

MySQL Week 4 Exercises

Background

We have been developing a menu-driven application that demonstrates how to perform CRUD (Create, Read, Update, Read) operations on a DIY project database. Thus far, we have learned how to create a connection to a MySQL database and how to insert records into a table. In this section, we will apply our knowledge of querying data to list all projects without project details, and to list a single project with all details.


Objectives

In these exercises, you will expand the menu application to list all projects (name and ID). Then, you will write code to select a project to edit. This will involve returning a selected project along with all project details. This will further our pursuit of implementing CRUD operations on database tables.

In these exercises, you will:

- Hone your SQL query skills by writing SQL statements to fetch a `List of Project` records.
- Learn how to perform multiple queries in a single transaction.
- Write an inner join to fetch `category` rows related to a `project` row.
- Use an `Optional` to either return a `project` record or to throw a custom `Exception`.
- Practice writing Lambda expressions both to list the projects and to throw a custom `Exception` from an `Optional`.

Important

In the exercises below, you will see this icon: . This means to take a screen shot or snip showing the results of the action or the code in the editor.

Also important: you should take the variable names and method names as suggestions. They're good suggestions, but if you want to deviate from them, feel free to do so. However, don't go crazy and change `listProjects()` to `emptyMyBankAccountByBuyingAJeep()`. You should follow Java best practices. Method names should describe what the method does in the interest of self-documentation.

Exercises

Complete these exercises as directed. If you get hopelessly stuck, please see the "Solutions" section below.

In these exercises, you will often be told to call a method prior to creating it. This is a good approach. You set up the return type by assigning it to a variable and setting up the parameters. Then, Eclipse can correctly create the method.

List projects

In this section, you will add code to return and print a list of projects. Several application methods will call this method to let the user select a project from a list. To get the list of projects, you will add the menu option and method in the menu class, then you will add a method in the service class. The DAO class will perform the actual work of fetching the list using JDBC method calls.

Modifications to menu app

In this section you will add another option to the list of available options. Then you will add another case to the switch method along with the method to call the service to retrieve the list of projects.

In this section, you will be working in `ProjectsApp.java`.

1. Add this line to the list of operations at the top of `ProjectsApp.java`: `"2) List projects"`.
2. Add case 2 to the switch statement in `processUserSelection()`. In the case, call method `listProjects()`. Don't forget to add the break statement.
3. Have Eclipse create the method `listProjects()`. It should take no parameters and should return nothing. In the method:
 - a. Create a variable to hold a List of Projects named `projects`. Assign the variable the results of a method call to `projectService.fetchAllProjects()`.
 - b. Print `"\nProjects:"` (without quotes) to the console.
 - c. For each `Project`, print the ID and name separated by `": "`. Indent each line with a couple of spaces.
 - d. At this point, the method should look like this:

```
private void listProjects() {
    List<Project> projects = projectService.fetchAllProjects();

    System.out.println("\nProjects:");

    projects.forEach(project -> System.out
        .println("    " + project.getProjectId()
            + ": " + project.getProjectName()));
}
```
 - e. Have Eclipse create the method `fetchAllProjects()` in the `ProjectService` class, or create it yourself.
 - f. Save all files. At this point the project should have no errors.

Modifications to project service

You need to fill in the method in the service class to call the DAO class. This method will simply return the results of the method call to the DAO class. The service class in our small application does not do very much. But it allows us to properly separate concerns of input/output, business logic, and database reads and writes. If you always structure your code like this it will be much easier to understand and make changes if needed.

In this section, you will be working in `ProjectService.java`.

1. In the method `fetchAllProjects`, call the `fetchAllProjects()` method on the `projectDao` object.
2. Have Eclipse create the method `fetchAllProjects()` in `ProjectDao.java` or create it yourself. It takes no parameters and returns a `List of Projects`.

Modifications to project dao

Now you need to write the code to retrieve all the projects from the database. It is structured similarly to the `insertProject()` method, but it will also incorporate a `ResultSet` to retrieve the project row(s).

