import org.apache.camel.builder.RouteBuilder;

import org.apache.camel.main.Main;

import org.apache.camel.spi.PropertiesComponent;

import org.apache.camel.component.jackson.JacksonDataFormat;

public class DataProcessingRoute {

public static void main(String[] args) throws Exception {

Main main = new Main();

main.configure().addRoutesBuilder(new JdbcToApiRoute());

// Загрузка конфигурационного файла

PropertiesComponent pc = new PropertiesComponent();

pc.setLocation("classpath:application.properties");

main.getCamelContext().addComponent("properties", pc);

main.run(args);

}

static class JdbcToApiRoute extends RouteBuilder {

@Override

public void configure() throws Exception {

// Форматирование данных в JSON

JacksonDataFormat jsonFormat = new JacksonDataFormat(List.class);

// Настройка параметров подключения к базе данных из конфигурации

String jdbcUrl = "{{db.url}}";

String jdbcUsername = "{{db.username}}";

String jdbcPassword = "{{db.password}}";

String jdbcDriver = "{{db.driverClassName}}";

// Извлечение данных из базы данных

fromF("timer://runOnce?repeatCount=1")

.setBody(constant("SELECT \* FROM your\_table"))

.toF("jdbc:yourDataSource?dataSource.url=%s&dataSource.username=%s&dataSource.password=%s&dataSource.driverClassName=%s",

jdbcUrl, jdbcUsername, jdbcPassword, jdbcDriver)

.log("Data fetched from DB: ${body}")

// Преобразование данных в JSON перед отправкой

.marshal(jsonFormat)

.log("Data as JSON: ${body}")

// Отправка данных на сторонний API для фильтрации

.setHeader("Content-Type", constant("application/json"))

.to("http4://external-api-url.com/filter")

.log("Filtered data from API: ${body}")

// Дополнительная обработка или сохранение результата

.to("mock:result");

}

}

}

### **Объяснение:**

1. **Извлечение данных из базы данных**:
   * Сначала маршрут извлекает данные из базы данных с использованием JDBC-компонента Camel.
   * SQL-запрос "SELECT \* FROM your\_table" возвращает данные, которые сохраняются в теле сообщения в виде списка строк (List<Map<String, Object>>).
2. **Форматирование данных в JSON**:
   * Используем JacksonDataFormat для преобразования данных из списка строк в формат JSON, так как API обычно принимает данные в формате JSON.
   * marshal(jsonFormat) преобразует тело сообщения в JSON.
3. **Отправка данных в сторонний сервис по API**:
   * Устанавливаем заголовок Content-Type как application/json для указания, что данные передаются в формате JSON.
   * Используем компонент http4 для отправки POST-запроса на внешний API (http4://external-api-url.com/filter). Тело сообщения, содержащее JSON, отправляется в сервис.
   * API обрабатывает и фильтрует данные, затем возвращает отфильтрованный результат, который сохраняется в теле сообщения.
4. **Обработка ответа**:
   * Полученные отфильтрованные данные из API логируются и могут быть переданы на дальнейшую обработку или сохранены.
5. **Использование компонента mock для тестирования**:
   * В данном примере маршрут завершается передачей результата в компонент mock:result, который часто используется для тестирования. В реальном приложении вы, вероятно, захотите сохранить данные обратно в базу данных или выполнить другие действия.

import org.apache.camel.builder.RouteBuilder;

import org.apache.camel.main.Main;

import org.apache.camel.spi.PropertiesComponent;

import java.util.List;

import java.util.Map;

public class DataProcessingRoute {

public static void main(String[] args) throws Exception {

Main main = new Main();

main.configure().addRoutesBuilder(new JdbcToApiAndBackRoute());

// Загрузка конфигурационного файла

PropertiesComponent pc = new PropertiesComponent();

pc.setLocation("classpath:application.properties");

main.getCamelContext().addComponent("properties", pc);

main.run(args);

