SPRING-JDBC-WORKING-02-25

1)
Mysql Table-Relationship ---

links –
https://medium.com/@magenta2127/how-to-design-mysql-database-model-for-1-to-1-1-to-n-and-m-to-n-relationship-fbedd434aeab
https://www.scaler.com/topics/uuid-mysql/

2)
Spring JDBC ---

links --
https://dimitri.codes/difference-spring-data-jdbc-jpa/#what-is-spring-jdbc
https://krishaniindrachapa.medium.com/get-results-from-join-queries-using-result-extractor-069afc4d792b

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/jdbc/core/namedparam/NamedParameterJdbcTemplate.html

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/jdbc/core/JdbcTemplate.html
https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/jdbc/core/simple/JdbcClient.html

To add Spring JDBC to your Spring Boot project, you include the following dependency:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
```

After that, you can use it by configuring the HikariCP DataSource through some properties.

```
spring.datasource.url=jdbc:h2:mem:
spring.datasource.hikari.maximum-pool-size=2
```

Behind the screens, Spring Boot will configure a HikariDataSource for you with the provided configuration. It will also create a *JdbcTemplate* and *NamedParameterJdbcTemplate* bean, and since Spring Boot 3.2, also a *JdbcClient* bean.

Using NamedParameterJdbcTemplate ---

```
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Repository;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

@Repository
@AllArgsConstructor
public class JdbcRepository {
  private final NamedParameterJdbcTemplate jdbcTemplate;
```

SELECT ---

Method :
<T> List<T> query(String sql, RowMapper<T> rowMapper)
Query given SQL to create a prepared statement from SQL, mapping each row to a Java object via a RowMapper.

```
List<Person> people = jdbcTemplate.query(
   "select name, firstname from person",
   (rs, rowNum) -> new Person(
      rs.getString("firstname"),
      rs.getString("name")
   ));

Or --

import org.springframework.jdbc.core.RowMapper;

public class PersonRowMapper implements RowMapper<Person> {
   @Override
   public Person mapRow(ResultSet rs, int rowNum) throws SQLException {
      return new Person(
         rs.getString("firstname"),
         rs.getString("name")
      );
   }
}

List<Person> people = jdbcTemplate.query(
   "select name, firstname from person",
   new PersonRowMapper() );
```

Method :
<T> List<T> query(String sql, SqlParameterSource paramSource, RowMapper<T> rowMapper)
Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, mapping each row to a Java object via a RowMapper.

Method :
<T> T queryForObject(String sql, SqlParameterSource paramSource, RowMapper<T> rowMapper)
Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, mapping a single result row to a Java object via a RowMapper.

```
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
import org.springframework.jdbc.core.namedparam.SqlParameterSource;

// SqlParameterSource parameters = new MapSqlParameterSource().addValue("id", 1L);

List<Person> people = jdbcTemplate.query(
   "select name, firstname from person where id = :id",
   new MapSqlParameterSource().addValue("id", 1L),  // parameters
   new PersonRowMapper());
```

Note: Since Spring Boot 3.2, the new JdbcClient is also autoconfigured for you. This new class allows you to query using a fluent API.

```
List<Person> people = jdbcClient
   .sql("select name, firstname from person where id = :id")
   .param("id", 1L)
   .query(new PersonRowMapper())
   .list();
```

Method:
<T> T query(String sql, ResultSetExtractor<T> rse)
Query given SQL to create a prepared statement from SQL, reading the ResultSet with a ResultSetExtractor.

Method:
<T> T query(String sql, SqlParameterSource paramSource, ResultSetExtractor<T> rse)
Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, reading the ResultSet with a ResultSetExtractor.

```
public Book getBookById(Integer bookId) {
   String sql = """
         SELECT b.id AS book_id, b.title, b.content, b.created_on, b.updated_on,
         r.id, r.name, r.email, r.comment, r.like_status, r.created_on AS r_created_on
         FROM books b LEFT JOIN reviews r ON b.id = r.book_id WHERE b.id = :bookId
         """;
   SqlParameterSource parameters = new MapSqlParameterSource().addValue("bookId", bookId);

   Book result = jdbcTemplate.query(sql, parameters, new ResultSetExtractor<Book>() {
         @Override
         public Book extractData(ResultSet rs) throws SQLException, DataAccessException {
            Book book = null;
            Map<String, Review> reviewMap = new HashMap<>();
            int row = 0;

            while (rs.next()) {
               if (book == null) {
                  book = new Book();
                  book.setId(rs.getInt("book_id"));
                  book.setTitle(rs.getString("title"));
                  book.setContent(rs.getString("content"));
                  book.setCreatedOn(convertToLocalDateTime(rs.getTimestamp("created_on")));
                  book.setUpdatedOn(convertToLocalDateTime(rs.getTimestamp("updated_on")));
               }
               if (rs.getString("id") != null){
                  Review review = new Review();
                  review.setBookId(rs.getInt("book_id"));
                  review.setId(rs.getInt("id"));
                  review.setName(rs.getString("name"));
                  review.setEmail(rs.getString("email"));
                  review.setComment(rs.getString("comment"));
                  review.setLikeStatus(LikeStatus.valueOf(rs.getString("like_status")));
                  review.setCreatedOn(convertToLocalDateTime(rs.getTimestamp("r_created_on")));
                  reviewMap.put(rs.getString("id"), review);
                  row++;
               }
            }