To implement this method, first you will write the SQL statement that instructs MySQL to return all project rows without any materials, steps, or categories. Then, you will obtain a `Connection` and start a transaction. Next, you will obtain a `PreparedStatement` from the `Connection` object. Then, you will get a `ResultSet` from the `PreparedStatement`. Finally, you will iterate over the `ResultSet` to create a `Project` object for each row returned.

In this section, you will be working in `ProjectDao.java`.

1. In the method `fetchAllProjects()`:
 - a. Write the SQL statement to return all projects not including materials, steps, or categories. Order the results by project name.
 - b. Add a `try-with-resource` statement to obtain the `Connection` object. Catch the `SQLException` in a `catch` block and rethrow a new `DbException`, passing in the `SQLException` object.
 - c. Inside the `try` block, start a new transaction.
 - d. Add an inner `try-with-resource` statement to obtain the `PreparedStatement` from the `Connection` object. In a `catch` block, catch an `Exception` object. Rollback the transaction and throw a new `DbException`, passing in the `Exception` object as the cause.
 - e. Inside the (currently) innermost `try-with-resource` statement, add a `try-with-resource` statement to obtain a `ResultSet` from the `PreparedStatement`. Include the import statement for `ResultSet`. It is in the `java.sql` package.
 - f. Inside the new innermost `try-with-resource`, create and return a `List of Projects`.
 - g. Loop through the result set. Create and assign each result row to a new `Project` object. Add the `Project` object to the `List of Projects`. You can do this by calling the `extract` method:

```
while(rs.next()) {
    projects.add(extract(rs, Project.class));
}
```

or by doing it manually like this:

```
while(rs.next()) {
    Project project = new Project();

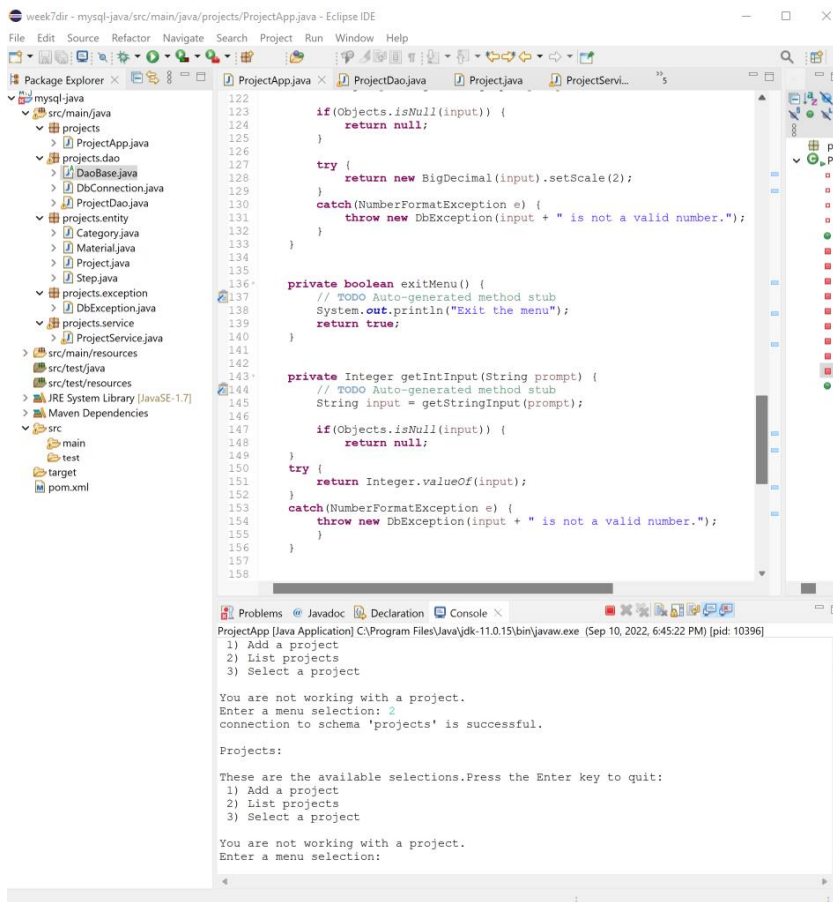
    project.setActualHours(rs.getBigDecimal("actual_hours"));
    project.setDifficulty(rs.getObject("difficulty", Integer.class));
    project.setEstimatedHours(rs.getBigDecimal("estimated_hours"));
    project.setNotes(rs.getString("notes"));
    project.setProjectId(rs.getObject("project_id", Integer.class));
    project.setProjectName(rs.getString("project_name"));

    projects.add(project);
}
```

Test it

Test your solution by running `ProjectsApp`. Select "List projects". The app should return a list of projects that you have created. Make sure that you have created at least one project. Take a screen shot

showing the console with the menu selections, your input, and the listed project(s).



