MySQL Week 11 Exercises

Background

We have been developing a menu-driven application that demonstrates how to perform CRUD (Create, Read, Update, and Delete) operations on a project database. Thus far, we have learned how to create a connection to a MySQL database and how to insert records into a table. Then, we learned how to query for a list of records and for all details on a single record. In these exercises, we will learn the final two parts of CRUD: Updating and Deleting.

Objectives

In these exercises, you will:

- Modify project details using the UPDATE statement.
- Delete a project and all child rows using the DELETE statement.
- Observe that using ON DELETE CASCADE automatically deletes child rows with a foreign key relationship.
- Use the return value from `PreparedStatement.executeUpdate()` to determine if a row was updated or deleted.

Important

In the exercises below, you will see this icon: .

This means to make sure that you include this functionality in your video showcase.

MySQL Week 11 Exercises

Instructions

URL to GitHub Repository: https://github.com/Jeffrweinstein/Week7-11

URL to Public Link of your Video: https://www.youtube.com/watch?v=4xtnXCOR_Q4

Instructions :

1. Follow the **Exercises** below to complete this assignment.
    - In Eclipse, or an IDE of your choice, write the code that accomplishes the objectives listed below. Ensure that the code compiles and runs as directed.
    - Create a new repository on GitHub for this week's assignment and push your completed code to this dedicated repo, including your entire Maven Project Directory (e.g., mysql-java) and any .sql files that you create. In addition, screenshot your ERD and push the screenshot to your GitHub repo.
    - Include the functionality into your Video when you see: 📷
    - Create a video showcasing your work:
        - In this video: record and present your project verbally while showing the results of the working project. Don't forget to include the requested functionality, indicated by: 📷
        - Easy way to Create a video: Start a meeting in Zoom, share your screen, open Eclipse with the code and your Console window, start recording & record yourself describing and running the program showing the results.
        - Your video should be a maximum of 5 minutes.
        - Upload your video with a public link.
        - Easy way to Create a Public Video Link: Upload your video recording to YouTube with a public link.
2. In addition, please include the following in your Coding Assignment Document:
    - The URL for this week's GitHub repository.
    - The URL of the public link of your video.
3. Save the Coding Assignment Document as a .pdf and do the following:
    - Push the .pdf to the GitHub repo for this week.
    - Upload the .pdf to the LMS in your Coding Assignment Submission.

MySQL Week 11 Exercises

Exercises

In these exercises, you will modify project contents and delete a project. You have already learned how to perform the Create and Read part of CRUD operations. This will complete your CRUD experience by adding Update and Delete.

You should try to follow the instructions as best you can. Suggestions for variable and method names are given – you can take those suggestions or not as you wish. If you deviate from the instructions, try to stick to Java best practices by naming methods and variables for what they do or what they are. If you get stuck, see the Solutions section at the end of this document.

Update project details

In this section, you will update a project row. There is a lot remaining to be done for an industrious student: adding materials, steps, and categories, maintaining categories; modifying materials and steps; changing step order, etc. In this section, you will gain part of that skill set.

Follow these steps to update the project details.

Changes to the menu application

In this section, you will make changes to the menu application to allow the user to update project details. You will add a new menu selection and add a method call in the `switch` statement. Finally, you will create a method to get project detail changes from the user and call the project service to make the modifications.

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

1. Add the line `"4) Update project details"` to the list of operations.
2. Add `case 4` to the `switch` statement and call method `updateProjectDetails()`. Let Eclipse create the method for you.
3. In method `updateProjectDetails()`:
    a. Check to see if `curProject` is `null`. If so, print a message `"\nPlease select a project."` and return from the method.
    b. For each field in the `Project` object, print a message along with the current setting in `curProject`. Here is an example:

```
String projectName =
    getStringInput("Enter the project name ["
        + curProject.getProjectName() + "]");
```

    c. Create a new `Project` object. If the user input for a value is not `null`, add the value to the `Project` object. If the value is `null`, add the value from `curProject`. Repeat for all `Project` variables.

```
Project project = new Project();
project.setProjectName(Objects.isNull(projectName)
    ? curProject.getProjectName() : projectName);
```

d. Set the project ID field in the `Project` object to the value in the `curProject` object.

e. Call `projectService.modifyProjectDetails()`. Pass the `Project` object as a parameter. Let Eclipse create the method for you in `ProjectService.java`.

f. Reread the current project to pick up the changes by calling `projectService.fetchProjectById()`. Pass the project ID obtained from `curProject`.

```
projectService.modifyProjectDetails(project);
curProject = projectService
    .fetchProjectById(curProject.getProjectId());
```

g. Save all files. At this point you should have no compilation errors.

Changes to the project service

In this section you will make changes to the project service. The service is responsible for calling the DAO to update the project details and to return those details to the caller. If the project cannot be found, the service throws an exception. The service method is called by the menu application class, and results are returned to that class.

