query database:



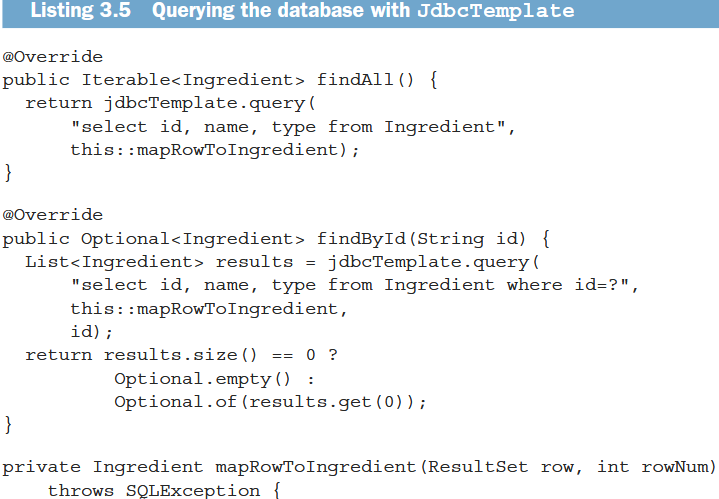


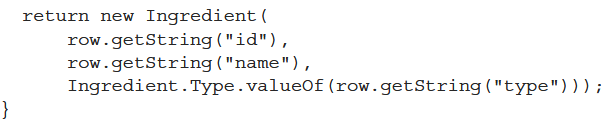
+JdbcIngredientRepository is annotated with **@Repository**: It should be automatically discovered by Spring component scanning and instantiated as a bean in Spring application context.

+When Spring creates JdbcIngredientRepository bean, it **injects** it with **JdbcTemplate**. If there is more one constructor, annotate constructor with **@Autowired**



+3 others methods:



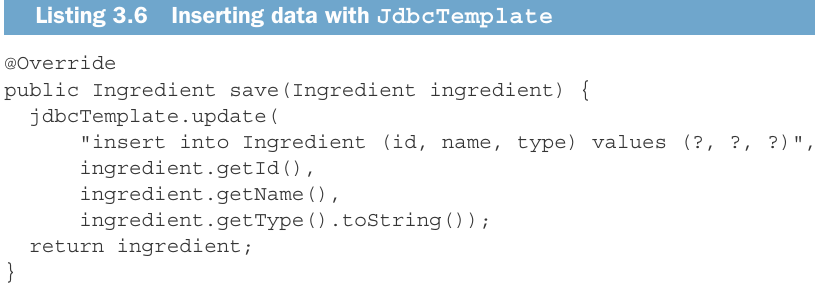


+**query()** accepts SQL for the query and an **implementation** of Spring’s **RowMapper** to **map** each row in the result set to object.

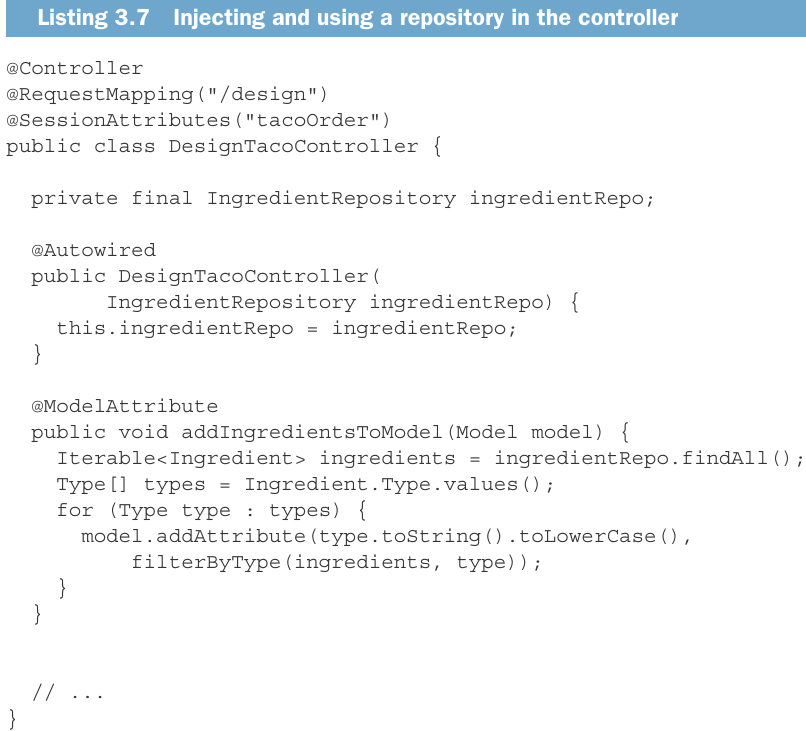
+Reading data from database is only part of the story. Let see how to write data to database.