Select a project

In this section you will write code to select a current project. With a current project selected, you will be able to add materials, steps, and categories in future exercises. This will involve more code than it sounds like on the surface. When you select a current project using the project ID, you will query the project tables to fetch all project details (materials, steps, and categories) within a single transaction.

Modifications to menu app

In this section, you will add a new option to the selections available to the user. Then you will add a method to select the current project. After selecting the current project, you will modify the `printOperations()` method to display the currently selected project, if any. It will print all project details including materials, steps, and categories.

In this section you will be working in `ProjectsApp.java`.

1. Add an instance variable of type `Project` named `curProject`.
2. Add a new operation: "3) Select a project".
3. Add a case to the `switch` to handle the operation. Call method `selectProject()`.
4. In this step you will create the method, `selectProject()`. This method will list the project IDs and names so that the user can select a project ID. Once the ID is entered, the service is called to return the project details. If successful, the current project is set to the returned project. Follow these instructions to write the method.

Add a new method named `selectProject()`. It takes no parameters and returns nothing.

- a. Call `listProjects()` to print a List of Projects.
- b. Collect a project ID from the user and assign it to an Integer variable named `projectId`. Prompt the user with "Enter a project ID to select a project".
- c. Set the instance variable `curProject` to `null` to unselect any currently selected project. This is done in case the call to the service results in an exception being thrown. Rather than leave the current project selected in that case, it is unselected first.
- d. Call a new method, `fetchProjectById()` on the `projectService` object. The method should take a single parameter, the project ID input by the user. It should return a `Project` object. Assign the returned `Project` object to the instance variable `curProject`. Note that if an invalid project ID is entered, `projectService.fetchProjectById()` will throw a `NoSuchElementException`, which is handled by the `catch` block in `processUserSelections()`.
- e. At the end of the method, add a check to see if `curProject` is null. If so, print "Invalid project ID selected." on the console.
- f. The method should look like this:

```

private void selectProject() {
    listProjects();
    Integer projectId = getIntInput("Enter a project ID to select a project");

    /* Unselect the current project. */
    curProject = null;

    /* This will throw an exception if an invalid project ID is entered. */
    curProject = projectService.fetchProjectById(projectId);
}

```

5. In this step, you will add code to print the current project when the available menu selections are displayed to the user. To do this, find the method `printOperations()`. At the bottom of method `printOperations()`, check if `curProject` is null. If null, print a message: `"\nYou are not working with a project."`. Otherwise, print the message: `"\nYou are working with project: " + curProject`.

```

if(Objects.isNull(curProject)) {
    System.out.println("\nYou are not working with a project.");
}
else {
    System.out.println("\nYou are working with project: " + curProject);
}

```

Modifications to project service

In this section you will create a method in the project service that will call the DAO to retrieve a single Project object with all details, including materials, steps, and categories. This method will throw an exception if the project with the given ID does not exist.

Note that you will temporarily assign the results of a method call to the DAO to an `Optional<Project>` object. This will cause Eclipse to create the return type on the DAO method as `Optional<Project>`. Once the method has been created, you can delete the assignment and return the `Project`, if successful. If not successful, the method will throw a `NoSuchElementException`.

In this section you will be working in `ProjectService.java`.

1. Create method `fetchProjectById()`. It returns a `Project` object and takes an `Integer` `projectId` as a parameter. Inside the method:

- a. Temporarily assign a variable of type `Optional<Project>` to the results of calling `projectDao.fetchProjectById()`. Pass the project ID to the method.

```
Optional<Project> op = projectDao.fetchProjectById(projectId);
```

This temporary assignment will cause Eclipse to create the correct return value (`Optional<Project>`) in `ProjectService.java`.