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

1. In the method `modifyProjectDetails()`,

   a. Call `projectDao.modifyProjectDetails()`. Pass the `Project` object as a parameter. The DAO method returns a `boolean` that indicates whether the `UPDATE` operation was successful. Check the return value. If it is `false`, throw a `DbException` with a message that says the project does not exist.

```
public void modifyProjectDetails(Project project) {
  if(!projectDao.modifyProjectDetails(project)) {
    throw new DbException("Project with ID="
        + project.getProjectId() + " does not exist.");
  }
}
```

   b. Let Eclipse create the `modifyProjectDetails()` method for you in `ProjectDao.java`. Save all files. At this point you should have no compilation errors.

Changes to the project DAO

Now, complete the code in the project DAO to update the project details. The method structure is similar to the `insertProject()` method. You will write the SQL `UPDATE` statement with the parameter placeholders. Then, obtain a `Connection` and start a transaction. Next, you will obtain a

`PreparedStatement` object and set the six parameter values. Finally, you will call `executeUpdate()` on the `PreparedStatement` and commit the transaction.

The difference in this method and the insert method is that you will examine the return value from `executeUpdate()`. The `executeUpdate()` method returns the number of rows affected by the `UPDATE` operation. Since a single row is being acted on (comparing to the primary key in the `WHERE` clause guarantees this), the return value should be `1`. If it is `0` it means that no rows were acted on and the primary key value (project ID) is not found. So, the method returns `true` if `executeUpdate()` returns `1` and `false` if it returns `0`.

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

1. In `modifyProjectDetails()`, write the SQL statement to modify the project details. Do not update the project ID – it should be part of the `WHERE` clause. Remember to use question marks as parameter placeholders.

```
// @formatter:off
String sql = ""
    + "UPDATE " + PROJECT_TABLE + " SET "
    + "project_name = ?, "
    + "estimated_hours = ?, "
    + "actual_hours = ?, "
    + "difficulty = ?, "
    + "notes = ? "
    + "WHERE project_id = ?";
// @formatter:on
```

2. Obtain the `Connection` and `PreparedStatement` using the appropriate `try-with-resource` and `catch` blocks. Start and rollback a transaction as usual. Throw a `DbException` from each `catch` block.

3. Set all parameters on the `PreparedStatement`. Call `executeUpdate()` and check if the return value is `1`. Save the result in a variable.

4. Commit the transaction and return the result from `executeUpdate()` as a `boolean`. At this point there should be no compilation errors.

Test it

1. First, test the application by updating project details without selecting a project. You should receive an error message. Include in your video a shot of the console showing the selections and error message.

2. Next, select a project. Then, select "`Update project details`". Enter new project details and update the project. Include in your video a shot of the console showing the selected project details, the data you input, and the new project details. It should look something like this:

```
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: 4
Enter the project name [Hang a door]: Hang a closet door
Enter the estimated hours [4.00]: 4.5
Enter the actual hours + [3.00]: 3.5
Enter the project difficulty (1-5) [3]: 4
Enter the project notes [Use the door hangers from Home Depot]:
Connection to schema 'projects' is successful.
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
  4) Update project details

You are working with project:
   ID=1
   name=Hang a closet door
   estimatedHours=4.50
   actualHours=3.50
   difficulty=4
   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
```

Delete a project

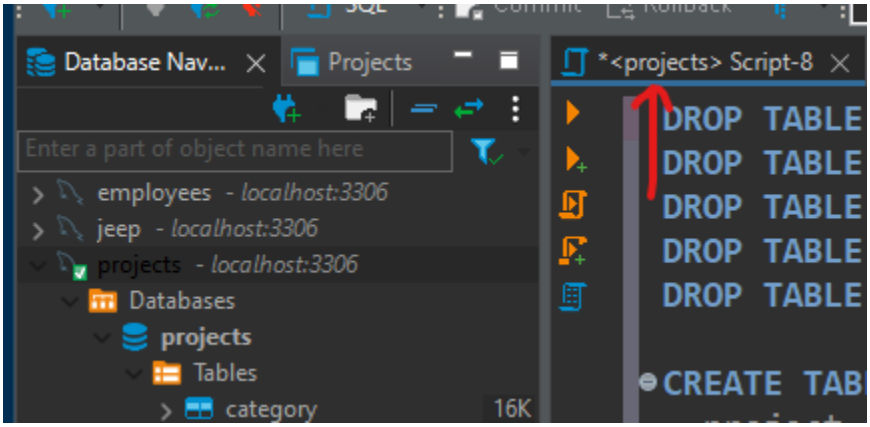
In this section, you will write the code to delete a project. This will require a little preparation. You must verify that `ON DELETE CASCADE` in the `CREATE TABLE` statements works to remove child rows (materials, steps, and project_category rows). This means that you will need to make sure the project has child records. Since the application does not currently add the child rows, you will need to add them using a MySQL client like DBeaver or the MySQL CLI.

Hint: you may want to test this a couple of times. If you add some insert statements at the end of projects-schema.sql, you can simply load and execute the SQL statements as many times as you want. In the following example, not all `CREATE TABLE` statements are shown.