            if(book != null){
               List<Review> reviews = new ArrayList<>(reviewMap.values());
               book.setReviews(reviews);
            }

            return book;
         }
      });

      if(result == null) // ? Optional.empty() : Optional.of(result);
         throw new ResourceNotFoundException("book with ID="+bookId.toString()+" not found");
      return result;
}
```

Or --

```
public Book getBookById(Integer bookId) {
   String sql =
   SqlParameterSource parameters = new MapSqlParameterSource().addValue("bookId", bookId);

   Book result = jdbcTemplate.query(sql, parameters, new ResultSetExtractor<Book>() {

      @Override
      public Book extractData(ResultSet rs) throws SQLException, DataAccessException {
         Book book = null;
         Map<String, Review> reviewMap = new HashMap<>();
         int row = 0;
         while (rs.next()) {
            if (book == null) {
               book = bookMapper.mapRow(rs, row);
            }

            if (rs.getString("id") != null){
               Review review = reviewMapper.mapRow(rs, row);
               reviewMap.put(rs.getString("id"), review);
               row++;
            }
         }

         if(book != null){
            List<Review> reviews = new ArrayList<>(reviewMap.values());
            book.setReviews(reviews);
         }
         return book;

       }
    });

    if(result == null) // ? Optional.empty() : Optional.of(result);
      throw new ResourceNotFoundException("book with ID="+bookId.toString()+" not found");

    return result;
}

private final RowMapper<Book> bookMapper = (rs, rowNum) -> {
      Book book = new Book();
      book.setId(rs.getInt("book_id"));
      book.setTitle(rs.getString("title"));
      book.setContent(rs.getString("content"));
      book.setCreatedOn(convertToLocalDateTime(rs.getTimestamp("created_on")));
      book.setUpdatedOn(convertToLocalDateTime(rs.getTimestamp("updated_on")));
      return book;
   };

   private final RowMapper<Review> reviewMapper = (rs, rowNum) -> {
      Review review = new Review();
      review.setBookId(rs.getInt("book_id"));
      review.setId(rs.getInt("id"));
      review.setName(rs.getString("name"));
      review.setLikeStatus(LikeStatus.valueOf(rs.getString("like_status")));
      review.setCreatedOn(convertToLocalDateTime(rs.getTimestamp("r_created_on")));
      return review;
   };
```

```
    private LocalDateTime convertToLocalDateTime(Timestamp tst) {
      if (tst == null) {
        return null;
      } else {
        return tst.toLocalDateTime();
      }
    }
```

Or --

```
public class BookMapExtractor implements ResultSetExtractor<Book> {
  @Override
  public Book extractData(ResultSet rs) throws SQLException, DataAccessException {
    ...
    return book;
  }
  ...
}

public Book getBookById(Integer bookId) {
  ...
  Book result = jdbcTemplate.query(sql, parameters, new BookMapExtractor());
  ...
  return result;
}
```

INSERT, UPDATE, DELETE ---

Method:
int update(String sql, SqlParameterSource paramSource)
Issue an update via a prepared statement, binding the given arguments.

Method:
int update(String sql, SqlParameterSource paramSource, KeyHolder generatedKeyHolder)
Issue an update via a prepared statement, binding the given arguments, returning generated keys.

```
import org.springframework.jdbc.support.GeneratedKeyHolder;
import org.springframework.jdbc.support.KeyHolder;

@Transactional
public Book saveBook(Book book) {
  String sql = "INSERT INTO books (title, content, created_on) VALUES (:title, :content, :created_on)";
  SqlParameterSource parameters = new MapSqlParameterSource()
    .addValue("title", book.getTitle())
    .addValue("content", book.getContent())
    .addValue("created_on", Timestamp.valueOf(LocalDateTime.now()));
  KeyHolder generatedKeyHolder = new GeneratedKeyHolder();

  jdbcTemplate.update(sql, parameters, generatedKeyHolder);  // return > 0 if ok

  Number key = generatedKeyHolder.getKey();
  return getBookById(key.intValue());
}

public boolean deleteBook(Integer bookId) {
  String sql = "DELETE FROM books WHERE id = :bookId";
  SqlParameterSource parameters = new MapSqlParameterSource().addValue("bookId", bookId);
  return jdbcTemplate.update(sql, parameters) > 0;
}
```

3) Project

Links ---
https://github.com/hong1234/Spring-JDBC-N-M-Relation
https://github.com/hong1234/Spring-JDBC-UUID-MySQL
https://github.com/hong1234/spring-boot3-mvc-jdbc-restApi