- b. Let Eclipse create the method for you in the `ProjectDao` class. The editor will display `ProjectDao.java`. Return to `ProjectService.java`. Save all files.
- c. Replace the variable and assignment with a `return` statement. This will cause a compilation error, which you will correct next.

- d. Add a method call to `.orElseThrow()` just inside the semicolon at the end of the method call to `projectDao.fetchProjectById()`. Use a zero-argument Lambda expression inside the call to `.orElseThrow()` to create and return a new `NoSuchElementException` with the custom message, "Project with project ID=" + `projectId` + " does not exist.". The method should look like this:

```
public Project fetchProjectById(Integer projectId) {  
    return projectDao.fetchProjectById(projectId).orElseThrow(  
        () -> new NoSuchElementException(  
            "Project with project ID=" + projectId  
            + " does not exist."));  
}
```

Save all files. At this point there should be no compilation errors.

Modifications to project dao

In this section you will write the code that will retrieve a project row and all associated child rows: materials, steps, and categories. The method will start with the usual `try-with-resource` statement to obtain the `Connection`. Then you will add a `try/catch` before obtaining the `PreparedStatement`. This is done so that after obtaining the project details, the materials, steps, and categories can be retrieved within the same transaction.

If you get stuck and don't understand the instructions, please refer to the "Solutions" section at the end of this assignment.

In this section you will be working in `ProjectDao.java`.

1. In the method `fetchProjectById()`:
 - a. Write the SQL statement to return all columns from the project table in the row that matches the given `projectId`. Make sure to use the parameter placeholder "?" in the SQL statement.
 - b. Obtain a `Connection` object in a `try-with-resource` statement. Add the `catch` block to handle the `SQLException`. In the `catch` block throw a new `DbException` passing the `SQLException` object as a parameter.
 - c. Start a transaction inside the `try-with-resource` statement.
 - d. Below the method call to `startTransaction()`, add an inner `try/catch`. The `catch` block should handle `Exception`. Inside the `catch` block, rollback the transaction and throw a new `DbException` that takes the `Exception` object as a parameter.
 - e. Inside the `try` block, create a variable of type `Project` and set it to null. Return the `Project` object as an `Optional` object using `Optional.ofNullable()`. Save the file. You should have no compilation errors at this point but you may see some warnings. This is OK. Here is the method at this point.

```

public Optional<Project> fetchProjectById(Integer projectId) {
    String sql = "SELECT * FROM " + PROJECT_TABLE + " WHERE project_id = ?";

    try(Connection conn = DbConnection.getConnection()) {
        startTransaction(conn);

        try {
            Project project = null;

            return Optional.ofNullable(project);
        }
        catch(Exception e) {
            rollbackTransaction(conn);
            throw new DbException(e);
        }
    }
    catch(SQLException e) {
        throw new DbException(e);
    }
}

```

- f. Inside the inner try block, obtain a PreparedStatement from the Connection object in a try-with-resource statement. Pass the SQL statement in the method call to preparedStatement(). Add the projectId method parameter as a parameter to the PreparedStatement.
- g. Obtain a ResultSet in a try-with-resource statement. If the ResultSet has a row in it (rs.next()) set the Project variable to a new Project object and set all fields from values in the ResultSet. You can call the extract() method for this.
- h. Below the try-with-resource statement that obtains the PreparedStatement but inside the try block that manages the rollback, add three method calls to obtain the list of materials, steps, and categories. Since each method returns a List of the appropriate type, you can call addAll() to add the entire List to the List in the Project object:

```

project.getMaterials().addAll(fetchMaterialsForProject(conn, projectId));