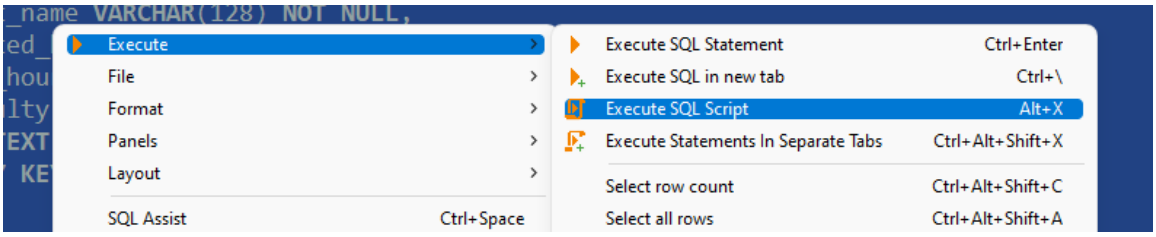
```
CREATE TABLE project_category (
  project_id INT NOT NULL,
  category_id INT NOT NULL,
  FOREIGN KEY (project_id) REFERENCES project (project_id) ON DELETE CASCADE,
  FOREIGN KEY (category_id) REFERENCES category (category_id) ON DELETE CASCADE,
  UNIQUE KEY (project_id, category_id)
);

-- Add some data

INSERT INTO project (project_name, estimated_hours, actual_hours, difficulty, notes) VALUES ('H
INSERT INTO material (project_id, material_name, num_required, cost) VALUES (1, 'Door hangers',
INSERT INTO material (project_id, material_name, num_required, cost) VALUES(1, 'Screws', 20, 4.
INSERT INTO step (project_id, step_text, step_order) VALUES(1, 'Align hangers on opening side c
INSERT INTO step (project_id, step_text, step_order) VALUES(1, 'Screw hangers into frame', 2);
INSERT INTO category (category_id, category_name) VALUES(1, 'Doors and Windows');
INSERT INTO category (category_id, category_name) VALUES(2, 'Repairs');
INSERT INTO category (category_id, category_name) VALUES(3, 'Gardening');
INSERT INTO project_category (project_id, category_id) VALUES(1, 1);
INSERT INTO project_category (project_id, category_id) VALUES(1, 2);
```

Here are the steps for DBeaver:

1. Right-click on the connection name. Select "`SQL Editor`" / "`Recent SQL script`". The editor should open and it should have the name `<projects>` in the top tab (assuming the connection is named "projects").



2. Paste the entire contents of `projects-schema.sql` into the DBeaver editor. Select all the text in the editor. Right-click in the editor. Select "`Execute`" / "`Execute SQL Script`"



Changes to the menu application

In this section you will add code to display a new menu operation to the user ("`Delete a project`"). Then you will add the `case` statement to the `switch`. Next, you will write the method that will list the projects to delete, get the project ID from the user, and call the service to delete the project.

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

1. Add a new option: "`5) Delete a project`" to the list of operations.
2. Add `case 5` to the `switch` statement. Call the method `deleteProject()`. Let Eclipse create the method for you.
3. In method `deleteProject()`:
    a. Call method `listProjects()`.
    b. Ask the user to enter the ID of the project to delete.
    c. Call `projectService.deleteProject()` and pass the project ID entered by the user.
    d. Print a message stating that the project was deleted. (If it wasn't deleted, an exception is thrown by the service class.)
    e. Add a check to see if the project ID in the current project is the same as the ID entered by the user. If so, set the value of `curProject` to `null`.
    f. Have Eclipse create the `deleteProject()` method in the project service.
    g. Save all files. At this point there should be no compilation errors.

Changes to the project service

The `deleteProject()` method in the service is very similar to the `modifyProjectDetails()` method. You will call the `deleteProject()` method in the DAO class and check the `boolean` return value. If the return value is `false`, a `DbException` is thrown with a message that the project with the given ID does not exist. The exception will be picked up by the exception handler in the application menu class.

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

1. Call `deleteProject()` in the project DAO. Pass the project ID as a parameter. The method returns a `boolean`. Test the return value from the method call. If it returns `false`, throw a `DbException` with a message stating that the project doesn't exist.
2. Have Eclipse create the `deleteProject()` method in the `ProjectDao` class.
3. Save all files. At this point there should be no compilation errors.

Changes to the project DAO

The `deleteProject()` method in the DAO is very similar to the `modifyProjectDetails()` method. You will first create the SQL `DELETE` statement. Then, you will obtain the `Connection` and `PreparedStatement`, and set the project ID parameter on the `PreparedStatement`. Then, you will call `executeUpdate()` and verify that the return value is 1, indicating a successful deletion. Finally, you will commit the transaction and return success or failure.

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

1. In the method `deleteProject()`:
    a. Write the SQL `DELETE` statement. Remember to use the placeholder for the project ID in the `WHERE` clause.
    b. Obtain a `Connection` and a `PreparedStatement`. Start, commit, and rollback a transaction in the appropriate sections.
    c. Set the project ID parameter on the `PreparedStatement`.
    d. Return `true` from the menu if `executeUpdate()` returns `1`.

