MySQL Week 5 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 take a screen shot or snip showing the results of the action or the code in the editor.

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.

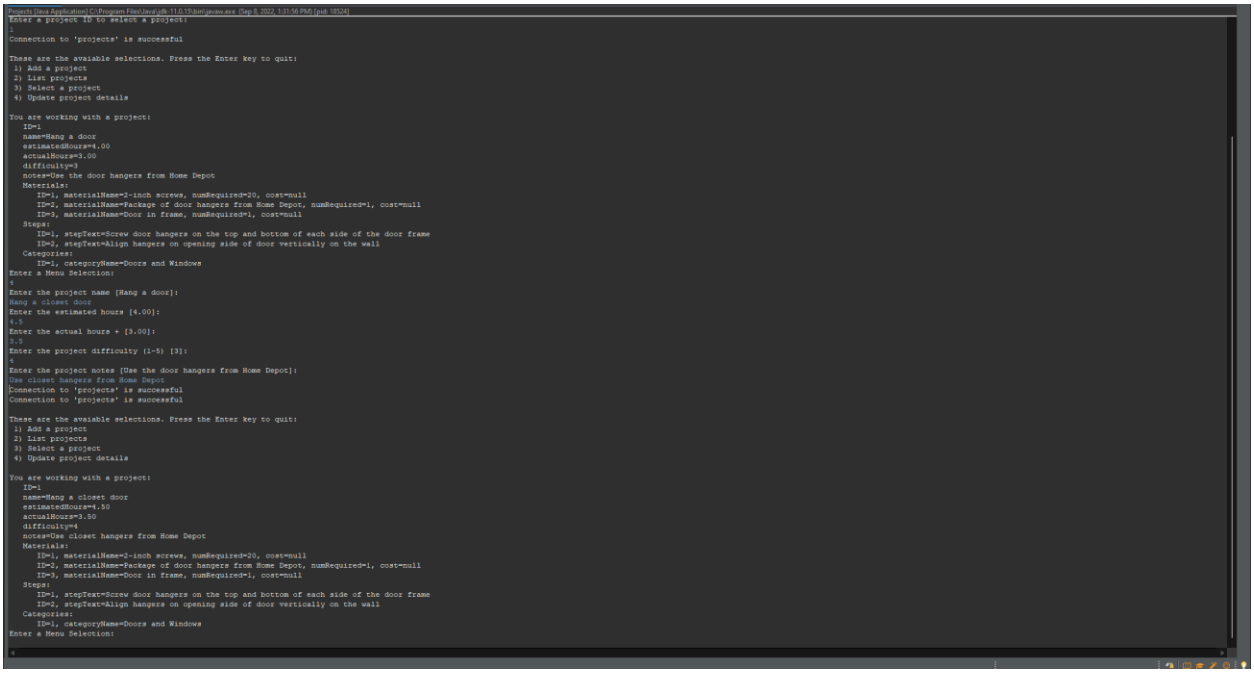
4. `C`ommit the transaction and return the result from `executeUpdate()` as a `boolean`. At this point there should be no compilation errors.

Test it

1. First, test the application by updating project details without selecting a project. You should receive an error message. Submit a screen 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. Submit a screen shot of the console showing the selected project details, the data you input, and the new project details. The screen shot 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