```

- i. Commit the transaction. Here's what the method should look like now:


```

public Optional<Project> fetchProjectById(Integer projectId) {
    String sql = "SELECT * FROM " + PROJECT_TABLE + " WHERE project_id = ?";

    try(Connection conn = DbConnection.getConnection()) {
        startTransaction(conn);

        try {
            Project project = null;

            try(PreparedStatement stmt = conn.prepareStatement(sql)) {
                setParameter(stmt, 1, projectId, Integer.class);

                try(ResultSet rs = stmt.executeQuery()) {
                    if(rs.next()) {
                        project = extract(rs, Project.class);
                    }
                }
            }

            if(Objects.nonNull(project)) {
                project.getMaterials().addAll(fetchMaterialsForProject(conn, projectId));
                project.getSteps().addAll(fetchStepsForProject(conn, projectId));
                project.getCategories().addAll(fetchCategoriesForProject(conn, projectId));
            }

            commitTransaction(conn);

            return Optional.ofNullable(project);
        }
        catch(Exception e) {
            rollbackTransaction(conn);
            throw new DbException(e);
        }
    }
    catch(SQLException e) {
        throw new DbException(e);
    }
}

```

2. In this step you will write the methods that will return materials, steps, and categories as Lists. Each method is structured similarly. Since the Connection object is passed into each method, you won't have to obtain the Connection from DbConnection.getConnection().

Also, you won't need to add catch blocks to the try-with-resource statements because the caller makes the method calls within a try block. It won't hurt to catch the SQLException and turn it into an unchecked exception as you have been doing. But it won't hurt to simply declare the exception in the method signature either. It's your choice. So, this:

```

private List<Material> fetchMaterialsForProject(Connection conn,
    Integer projectId) throws SQLException {

```

versus this:

Add the "throws" declaration

```
try(PreparedStatement stmt = conn.prepareStatement(sql)) {
}
catch(SQLException e) {
    throw new DbException(e);
}
```

Or catch the Exception

Follow these instructions to write the three methods to return materials, steps, and categories. Each method should return a `List` of the appropriate type. At this point there should be no compilation errors.

- Each method should take the `Connection` and the project ID as parameters.
- Each method should return a `List` of the appropriate type (i.e., `List<Material>`).
- Each method is written in the same way as the other query methods with the exception that the `Connection` is passed as a parameter, so you don't need to call `DbConnection.getConnection()` to obtain it.
- Each method can add `throws SQLException` to the method declaration. This is because the method call to each method is within a `try/catch` block.
- Here is a sample method (all three methods should have the identical structure). However, when you fetch the categories, you will need to join with the `project_category` join table as shown below.

```
private List<Category> fetchCategoriesForProject(Connection conn,
    Integer projectId) throws SQLException {
    // @formatter:off
    String sql = "
        + "SELECT c.* FROM " + CATEGORY_TABLE + " c "
        + "JOIN " + PROJECT_CATEGORY_TABLE + " pc USING (category_id) "
        + "WHERE project_id = ?";
    // @formatter:on

    try(PreparedStatement stmt = conn.prepareStatement(sql)) {
        setParameter(stmt, 1, projectId, Integer.class);

        try(ResultSet rs = stmt.executeQuery()) {
            List<Category> categories = new LinkedList<>();

            while(rs.next()) {
                categories.add(extract(rs, Category.class));
            }

            return categories;
        }
    }
}
```

Test it

Now it's time to test that the code works. Since you haven't written the code to add materials, steps, and categories yet, you will have to add some rows manually.

Instructions for DBeaver

1. Create a connection to the projects schema in DBeaver if you haven't already.
2. Right-click on the connection name and select "SQL Editor" / "Recent SQL Script".

Instructions for MySQL CLI

1. Start up MySQL CLI. Enter the root password.
2. Type "use projects;" (without the quotes).

The test

1. Add one or more categories. You don't have to enter the category ID, MySQL will manage that for you.

```
INSERT INTO category (category_name) VALUES ('Doors and Windows');
```

2. Make sure you have added one or more projects. In the editor type this to find a valid project_id:

```
SELECT * FROM project;
```