Test it

In this section, you will perform two tests. The first test will delete a project with an unknown project ID and the second test will actually perform the deletion.

Delete with invalid ID

This tests the delete operation with an invalid project ID.

1. Run the application.
2. Select "`Delete a project`". When you are prompted to enter a project ID to delete, enter an invalid ID.
3. Include a shot of the console showing that an error was generated, and that the application handled it gracefully. Here is a sample:

```
These are the available selections. Press the Enter key to quit:
  1) Add a project
  2) List projects
  3) Select a project
  4) Update project details
  5) Delete a project

You are not working with a project.
Enter a menu selection: 5
Connection to schema 'projects' is successful.

Projects:
   1: Hang a closet door
Enter the ID of the project to delete: 57
Connection to schema 'projects' is successful.

Error: projects.exception.DbException: Project with ID=57 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
  4) Update project details
  5) Delete a project

You are not working with a project.
Enter a menu selection:
Exiting the menu.
```

Delete a project

In this section you will test that you can do an actual deletion.

1. Run the application.
2. Select "Delete a project". When you are prompted to enter a project ID to delete, enter a valid ID.
3. List the projects to show that the project was deleted with no errors.
4. Include a shot of the console. Here is a sample:

```
These are the available selections. Press the Enter key to quit:
  1) Add a project
  2) List projects
  3) Select a project
  4) Update project details
  5) Delete a project

You are not working with a project.
Enter a menu selection: 5
Connection to schema 'projects' is successful.

Projects:
   1: Hang a closet door
Enter the ID of the project to delete: 1
Connection to schema 'projects' is successful.
Project 1 was deleted successfully.

These are the available selections. Press the Enter key to quit:
  1) Add a project
  2) List projects
  3) Select a project
  4) Update project details
  5) Delete a project

You are not working with a project.
Enter a menu selection: 2
Connection to schema 'projects' is successful.

Projects:

These are the available selections. Press the Enter key to quit:
  1) Add a project
  2) List projects
  3) Select a project
  4) Update project details
  5) Delete a project

You are not working with a project.
Enter a menu selection:
Exiting the menu.
```

5. Verify that materials, steps, and project_category rows were deleted as well. Use DBeaver or the MySQL CLI for this. The child rows should have been deleted due to the ON DELETE CASCADE in the foreign key statements.

Solutions

In these solutions, only the changed parts of the code are shown.

ProjectsApp.java

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

```java
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;

        case 4:
          updateProjectDetails();
          break;

        case 5:
          deleteProject();
          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.");
    }
  }
}
```

```java
private void deleteProject() {
  listProjects();

  Integer projectId = getIntInput("Enter the ID of the project to delete");

  projectService.deleteProject(projectId);
  System.out.println("Project " + projectId + " was deleted successfully.");

  if(Objects.nonNull(curProject) && curProject.getProjectId().equals(projectId)) {
    curProject = null;
  }
}
```

```java
private void updateProjectDetails() {
  if(Objects.isNull(curProject)) {
    System.out.println("\nPlease select a project.");
    return;
  }

  String projectName =
      getStringInput("Enter the project name [" + curProject.getProjectName() + "]");

  BigDecimal estimatedHours =
      getDecimalInput("Enter the estimated hours [" + curProject.getEstimatedHours() + "]");

  BigDecimal actualHours =
      getDecimalInput("Enter the actual hours + [" + curProject.getActualHours() + "]");

  Integer difficulty =
      getIntInput("Enter the project difficulty (1-5) [" + curProject.getDifficulty() + "]");

  String notes = getStringInput("Enter the project notes [" + curProject.getNotes() + "]");

  Project project = new Project();

  project.setProjectId(curProject.getProjectId());
  project.setProjectName(Objects.isNull(projectName) ? curProject.getProjectName() : projectName);

  project.setEstimatedHours(
      Objects.isNull(estimatedHours) ? curProject.getEstimatedHours() : estimatedHours);

  project.setActualHours(Objects.isNull(actualHours) ? curProject.getActualHours() : actualHours);
  project.setDifficulty(Objects.isNull(difficulty) ? curProject.getDifficulty() : difficulty);
  project.setNotes(Objects.isNull(notes) ? curProject.getNotes() : notes);

  projectService.modifyProjectDetails(project);

  curProject = projectService.fetchProjectById(curProject.getProjectId());
}
```

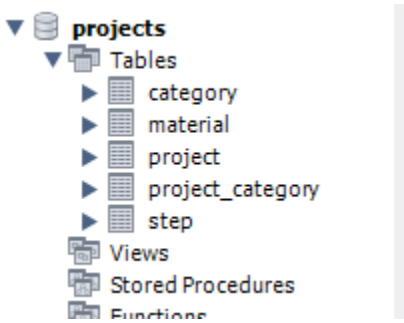
ProjectService.java

```
public void modifyProjectDetails(Project project) {
  if(!projectDao.modifyProjectDetails(project)) {
    throw new DbException("Project with ID=" + project.getProjectId() + " does not exist.");
  }
}

/**
 * @param projectId
 */
public void deleteProject(Integer projectId) {
  if(!projectDao.deleteProject(projectId)) {
    throw new DbException("Project with ID=" + projectId + " does not exist.");
  }
}
```

