ЛАБОРАТОРНА РОБОТА 6

Тема: Рівень збереження в корпоративних застосунках. Технологія JPA.

На рис. 1 представлено традиційну чотирирівневу архітектуру, яка має велику популярність. У цій архітектурі рівень представлення відповідає за відображення графічного інтерфейсу користувача (Graphical User Interface, GUI) та обробку введення користувача. Часто цей рівень реалізується із застосуванням браузера або окремої програми. Рівень представлення передає всі запити до рівня прикладної логіки, який є ядром програми і містить основну логіку обробки, що визначає бізнес-правила. На цьому рівні знаходяться компоненти, що виконують різні прикладні операції, наприклад облік, пошук, впорядкування даних, обслуговування облікових записів, тощо. Рівень прикладної логіки отримує та зберігає дані у базі, при цьому використовується рівень зберігання. Рівень зберігання забезпечує високорівневі, об'єктно орієнтовані абстракції над рівнем бази даних. Рівень бази даних зазвичай включає систему управління реляційними базами даних (Relational Database Management System, RDBMS).

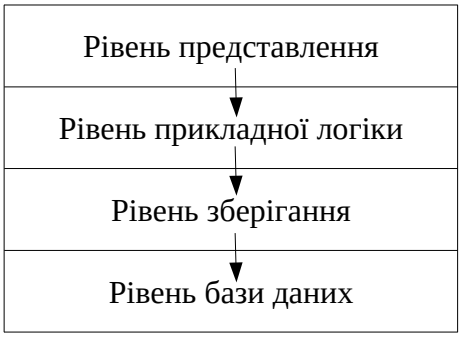


Рис.1. Чотирирівнева архітектура додатку.

Рівень збереження (persistence layer) у корпоративних Java-додатках реалізується через спеціалізовані технології, які забезпечують взаємодію з базами даних, ORM (Object-Relational Mapping), кешуванням та управління транзакціями.

Java Persistence API (JPA) — це стандартна специфікація Java для ORM (Object-Relational Mapping), яка дозволяє розробникам працювати з реляційними базами даних через об'єктну модель. Основна мета — абстрагуватися від SQL-запитів і спростити взаємодію з БД у корпоративних додатках. JPA є основним інструментом для корпоративних Java-додатків, де потрібна гнучкість, масштабованість і легка підтримка складних моделей даних.

Таблиця 1. Порівняння застосування JPA та JDBC

Критерій	**JPA**	**JDBC**
Рівень абстракції	Високий (робота з об'єктами)	Низький (SQL-запити)
Продуктивність	Оптимізована через кеші	Вища, але потребує ручної оптимізації
Складність	Простота для складних моделей	Вища для великих проектів

До основних переваг використання JPA слід віднести відсутність необхідності розробки SQL запитів вручну, що, як правило, веде до скорочення шаблонного коду. Також важливою є переносність створеного коду — один і той самий код використовується для різних СУБД (PostgreSQL, MySQL та інших). Однією з основних цілей JPA було створення простої та зрозумілої технології. Попри складність проблемної області, рішення, яке допомагає розробникам з нею працювати є доступним та інтуїтивно зрозумілим щодо використання.

Основою розуміння технології є поняття сутності (entity), яка використовується в управлінні даними. Загалом, сутність – це представлення Java таблиці бази даних, яка має такі характеристики, як стійкість, ідентичність, трансакційність та деталізація. Сутності мають

атрибути (групу станів, пов’язаних разом як єдине ціле) та відносини (зв’язки). Атрибути та зв’язки зберігаються в реляційній базі даних. Сутність може мати зв’язки з будь-якою кількістю інших сутностей. В об’єктно-орієнтованій парадигмі до сутності додається поведінка (методи та алгоритми обробки атрибутів) і, таким чином, утворюється його об’єкт. У Jakarta Persistence будь-який визначений програмою об’єкт може бути сутністю. Першою і основною характеристикою сутностей є їх здатність до збереження. Загалом це означає, що їх стан може бути представлений у сховищі даних і доступний для використання пізніше, після завершення процесу, який їх створив.

Сутність не зберігається автоматично – для створення її представлення додаток має викликати відповідний метод API. Це дає додатку повний контроль над процесом збереження, дозволяючи йому виконувати операції з даними та бізнес-логікою, а зберігати сутність лише тоді, коли це необхідно. Тому сутності можна змінювати, не зберігаючи їх одразу, і саме додаток визначає, коли їх потрібно зробити персистентними. Як і будь-який інший об'єкт у Java, сутність має об'єктну ідентичність, але коли вона зберігається в базі даних, вона також отримує персистентну ідентичність. Об'єктна ідентичність означає лише відмінність між об'єктами, які займають пам’ять у межах виконуваного процесу. Натомість персистентна ідентичність (або ідентифікатор) — це унікальний ключ, який однозначно визначає конкретний екземпляр сутності та відрізняє його від інших екземплярів того ж типу. Сутність набуває персистентної ідентичності, коли її представлення зберігається в сховищі даних, тобто у вигляді рядка в таблиці бази даних. Якщо сутність ще не збережена в базі даних, то навіть якщо її ідентифікатор вже встановлений у полі, вона все одно не має персистентної ідентичності. Таким чином, ідентифікатор сутності еквівалентний первинному ключу в таблиці бази даних, яка зберігає її стан.