3. Add one or more material records. If your project_id is 1, enter something like this:

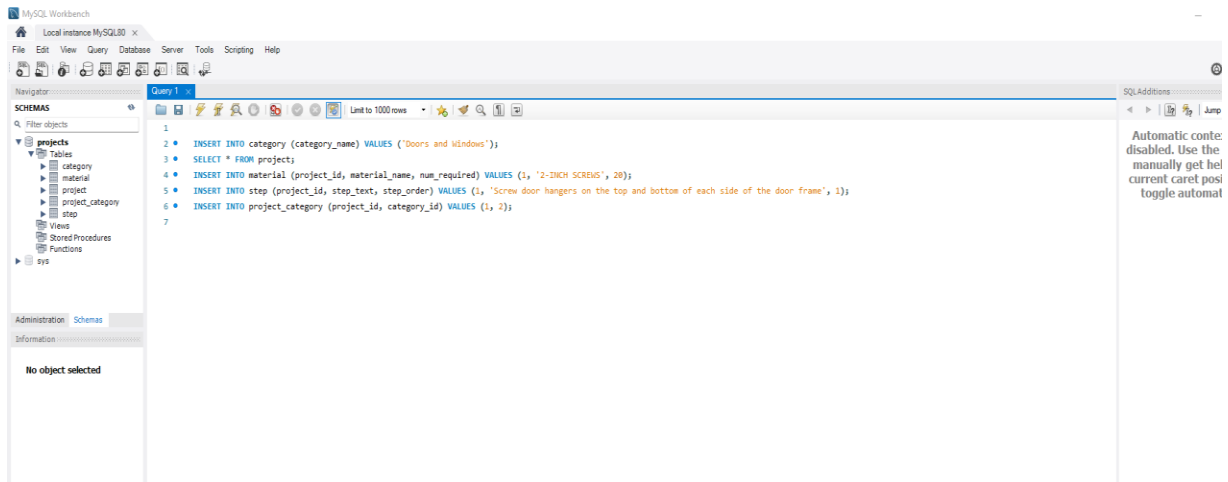
```
INSERT INTO material (project_id, material_name, num_required)
VALUES
(1, '2-inch screws', 20);
```

4. Add one or more step records. If your project_id is 1, enter something like the following:

```
INSERT INTO step (project_id, step_text, step_order)
VALUES
(1, 'Screw door hangers on the top and bottom of each side of the door
frame', 1);
```


5. Add one or more project_category records. This is a join table that contains two foreign keys. One foreign key points to a project row and the other points to a category row. So, if your project ID is 1 and the category ID for 'Doors and Windows' is 2, enter the join row like this:

```
INSERT INTO project_category (project_id, category_id)
VALUES
(1, 2);
```



GitHub Repo link

<https://github.com/shegyes/week7assignent>

6. Run `ProjectsApp` as a Java application. Enter "3" to select a project. Enter a project ID. Take a screen shot showing that the project is selected.  It should look something like this:

These are the available selections. Press the Enter key to quit:

- 1) Add a project
- 2) List projects
- 3) Select a project

You are not working with a project.

Enter a menu selection: 3

Connection to schema 'projects' is successful.

Projects:

- 1: Hang a door

Enter a project ID to select a project: 1

Connection to schema 'projects' is successful.

These are the available selections. Press the Enter key to quit:

- 1) Add a project
- 2) List projects
- 3) Select a project

You are working with project:

ID=1

name=Hang a door

estimatedHours=4.00

actualHours=3.00

difficulty=3

notes=Use the door hangers from Home Depot

Materials:

ID=1, materialName=Door in frame, numRequired=1, cost=null

ID=2, materialName=Package of door hangers from Home Depot, numRequired=1, cost=null

ID=3, materialName=2-inch screws, numRequired=20, cost=null

Steps:

ID=1, stepText=Align hangers on opening side of door vertically on the wall

ID=2, stepText=Screw hangers into frame

Categories:

ID=1, categoryName=Doors and Windows

ID=2, categoryName=Repairs

Enter a menu selection:


You should see project details, a list of materials, a list of steps, and a list of categories. If you do not get a result like that shown above, check the console for errors. If you can't figure it out there are two things you can try:

- In the catch block in method `processUserSelection()`, print the entire stack trace of the exception like this:

```
catch(Exception e) {  
    System.out.println("\nError: " + e + " Try again.");  
    e.printStackTrace();  
}
```

- Start the application in debug mode. Load `ProjectsApp.java` into the editor. Right-click in editor and select "Debug As" / "Java Application". Set breakpoints as appropriate. Work through the application until you find the error.
- This article on debugging is a little dated but still applicable. You don't need to worry about the section on remote debugging.

https://www.eclipse.org/community/eclipse_newsletter/2017/june/article1.php

7. Now test with an invalid project ID. Run the application. Enter "3" to select a project. Enter an invalid number. Take a screen shot of the console.  It should look something like the screen shot below.