ProjectDao.java

```
public boolean modifyProjectDetails(Project project) {
  // @formatter:off
  String sql = ""
      + "UPDATE " + PROJECT_TABLE + " SET "
      + "project_name = ?, "
      + "estimated_hours = ?, "
      + "actual_hours = ?, "
      + "difficulty = ?, "
      + "notes = ? "
      + "WHERE project_id = ?";
  // @formatter:on

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

    try(PreparedStatement stmt = conn.prepareStatement(sql)) {
      setParameter(stmt, 1, project.getProjectName(), String.class);
      setParameter(stmt, 2, project.getEstimatedHours(), BigDecimal.class);
      setParameter(stmt, 3, project.getActualHours(), BigDecimal.class);
      setParameter(stmt, 4, project.getDifficulty(), Integer.class);
      setParameter(stmt, 5, project.getNotes(), String.class);
      setParameter(stmt, 6, project.getProjectId(), Integer.class);

      boolean modified = stmt.executeUpdate() == 1;
      commitTransaction(conn);

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

```java
public boolean deleteProject(Integer projectId) {
  String sql = "DELETE FROM " + PROJECT_TABLE + " WHERE project_id = ?";

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

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

      boolean deleted = stmt.executeUpdate() == 1;

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



MySQL Week 11 Exercises

```sql
DROP TABLE IF EXISTS project_category;
DROP TABLE IF EXISTS category;
DROP TABLE IF EXISTS step;
DROP TABLE IF EXISTS material;
DROP TABLE IF EXISTS project;



CREATE TABLE project (
 project_id INT AUTO_INCREMENT NOT NULL,
 project_name VARCHAR(128) NOT NULL,
 estimated_hours DECIMAL(7, 2),
 actual_hours DECIMAL (7, 2),
 difficulty INT,
 notes TEXT,
 PRIMARY KEY (project_id)
);

CREATE TABLE material (
 material_id INT AUTO_INCREMENT NOT NULL,
 project_id INT NOT NULL,
 material_name VARCHAR(128) NOT NULL,
 num_required INT,
 cost DECIMAL (7, 2),
 PRIMARY KEY (material_id),
 FOREIGN KEY (project_id) REFERENCES project (project_id) ON DELETE CASCADE
);

CREATE TABLE step (
 step_id INT AUTO_INCREMENT NOT NULL,
 project_id INT NOT NULL,
 step_text TEXT NOT NULL,
 step_order INT NOT NULL,
 PRIMARY KEY (step_id),
 FOREIGN KEY (project_id) REFERENCES project (project_id) ON DELETE CASCADE
);

CREATE TABLE category (
 category_id INT AUTO_INCREMENT NOT NULL,
 category_name VARCHAR(128) NOT NULL,
 PRIMARY KEY (category_id)
);

CREATE TABLE project category (
 project_id INT NOT NULL,
 category_id INT NOT NULL,
 FOREIGN KEY (project_id) REFERENCES project (project_id) ON DELETE CASCADE,
 FOREIGN KEY (category_id) REFERENCES category (category_id) ON DELETE CASCADE,
 UNIQUE KEY (project_id, category_id)
);
```

Context Help Snippets

Output

Action Output

#	Time	Action	Message	Duration / Fetch
5	15:53:43	DROP TABLE IF EXISTS project	0 row(s) affected	0.000 sec
6	15:53:43	CREATE TABLE project (project_id INT AUTO_INCREMENT NOT NULL, project_name VARCHAR(128) NOT NUL...	0 row(s) affected	0.016 sec
7	15:53:43	CREATE TABLE material (material_id INT AUTO_INCREMENT NOT NULL, project_id INT NOT NULL, material_na...	0 row(s) affected	0.015 sec
8	15:53:43	CREATE TABLE step (step_id INT AUTO_INCREMENT NOT NULL, project_id INT NOT NULL, step_text TEXT N...	0 row(s) affected	0.031 sec
9	15:53:43	CREATE TABLE category (category_id INT AUTO_INCREMENT NOT NULL, category_name VARCHAR(128) NOT...	0 row(s) affected	0.016 sec
10	15:53:43	CREATE TABLE project_category (project_id INT NOT NULL, category_id INT NOT NULL, FOREIGN KEY (project...	0 row(s) affected	0.016 sec

```sql
use projects;
INSERT INTO project (project_name, estimated_hours, actual_hours, difficulty, notes) VALUES ('Hang a door', 4, 3, 3, 'Use the door hangers from Home Depot
INSERT INTO material (project_id, material_name, num_required, cost) VALUES (1, 'Door in frame', 1, null);
INSERT INTO material (project_id, material_name, num_required, cost) VALUES (1, 'Package of door hangers from Home Depot', 1, null);
INSERT INTO material (project_id, material_name, num_required, cost) VALUES (1, '2 inch screws', 20, null);
INSERT INTO step (project_id, step_text, step_order) VALUES (1, 'Align hangers on opening side vertically on the wall', 1);
INSERT INTO step (project_id, step_text, step_order) VALUES (1, 'Screw hangers into the frame', 2);
INSERT INTO category (category_id, category_name) VALUES (1, 'Doors and Windows');
INSERT INTO category (category_id, category_name) VALUES (2, 'Repairs');
INSERT INTO category (category_id, category_name) VALUES (3, 'Gardening');
INSERT INTO project_category (project_id, category_id) VALUES (1, 1);
INSERT INTO project_category (project_id, category_id) VALUES (1, 2);
```