}

static class JdbcToApiAndBackRoute extends RouteBuilder {

@Override

public void configure() throws Exception {

// Настройка параметров подключения к базе данных из конфигурации

String jdbcUrl = "{{db.url}}";

String jdbcUsername = "{{db.username}}";

String jdbcPassword = "{{db.password}}";

String jdbcDriver = "{{db.driverClassName}}";

// Извлечение данных из базы данных

fromF("timer://runOnce?repeatCount=1")

.setBody(constant("SELECT \* FROM your\_table"))

.toF("jdbc:yourDataSource?dataSource.url=%s&dataSource.username=%s&dataSource.password=%s&dataSource.driverClassName=%s",

jdbcUrl, jdbcUsername, jdbcPassword, jdbcDriver)

.log("Data fetched from DB: ${body}")

// Передача данных на сторонний API для фильтрации

.setHeader("Content-Type", constant("application/x-java-serialized-object"))

.to("http4://external-api-url.com/filter")

.log("Filtered data from API: ${body}")

// Загрузка отфильтрованных данных обратно в базу данных

.process(exchange -> {

// Получение отфильтрованных данных из тела сообщения

List<Map<String, Object>> filteredData = exchange.getIn().getBody(List.class);

// Генерация SQL для вставки данных в целевую таблицу

String insertSql = "INSERT INTO your\_target\_table (column1, column2, ...) VALUES ";

StringBuilder values = new StringBuilder();

for (Map<String, Object> row : filteredData) {

// Создание строки вставки для каждой записи

values.append("(")

.append(row.get("column1")).append(", ")

.append(row.get("column2")).append(", ")

// добавьте остальные столбцы здесь

.append("),");

}

// Удаляем последнюю запятую

if (values.length() > 0) {

values.setLength(values.length() - 1);

}

insertSql += values.toString();

exchange.getIn().setBody(insertSql);

})

.toF("jdbc:yourDataSource?dataSource.url=%s&dataSource.username=%s&dataSource.password=%s&dataSource.driverClassName=%s",

jdbcUrl, jdbcUsername, jdbcPassword, jdbcDriver)

.log("Filtered data saved to DB");

}

}

}

### **Объяснение:**

1. **Извлечение данных из базы данных**:
   * Мы используем компонент jdbc для выполнения SQL-запроса, который извлекает данные из таблицы.
   * Данные сохраняются в теле сообщения в виде списка строк (List<Map<String, Object>>).
2. **Передача данных на внешний сервис**:
   * В данном случае данные передаются в виде сериализованного Java объекта с использованием заголовка Content-Type: application/x-java-serialized-object.
   * Компонент http4 отправляет данные на внешний API для фильтрации. API может работать с данными в виде объектов Java, если он поддерживает десериализацию.
3. **Получение отфильтрованных данных**:
   * После получения отфильтрованных данных от API они сохраняются в теле сообщения. Эти данные по-прежнему представляют собой список строк.
4. **Загрузка данных обратно в базу данных**:
   * В process шаге данные обрабатываются для создания SQL-запроса на вставку. Мы проходим по каждому ряду данных и генерируем SQL-запрос для вставки данных в целевую таблицу.
   * После создания SQL-запроса он передается обратно в компонент jdbc для выполнения вставки данных в базу.
5. **Логирование**:
   * Каждая стадия маршрута логирует промежуточные данные, чтобы отслеживать процесс.

### **Важно:**

* **Сериализация и десериализация**: В этом примере данные передаются в виде сериализованных объектов Java. Убедитесь, что сторонний API может корректно обрабатывать такие данные. В противном случае, потребуется другой формат передачи данных (например, XML, CSV и т.д.).
* **Защита данных**: Если данные чувствительные, подумайте о шифровании или использовании защищенного канала связи (например, HTTPS).

### **Заключение**

Этот маршрут позволяет извлечь данные из базы данных, передать их на сторонний сервис для фильтрации без преобразования в JSON, и затем загрузить отфильтрованные данные обратно в базу данных. Это полезно, когда необходимо работать с табличными данными и избегать дополнительного этапа сериализации/десериализации в формат JSON.