These are the available selections. Press the Enter key to quit:

- 1) Add a project
- 2) List projects
- 3) Select a project

You are not working with a project.

Enter a menu selection: 3

Connection to schema 'projects' is successful.

Projects:

- 1: Hang a door

Enter a project ID to select a project: 99

Connection to schema 'projects' is successful.

Error: [projects.exception.DbException](#): Project with project ID=99 does not exist. Try again.

These are the available selections. Press the Enter key to quit:

- 1) Add a project
- 2) List projects
- 3) Select a project

You are not working with a project.

Enter a menu selection:

Exiting the menu.

Here's the error

Solutions

These solutions are provided as a reference. Please work through the exercises on your own as best you can.

ProjectsApp.java

These screen shots do not contain the entire Java file. Only the parts changed since the prior exercises are shown.

```
public class ProjectsApp {
    private Scanner scanner = new Scanner(System.in);
    private ProjectService projectService = new ProjectService();
    private Project curProject;

    // @formatter:off
    private List<String> operations = List.of(
        "1) Add a project",
        "2) List projects",
        "3) Select a project"
    );
    // @formatter:on
}
```

```

private void processUserSelections() {
    boolean done = false;

    while(!done) {
        try {
            int selection = getUserSelection();

            switch(selection) {
                case -1:
                    done = exitMenu();
                    break;

                case 1:
                    createProject();
                    break;

                case 2:
                    listProjects();
                    break;

                case 3:
                    selectProject();
                    break;

                default:
                    System.out.println("\n" + selection + " is not a valid selection. Try again.");
                    break;
            }
        }
        catch(Exception e) {
            System.out.println("\nError: " + e + " Try again.");
        }
    }
}

/**
 *
 */
private void selectProject() {
    listProjects();
    Integer projectId = getIntInput("Enter a project ID to select a project");

    /* Unselect the current project. */
    curProject = null;

    /* This will throw an exception if an invalid project ID is entered. */
    curProject = projectService.fetchProjectById(projectId);
}

/**
 *
 */
private void listProjects() {
    List<Project> projects = projectService.fetchAllProjects();

    System.out.println("\nProjects:");

    projects.forEach(project -> System.out
        .println("    " + project.getProjectId() + ": " + project.getProjectName()));
}

```

```

/**
 * Print the menu selections, one per line.
 */
private void printOperations() {
    System.out.println("\nThese are the available selections. Press the Enter key to quit:");

    /* With Lambda expression */
    operations.forEach(line -> System.out.println(" " + line));

    /* With enhanced for loop */
    // for(String line : operations) {
    // System.out.println(" " + line);
    // }

    if(Objects.isNull(curProject)) {
        System.out.println("\nYou are not working with a project.");
    }
    else {
        System.out.println("\nYou are working with project: " + curProject);
    }
}

```

ProjectService.java

These screen shots do not contain the entire Java file. Only the parts changed since the prior exercises are shown.

```

/**
 * This method calls the project DAO to retrieve all project rows without accompanying details
 * (materials, steps and categories).
 *
 * @return A list of project records.
 */
public List<Project> fetchAllProjects() {
    return projectDao.fetchAllProjects();
}

/**
 * This method calls the project DAO to get all project details, including materials, steps, and
 * categories. If the project ID is invalid, it throws an exception.
 *
 * @param projectId The project ID.
 * @return A Project object if successful.
 * @throws NoSuchElementException Thrown if the project with the given ID does not exist.
 */
public Project fetchProjectById(Integer projectId) {
    return projectDao.fetchProjectById(projectId).orElseThrow(() -> new NoSuchElementException(
        "Project with project ID=" + projectId + " does not exist."));
}