Automatic context h
disabled. Use the tool
manually get help fo
current caret position
toggle automatic h

Context Help | Snippets

Output

Action Output

#	Time	Action	Message	Duration / Fetch
5	15:53:43	DROP TABLE IF EXISTS project	0 row(s) affected	0.000 sec
6	15:53:43	CREATE TABLE project (project_id INT AUTO_INCREMENT NOT NULL, project_name VARCHAR(128) NOT NUL...	0 row(s) affected	0.016 sec
7	15:53:43	CREATE TABLE material (material_id INT AUTO_INCREMENT NOT NULL, project_id INT NOT NULL, material_na...	0 row(s) affected	0.015 sec
8	15:53:43	CREATE TABLE step (step_id INT AUTO_INCREMENT NOT NULL, project_id INT NOT NULL, step_text TEXT N...	0 row(s) affected	0.031 sec
9	15:53:43	CREATE TABLE category (category_id INT AUTO_INCREMENT NOT NULL, category_name VARCHAR(128) NOT...	0 row(s) affected	0.016 sec
10	15:53:43	CREATE TABLE project_category (project_id INT NOT NULL, category_id INT NOT NULL, FOREIGN KEY (project...	0 row(s) affected	0.016 sec

mysql-java/src/main/resources/project-schema.sql

```java
package projects;

import java.math.BigDecimal;...

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",
            "4) Update a project",
            "5) Delete a project"
        );
    // @formatter:on

    public static void main(String[] args) {
        new ProjectsApp().processUserSelections();
    }
    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;
```

```java
import java.util.List;
import java.util.Objects;
import java.util.Scanner;

import projects.entity.Project;
import projects.exception.DbException;
import projects.service.ProjectService;
```

```
45              case 3:
46                  selectProject();
47                  break;
48              case 4:
49                  updateProjectDetails();
50                  break;
51              case 5:
52                  deleteProject();
53                  break;
54             default:
55                 System.out.println("\n" + selection + " is not a valid selection. Try again.");
56                 break;
57              }
58
59             }
60             catch(Exception e) {
61                 System.out.println("\nError: " + e + " Try again.");
62
63             }
64         }
65
66
67     }
68
69
70
71     private void deleteProject() {
72         listProjects();
73
74         Integer projectId =  getIntInput("Enter the ID of the project to delete");
75
76         projectService.deleteProject(projectId);
77         System.out.println("Project " + projectId + " was deleted successfully");
78
79         if(Objects.nonNull(curProject) && curProject.getProjectId().equals(projectId)) {
80             curProject = null;

81         }
82
83
84     }
85     private void updateProjectDetails() {
86         if(Objects.isNull(curProject)) {
87             System.out.println("\nPlease select a project.");
88             return;
89         }
90
91         String projectName = getStringInput("Enter the project name [" + curProject.getProjectName() + "]");
92
93         BigDecimal estimatedHours = getDecimalInput("Enter the estimated hours [" + curProject.getEstimatedHours() + "]");
94
95         BigDecimal actualHours = getDecimalInput("Enter the actual hours [" + curProject.getActualHours() + "]");
96
97         Integer difficulty = getIntInput("Enter the project difficulty (1-5) [" + curProject.getDifficulty() + "]");
98
99         String notes = getStringInput("Enter the project notes [" + curProject.getNotes() + "]");
100
101        Project project = new Project();
102
103        project.setProjectId(curProject.getProjectId());
104        project.setProjectName(Objects.isNull(projectName) ? curProject.getProjectName() : projectName);
105        project.setEstimatedHours(Objects.isNull(estimatedHours) ? curProject.getEstimatedHours() : estimatedHours);
106        project.setActualHours(Objects.isNull(actualHours) ? curProject.getActualHours() : actualHours);
107        project.setDifficulty(Objects.isNull(difficulty) ? curProject.getDifficulty() : difficulty);
108        project.setNotes(Objects.isNull(notes) ? curProject.getNotes() : notes);
109
110        projectService.modifyProjectDetails(project);
111
112        curProject = projectService.fetchProjectById(curProject.getProjectId());
113
114
115    }
116    private void selectProject() {
117        listProjects();
```

```
        Integer projectId = getIntInput("Enter a project ID to select a project");


        curProject = null;

        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()));
        }

    private void createProject() {
        String projectName = getStringInput("Enter the project name");
        BigDecimal estimatedHours = getDecimalInput("Enter the estimated hours");
        BigDecimal acutalHours = getDecimalInput("Enter the actual hours");
        Integer difficulty = getIntInput("Enter the project difficulty (1-5)");
        String notes = getStringInput("Enter the project notes");

        Project project = new Project();

        project.setProjectName(projectName);
        project.setEstimatedHours(estimatedHours);
        project.setActualHours(acutalHours);
        project.setDifficulty(difficulty);
        project.setNotes(notes);

        Project dbProject = projectService.addProject(project);
        System.out.println("You have successfully created project: " + dbProject);
    }
    private BigDecimal getDecimalInput(String prompt) {
        String input = getStringInput(prompt);

        if(Objects.isNull(input)) {
```

```
            return null;
        }
        try {
            return new BigDecimal(input).setScale(2);
        }
        catch(NumberFormatException e) {
            throw new DbException(input + " is not a valid decimal number.");
        }
    }
    private boolean exitMenu() {
        System.out.println("\nExiting the menu.");
        return true;
    }
    private int getUserSelection() {
        printOperations();

        Integer input = getIntInput("Enter a menu selection");

        return Objects.isNull(input) ? -1 : input;
    }
    private Integer getIntInput(String prompt) {
        String input = getStringInput(prompt);

        if(Objects.isNull(input)) {
            return null;
        }
        try {
            return Integer.valueOf(input);
        }
        catch(NumberFormatException e) {
            throw new DbException(input + " is not a valid number.");
        }
    }
    private String getStringInput(String prompt) {
        System.out.print(prompt + ": ");
        String input = scanner.nextLine();

