# Implementing Spring Profiles for Data Loading

## Spring Feature Description

Spring Profiles allow for conditional bean registration based on the active profile, enabling different configurations for different environments (e.g., development, testing, production). In this implementation, we will configure the application to read flight data from either `flights.csv` or `flights_small.csv` based on the active profile.

## Why Do We Need It in Flight Application?

The ability to switch data sources based on the active profile enhances the flexibility and adaptability of the application. During development or testing, using a smaller dataset (`flights_small.csv`) can improve performance and simplify debugging. In production, the application can utilize a more comprehensive dataset (`flights.csv`).

## Refactoring Application

We will introduce two profiles: `dev` for development and `prod` for production. Each profile will load the appropriate CSV file when initializing the `CSVDataLoader`.

## Update The Code

- **Define application.properties for Profiles**

Create an `application.properties` file and set the active profile using `spring.profiles.active=dev`. This file will dictate which profile to use at runtime.

- **Create Profile-Specific Properties Files**

Create two property files: `application-dev.properties` for development settings (e.g., point to `flights_small.csv`) and `application-prod.properties` for production settings (e.g., point to `flights.csv`).

- **Implement Profile-Based Bean Activation**

Use profile-based bean activation by creating a `DatabaseConnection` class. This class should be a Java Bean with a connection URL to database, username and password. Define different beans for this class that load different database configurations based on the active profile, with different connection URLs for development and production.

- **Define Beans Activated depending on Profile**

Add beans in a configuration class that are activated under specific profiles using `@Profile`. This will allow you to switch between different configurations seamlessly.

- **Update Data Loader to use DatabaseConnection**

Modify your DBDataLoader class to use the `DatabaseConnection` bean. It should not do anything except print the connection URL for demonstration purposes.

Test the application by activating different profiles and observing the data source changes.

## Changing Profiles with Command Line

You can change the active profile when starting your Spring Boot application using command line arguments (e.g., `--spring.profiles.active=prod` or `--spring.profiles.active=dev`).

## Switching Between CSV and DB DataLoader with Profiles

By utilizing Spring Profiles, you can easily switch between different data loading strategies. Annotate your data loader classes with `@Profile` to specify which implementation to use based on the active profile.

This flexibility allows you to adapt the application's behavior to different environments without changing the code.

## Benefits of Using Spring Profiles

- **Environment-Specific Configuration**: Easily switch between different configurations for development and production environments.
- **Improved Flexibility**: Tailor the application's behavior based on the active profile, enabling optimized performance and resource usage.
- **Simplified Management**: Manage properties for different environments in a clean and organized manner without altering code.

## Further Advices

Ensure that the profiles are properly activated in different deployment scenarios (e.g., through environment variables or command-line arguments). Regularly review and update profile-specific configurations to align with evolving application requirements.