```


ProjectDao.java

These screen shots do not contain the entire Java file. Only the parts changed since the prior exercises are shown.

```
public List<Project> fetchAllProjects() {
    String sql = "SELECT * FROM " + PROJECT_TABLE + " ORDER BY project_name";

    try(Connection conn = DbConnection.getConnection()) {
        startTransaction(conn);

        try(PreparedStatement stmt = conn.prepareStatement(sql)) {
            try(ResultSet rs = stmt.executeQuery()) {
                List<Project> projects = new LinkedList<>();

                while(rs.next()) {
                    projects.add(extract(rs, Project.class));

                    /* Alternative approach */
                    // Project project = new Project();
                    //
                    // project.setActualHours(rs.getBigDecimal("actual_hours"));
                    // project.setDifficulty(rs.getObject("difficulty", Integer.class));
                    // project.setEstimatedHours(rs.getBigDecimal("estimated_hours"));
                    // project.setNotes(rs.getString("notes"));
                    // project.setProjectId(rs.getObject("project_id", Integer.class));
                    // project.setProjectName(rs.getString("project_name"));
                    //
                    // projects.add(project);
                }

                return projects;
            }
        }
    } catch(Exception e) {
        rollbackTransaction(conn);
        throw new DbException(e);
    }
} catch(SQLException e) {
    throw new DbException(e);
}
```

```

public Optional<Project> fetchProjectById(Integer projectId) {
    String sql = "SELECT * FROM " + PROJECT_TABLE + " WHERE project_id = ?";

    try(Connection conn = DbConnection.getConnection()) {
        startTransaction(conn);

        try {
            Project project = null;

            try(PreparedStatement stmt = conn.prepareStatement(sql)) {
                setParameter(stmt, 1, projectId, Integer.class);

                try(ResultSet rs = stmt.executeQuery()) {
                    if(rs.next()) {
                        project = extract(rs, Project.class);
                    }
                }
            }

            if(Objects.nonNull(project)) {
                project.getMaterials().addAll(fetchMaterialsForProject(conn, projectId));
                project.getSteps().addAll(fetchStepsForProject(conn, projectId));
                project.getCategories().addAll(fetchCategoriesForProject(conn, projectId));
            }

            commitTransaction(conn);
            return Optional.ofNullable(project);
        }
        catch(Exception e) {
            rollbackTransaction(conn);
            throw new DbException(e);
        }
    }
    catch(SQLException e) {
        throw new DbException(e);
    }
}

private List<Category> fetchCategoriesForProject(Connection conn,
    Integer projectId) throws SQLException {
    // @formatter:off
    String sql = ""
        + "SELECT c.* FROM " + CATEGORY_TABLE + " c "
        + "JOIN " + PROJECT_CATEGORY_TABLE + " pc USING (category_id) "
        + "WHERE project_id = ?";
    // @formatter:on

    try(PreparedStatement stmt = conn.prepareStatement(sql)) {
        setParameter(stmt, 1, projectId, Integer.class);

        try(ResultSet rs = stmt.executeQuery()) {
            List<Category> categories = new LinkedList<>();

            while(rs.next()) {
                categories.add(extract(rs, Category.class));
            }

            return categories;
        }
    }
}

```

```

private List<Step> fetchStepsForProject(Connection conn, Integer projectId) throws SQLException {
    String sql = "SELECT * FROM " + STEP_TABLE + " WHERE project_id = ?";

    try(PreparedStatement stmt = conn.prepareStatement(sql)) {
        setParameter(stmt, 1, projectId, Integer.class);

        try(ResultSet rs = stmt.executeQuery()) {
            List<Step> steps = new LinkedList<>();

            while(rs.next()) {
                steps.add(extract(rs, Step.class));
            }

            return steps;
        }
    }
}

private List<Material> fetchMaterialsForProject(Connection conn, Integer projectId)
    throws SQLException {
    String sql = "SELECT * FROM " + MATERIAL_TABLE + " WHERE project_id = ?";

    try(PreparedStatement stmt = conn.prepareStatement(sql)) {
        setParameter(stmt, 1, projectId, Integer.class);

        try(ResultSet rs = stmt.executeQuery()) {
            List<Material> materials = new LinkedList<>();

            while(rs.next()) {
                materials.add(extract(rs, Material.class));
            }

            return materials;
        }
    }
}

```