```

```
        return input.isBlank() ? null : input.trim();


        }

    private void printOperations() {
        System.out.println("\nThese are the available selections. Press the Enter key to quit:");
        operations.forEach(line -> 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);
        }



    }

}
```

```
package projects.dao;

import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Collection;
import java.util.LinkedList;
import java.util.List;
import java.util.Objects;
import java.util.Optional;

import projects.entity.Category;
import projects.entity.Material;
import projects.entity.Project;
import projects.entity.Step;
import projects.exception.DbException;

import provided.util.DaoBase;

@SuppressWarnings("unused")
public class ProjectDao extends DaoBase {
    private static final  String CATEGORY_TABLE = "category";
    private static final  String MATERIAL_TABLE = "material";
    private static final  String PROJECT_TABLE = "project";
    private static final  String PROJECT_CATEGORY_TABLE = "project_category";
    private static final  String STEP_TABLE = "step";


    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)) {
```

```
38                  try(ResultSet rs = stmt.executeQuery()) {
39                      List<Project> projects = new LinkedList<>();
40
41                      while(rs.next()) {
42                          projects.add(extract(rs, Project.class));
43                      }
44                      return projects;
45                  }
46              }
47              catch(Exception e) {
48                  rollbackTransaction(conn);
49                  throw new DbException(e);
50              }
51          }
52              catch(SQLException e) {
53                  throw new DbException(e);
54              }
55              }
56
57
58      public Optional<Project> fetchProjectById(Integer projectId) {
59          String sql = "SELECT * FROM " + PROJECT_TABLE + " WHERE project_id = ?";
60
61          try(Connection conn = DbConnection.getConnection()) {
62              startTransaction(conn);
63
64              try {
65                  Project project = null;
66
67                  try(PreparedStatement stmt = conn.prepareStatement(sql)) {
68
69
70                      setParameter(stmt, 1, projectId, Integer.class);
71
72                  try(ResultSet rs = stmt.executeQuery()) {
73                      if(rs.next()) {
```

```
74                      project = extract(rs, Project.class);
75                  }
76              }
77              }
78
79              if(Objects.nonNull(project)) {
80                  project.getMaterials().addAll(fetchMaterialsForProject(conn, projectId));
81                  project.getSteps().addAll(fetchStepsForProject(conn, projectId));
82                  project.getCategories().addAll(fetchCategoriesForProject(conn, projectId));
83              }
84              commitTransaction(conn);
85
86              return Optional.ofNullable(project);
87
88          }
89          catch(Exception e) {
90              rollbackTransaction(conn);
91              throw new DbException(e);
92          }
93      }
94          catch(SQLException e) {
95          throw new DbException(e);
96      }
97  }
98      private List<Category> fetchCategoriesForProject(Connection conn, Integer projectId)
99              throws SQLException {
100         // @formatter:off
101         String sql = ""
102           + "SELECT c.* FROM " + CATEGORY_TABLE + " c "
103           + " JOIN " + PROJECT_CATEGORY_TABLE + " pc USING (category_id) "
104           + " WHERE project_id = ?";
105          // @formatter:on
106
107           try(PreparedStatement stmt = conn.prepareStatement(sql)) {
108               setParameter(stmt, 1, projectId, Integer.class);
109
110               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;
            }
            }

        }

    public Project insertProject(Project project) {
        // @formatter:off
        String sql = ""
            + "INSERT INTO " + PROJECT_TABLE + " "
            + "(project_name, estimated_hours, actual_hours, difficulty, notes) "
            + " VALUES "
            + "(?, ?, ?, ?, ?)";
        // @formatter:on

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

            try(PreparedStatement stmt = conn.prepareStatement(sql)) {
                setParameter(stmt, 1, project.getProjectName(), String.class);
                setParameter(stmt, 2, project.getEstimatedHours(), BigDecimal.class);
                setParameter(stmt, 3, project.getActualHours(), BigDecimal.class);
                setParameter(stmt, 4, project.getDifficulty(), Integer.class);
                setParameter(stmt, 5, project.getNotes(), String.class);

                stmt.executeUpdate();