-Inserting a row:

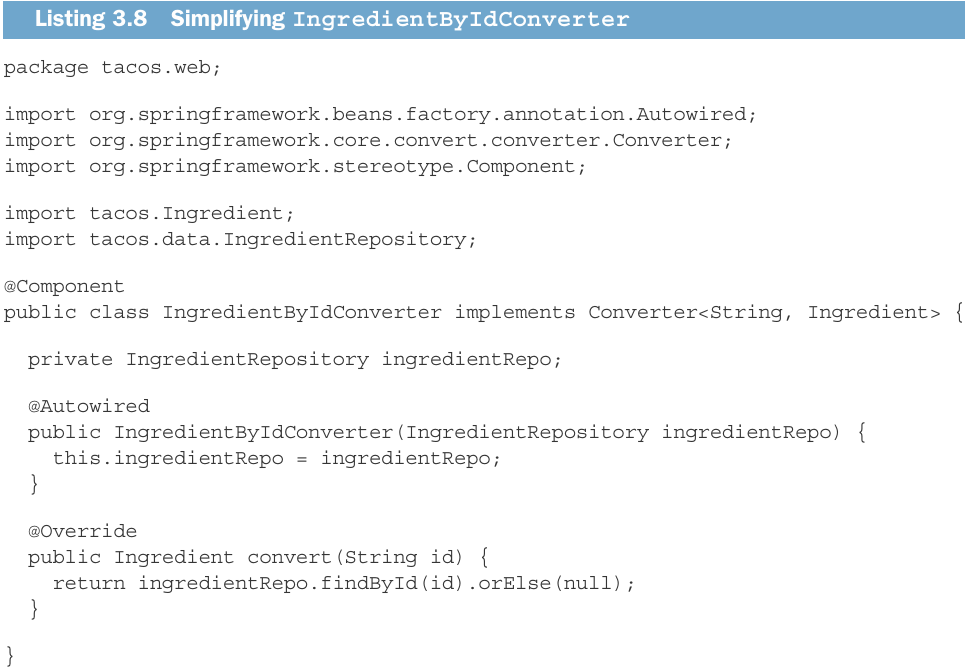
+JdbcTemplate **update()**: **write** or **updates** data



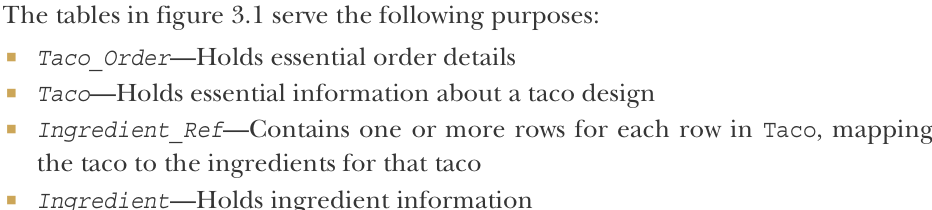
-Now inject JdbcIngredientRepository into DesignTacoController to provide a list of Ingredient objects:

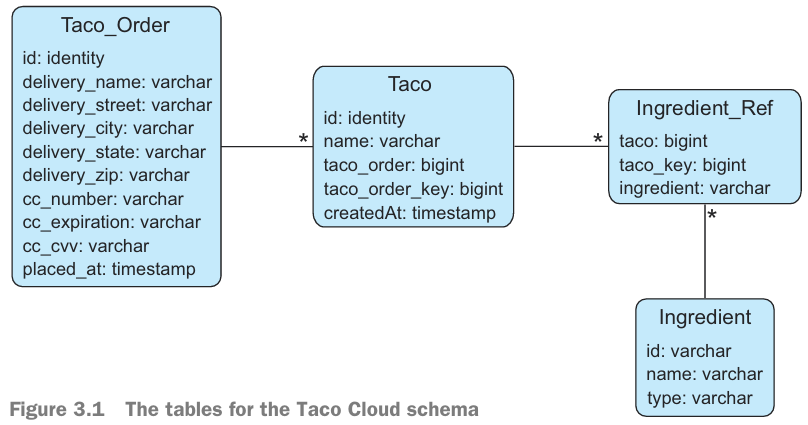


-Simplify the **IngredientByIdConverter**



## 3.1.3 Defining a schema and preloading data

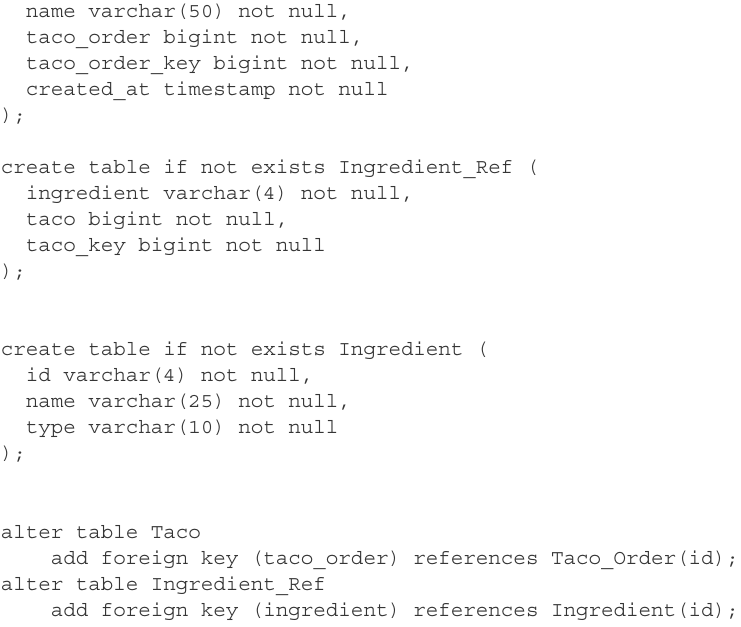
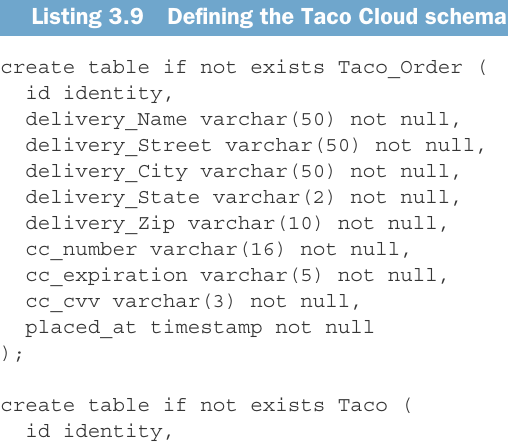




+Taco\_Order and Taco are members of an aggregate where Taco\_Order is the aggregate root.