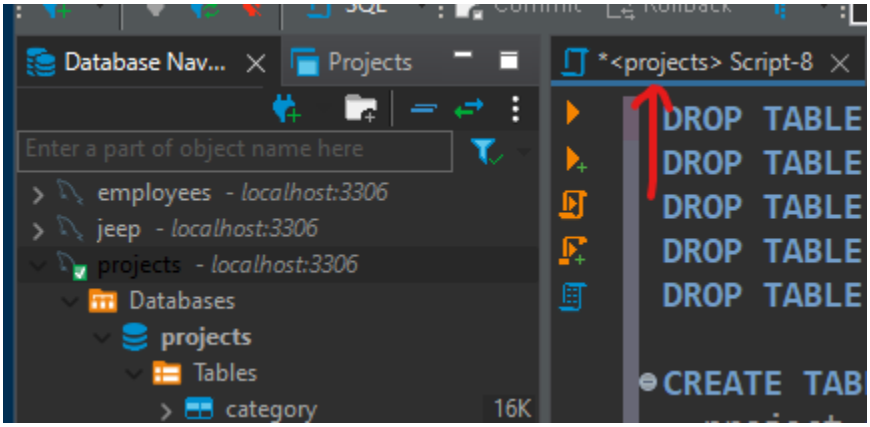
In this section, you will write the code to delete a project. This will require a little preparation. You must verify that `ON DELETE CASCADE` in the `CREATE TABLE` statements works to remove child rows (materials, steps, and project_category rows). This means that you will need to make sure the project 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.

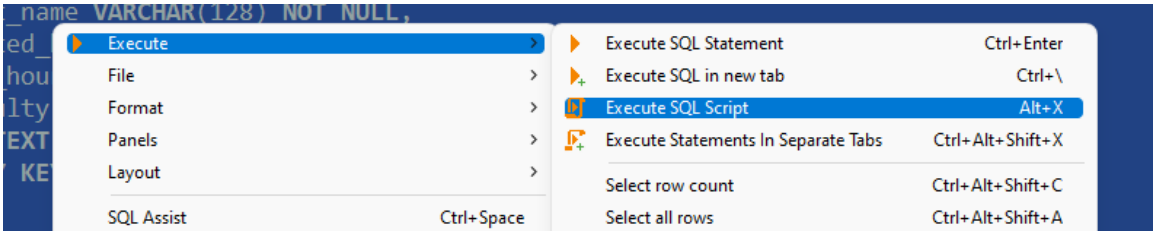
```
CREATE TABLE project_category (
  project_id INT NOT NULL,
  category_id INT NOT NULL,
  FOREIGN KEY (project_id) REFERENCES project (project_id) ON DELETE CASCADE,
  FOREIGN KEY (category_id) REFERENCES category (category_id) ON DELETE CASCADE,
  UNIQUE KEY (project_id, category_id)
);

-- 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 o
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

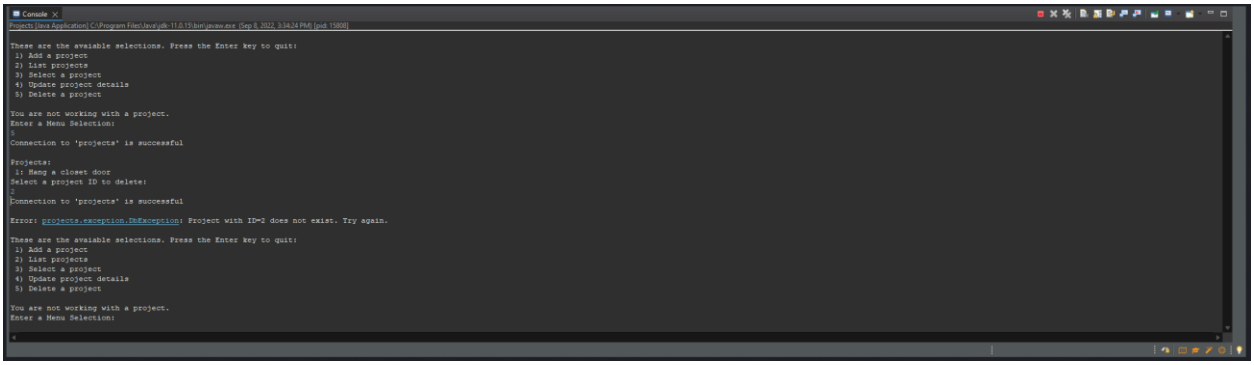
In this section, you will perform two tests. The first test will delete a project with an unknown project ID and the second test will actually perform the deletion.

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. Submit a screen 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. Submit a screen 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.

   RPador (Ranon Pador) (github.com)

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
```

```
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.");
    }
  }
}


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;
  }
}
```

```
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

```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);
  }
}
```