                Integer projectId = getLastInsertId(conn, PROJECT_TABLE);
                commitTransaction(conn);

                project.setProjectId(projectId);
                return project;
            }
            catch(Exception e) {
                rollbackTransaction(conn);
                throw new DbException(e);
```

```
185
186             }
187
188             } catch (SQLException e) {
189                 throw new DbException(e);
190             }
191
192
193
194
195
196
197
198
199
200
201
202
203
204     }
205
206
207     public boolean modifyProjectDetails(Project project) {
208         //@formatter:off
209                 String sql = "UPDATE " + PROJECT_TABLE + " SET "
210                         +"project_name = ?, "
211                         +"estimated_hours = ?, "
212                         +"actual_hours = ?, "
213                         +"difficulty = ?, "
214                         +"notes = ? "
215                         +"WHERE project_id = ?";
216
217                 //@formatter:on
218
219                 try(Connection conn = DbConnection.getConnection()){
220                     startTransaction(conn);
221
```

```
222                     try(PreparedStatement stmt = conn.prepareStatement(sql)){
223                         setParameter(stmt, 1, project.getProjectName(), String.class);
224                         setParameter(stmt, 2, project.getEstimatedHours(), BigDecimal.class);
225                         setParameter(stmt, 3, project.getActualHours(), BigDecimal.class);
226                         setParameter(stmt, 4, project.getDifficulty(), Integer.class);
227                         setParameter(stmt, 5, project.getNotes(), String.class);
228                         setParameter(stmt, 6, project.getProjectId(), Integer.class);
229
230                     boolean modified = stmt.executeUpdate() == 1;
231
232                     commitTransaction(conn);
233
234                     return modified;
235
236                     }
237                     catch(Exception e) {
238                         rollbackTransaction(conn);
239                         throw new DbException(e);
240                     }
241                 }
242                 catch(SQLException e) {
243                     throw new DbException(e);
244                 }
245
246             }
247
248
249     public boolean deleteProject(Integer projectId) {
250       String sql = "DELETE FROM " + PROJECT_TABLE + " WHERE project_id = ?";
251
252       try(Connection conn = DbConnection.getConnection()) {
253           startTransaction(conn);
254
255           try(PreparedStatement stmt = conn.prepareStatement(sql)) {
256               setParameter(stmt, 1, projectId, Integer.class);
257
258                 boolean deleted = stmt.executeUpdate() == 1;
259
260                 commitTransaction(conn);
261                 return deleted;
262             }
263               catch(Exception e) {
264                   rollbackTransaction(conn);
265                   throw new DbException(e);
266             }
267
268         } catch (SQLException e) {
269             throw new DbException(e);
270       }
271       }
272
273
274
275
276
277 }
```

```java
package projects.service;

import java.util.List;
import java.util.NoSuchElementException;
import java.util.Optional;

import projects.dao.ProjectDao;
import projects.entity.Project;
import projects.exception.DbException;


@SuppressWarnings("unused")
public class ProjectService {
    private ProjectDao projectDao = new ProjectDao();


    public Project addProject(Project project) {

        return projectDao.insertProject(project);
    }

    public List<Project> fetchAllProjects() {
        return projectDao.fetchAllProjects();
    }


    public Project fetchProjectById(Integer projectId) {
        // Optional<Project> op = projectDao.fetchProjectById(projectId);

        return projectDao.fetchProjectById(projectId).orElseThrow(()
                -> new NoSuchElementException("Project with project ID="
                        + projectId + " does not exist"));
        /*return projectDao.fetchProjectById(projectId).
                orElseThrow(()-> new NoSuchElementException(
                        "Project with project ID= " + projectId
                        + " does not exist.")); */

    }



    public void modifyProjectDetails(Project project) {
        if(!projectDao.modifyProjectDetails(project)) {
            throw new DbException("Project with ID= + project.getProjectId() + does not exist.");
        }

    }



    public void deleteProject(Integer projectId) {
        if(!projectDao.deleteProject(projectId)) {
            throw new DbException("Project with ID=" + projectId + " does not exist.");
        }

    }





    }
```