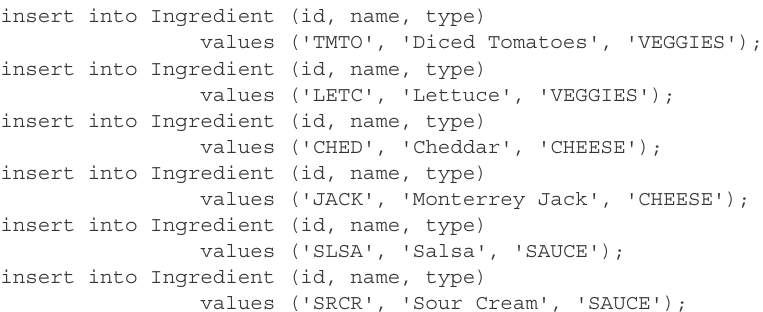
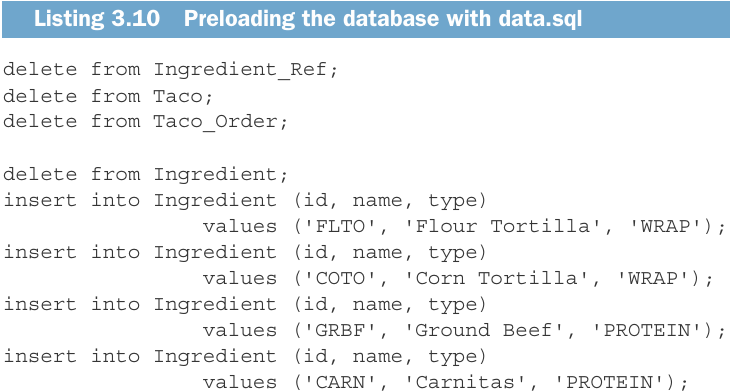
-Note: Aggregates and aggregate roots are core concept of **domain-driven design (DDD)**: The structure and language of software code should match the business domain. See more: *Domain-Driven Design: Tackling Complexity in the Heart of Software (https://www.dddcommunity .org/book/evans\_2003/)*

-SQL create tables:



-Spring Boot puts this schema definition. If there’s a file named schema.sql in the root of app’s classpath, SQL in that file will be executed when the app starts.->Place schema.sql in src/main/resource

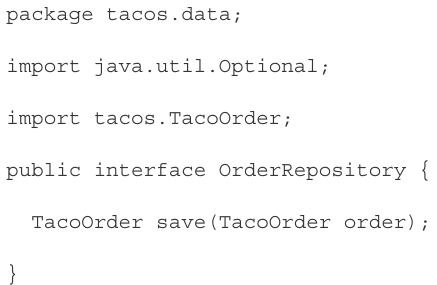
-Preload the database: SB will also execute a file data.sql from the root of classpath when app starts.



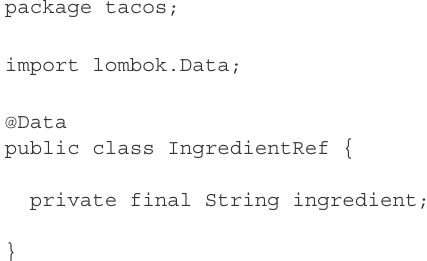
## 3.1.4 Inserting data

-Taco objects don’t exist outside of the content of a TacoOrder. We only need to define a repository to persist TacoOrder object.

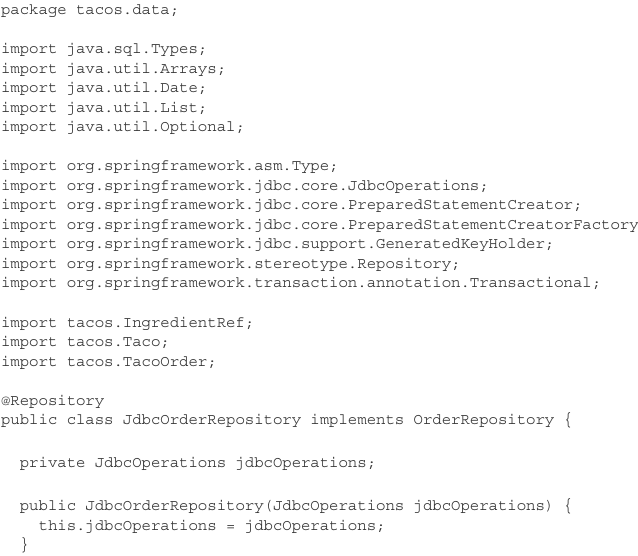
-**OrderRepository** interface:

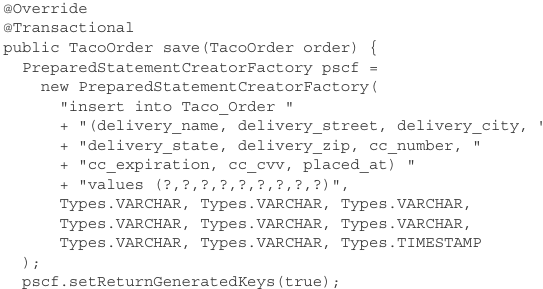


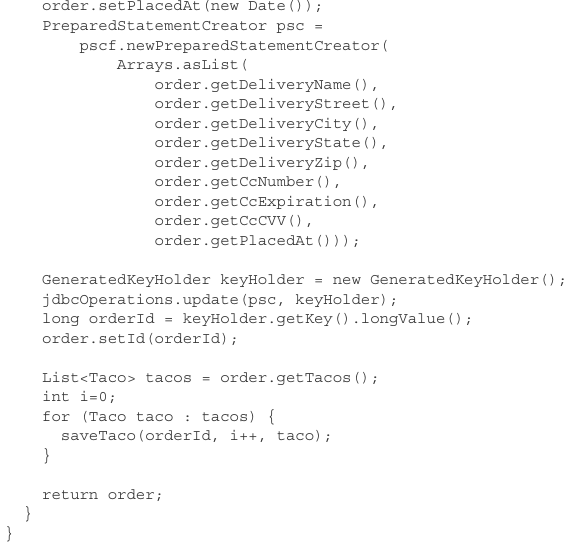
+When you save TacoOrder, also save Taco objects -> save an object that represents the link between Taco and each Ingredient-> **IngredientRef** class defines that link.



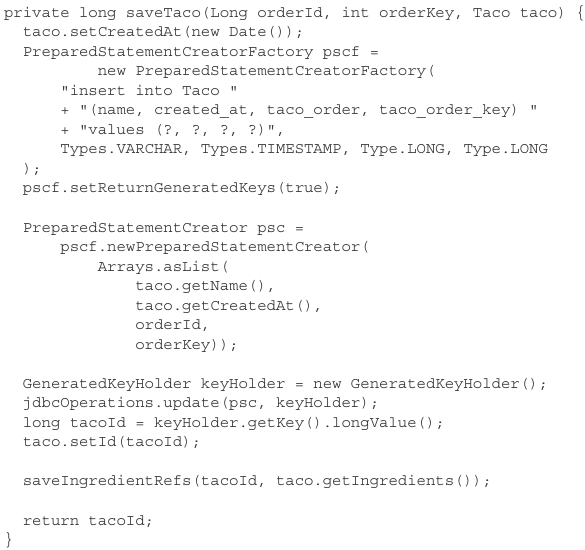
-id property on Taco\_Order table is **identity**: the database determines the value automatically. Spring offers **GeneratedKeyHolder** type to return that value. It involves working with **prepared statement** in **save()**:







-The order has been saved, but need to also save Taco objects associated with the order: **saveTaco()**



# 3.2 Working with Spring Data JDBC

## 3.2.1 Adding Spring Data JDBC to the build

## 3.2.2 Defining repository interfaces

## 3.2.3 Annotating the domain for persistence

## 3.2.4 Preloading data with CommandLineRunner

# 3.3 Persisting data with Spring Data JPA

## 3.3.1 Adding Spring Data JPA to the project

## 3.3.2 Annotating the domain as entities

## 3.3.3 Declaring JPA repositories

## 3.3.4 Customizing repositories