Adding a New Product in the Database

Step-By-Step Flow Document For The Project:

1.Database Setup:

- Create a MySQL database named "Products" using the provided MySQL queries.
- The "Products" database should have a table named "product" with columns: `id` (INT), `name` (VARCHAR), `description` (VARCHAR), and `price` (DOUBLE).

2. HTML Pages:

- Create an `index.html` file with links to "Read Product" and "Add Product" pages.
- Create an `AddProduct.jsp` file with a form to add a new product. The form should include fields for `id`, `name`, `description`, and `price`.

3. Hibernate Configuration:

- Create a `hibernate.cfg.xml` file to configure Hibernate.
- Specify the database connection details (driver class, URL, username, and password) in the configuration file.
- Configure Hibernate to use the MySQL dialect.
- Enable SQL logging and formatting for debugging purposes.

4. Entity Class:

- Create a `Products.java` class representing the `product` table as a Hibernate entity.
- Annotate the class with `@Entity` and specify the table name using `@Table(name = "product")`.
- Define the attributes (`id`, `name`, `description`, and `price`) with appropriate annotations (`@Id`, `@Column`, etc.).
- Generate getter and setter methods for the attributes.

5. Hibernate Utility:

- Create a `HibernateUtil.java` class to build the Hibernate `SessionFactory`.
- Use the Hibernate configuration file (`hibernate.cfg.xml`) and add the `Products` class as an annotated class in the configuration.
- Build the `SessionFactory` and return it.

6. Add Product Servlet:

- Create an `AddProductServlet.java` servlet class to handle the form submission and add a new product to the database.
- Override the `doPost` method to process the form data.
- Extract the values from the request parameters (`id`, `name`, `description`, and `price`).
- Obtain a Hibernate `Session` from the `SessionFactory` using the `HibernateUtil` class.
- Begin a transaction using `session.beginTransaction()`.
- Create a new `Products` object, set its attributes with the extracted values, and save it using `session.save()`.
- Commit the transaction using `tx.commit()`.
- Close the session.

7. Read Product Servlet:

- Create a `ReadProductServlet.java` servlet class to retrieve and display the list of products from the database.
- Override the `doGet` method to handle the GET request.
- Obtain a Hibernate `Session` from the `SessionFactory` using the `HibernateUtil` class.
- Create a Hibernate query to fetch all the products from the database.
- Execute the query and retrieve the list of `Products` objects.
- Close the session.
- Generate an HTML response to display the product list using a loop over the retrieved products.

8. Deployment and Testing:

- Deploy the application to a servlet container (e.g., Tomcat).
- Start the servlet container and access the application through the browser.
- Verify that the "Read Product" page displays the existing products correctly.
- Use the "Add Product" form to add a new product and verify that it is stored in the database.
- Refresh the "Read Product" page to see the newly added product in the list.

That's the step-by-step flow for the project. Follow these instructions to add a new product to the database using