Транзакційність у Jakarta EE — це спосіб, яким платформа забезпечує надійне керування транзакціями, щоб гарантувати, що всі операції в транзакції або успішно завершуються, або скасовуються, зберігаючи цілісність даних. Це особливо важливо для корпоративних застосунків, які працюють з базами даних або іншими ресурсами. Для її забезпечення в Jakarta EE використовується API Jakarta Transactions (раніше відоме як Java Transaction API або JTA).

Сутності можна назвати квазітранзакційними. Хоча їх можна створювати, оновлювати та видаляти в будь-якому контексті, зазвичай ці операції виконуються в межах транзакції, оскільки саме транзакція забезпечує фіксацію змін у базі даних. Зміни в базі даних або успішно зберігаються, або повністю відхиляються, тому персистентне представлення сутності має бути транзакційним. Але у пам’яті сутності можуть змінюватися без фактичного збереження цих змін у базі даних. Навіть якщо вони включені в транзакцію, у разі її відкату (rollback) або помилки транзакції вони можуть залишитися у невизначеному чи неконсистентному стані. В оперативній пам’яті сутності є звичайними Java-об'єктами, які підкоряються всім правилам і обмеженням, що застосовуються віртуальною машиною Java (JVM) до інших об'єктів.

Ще одним важливим поняттям для розуміння персистентності в технологіях Jakarta EE є гранулярність, що визначає рівень деталізації або розмір одиниць у різних частинах платформи. Це багатогранне поняття, яке проявляється в управлінні транзакціями, дизайні компонентів, зокрема Enterprise JavaBeans (EJB), а також у доступі до даних через Jakarta Persistence. У контексті Jakarta Transactions (JTA), гранулярність стосується рівня, на якому встановлюються межі транзакцій. Транзакції в Jakarta EE забезпечують атомарність, узгодженість, ізоляцію та стійкість (ACID) для операцій, і їхній розмір може варіюватися:

- Дрібнозернисті транзакції - транзакція охоплює лише один метод, аннотований як @Transactional, що дозволяє точніше керувати кожною операцією, але може призвести до більшої кількості транзакцій, що впливає на продуктивність.
- Грубозернисті транзакції - транзакція охоплює весь клас або кілька методів, що може бути ефективнішим для великих операцій, але менш гнучким у разі помилок, оскільки скасування торкнеться більшого обсягу даних.

Сутності — це дрібнозернисті об'єкти, які містять агрегований стан, зазвичай збережений в одному місці, наприклад, у рядку (записі) таблиці, і часто мають зв’язки з іншими сутностями. У загальному сенсі вони є бізнес-об'єктами, що мають конкретне значення для застосунку, який їх використовує. Теоретично сутності можуть визначатись і як надто дрібнозернисті (наприклад, для

збереження окремого рядка), так і навпаки — занадто великі (наприклад, містити до 500 стовпців даних). У Jakarta Persistence сутності задумувались як об'єкти з меншою гранулярностю. Ідеально сутності мають бути легковаговими об'єктами, розмір яких приблизно відповідає середньостатистичному Java-об'єкту.

Кожна сутність Jakarta Persistence має певні метадані, що її описують. Ці метадані можуть бути частиною збереженого файлу класу або знаходитися окремо від нього, проте вони не зберігаються в базі даних. Їхнє призначення — допомагати рівню персистентності ідентифікувати, інтерпретувати та правильно керувати сутністю від моменту її завантаження до виклику під час виконання. Обсяг обов’язкових метаданих для кожної сутності мінімальний, що робить їх легкими для визначення та використання. Але існує можливість задавати значно більше метаданих, ніж необхідно за замовчуванням. Їхня кількість може варіюватися залежно від вимог застосунку, і вони можуть використовуватись для налаштування кожної деталі конфігурації сутності або її відображення в сховище даних. Метадані сутності можна визначати двома способами: за допомогою анотацій або через XML. Обидва варіанти є рівноправними, а вибір між ними залежить від особистих уподобань або процесу розробки.

Анотації є мовною можливістю, введеною в Java SE 5, яка дозволяє додавати структуровані та типізовані метадані безпосередньо до вихідного коду. Хоча Jakarta Persistence не вимагає використання анотацій, вони є зручним способом роботи з API. Завдяки тому, що анотації розташовуються безпосередньо поряд із програмними артефактами, немає необхідності використовувати додаткові файли чи спеціальні мови для опису метаданих, наприклад, XML.

Однією з особливостей Jakarta Persistence є підтримка анотації @Repeatable. Це дає змогу застосовувати певну анотацію кілька разів для якогось класу чи атрибута без необхідності використання контейнерної анотації, що робить код прозорішим та зручним. До анотацій, які можуть повторюватись під час використання Jakarta Persistence, відносяться:

@AssociationOverride
@AttributeOverride
@Convert
@JoinColumn
@MapKeyJoinColumn
@NamedEntityGraph
@NamedNativeQuery
@NamedQuery
@NamedStoredProcedureQuery
@PersistenceContext
@PersistenceUnit
@PrimaryKeyJoinColumn
@SecondaryTable
@SqlResultSetMapping

Альтернативно, метадані можна визначити в XML-файлах, які зазвичай розташовуються в директорії META-INF певного проєкту. XML має пріоритет над анотаціями, якщо вони конфліктують. Цей підхід дозволяє відділити конфігурацію від коду, але вимагає від розробника узгоджувати внесені виправлення в код з вмістом відповідного XML файлу.

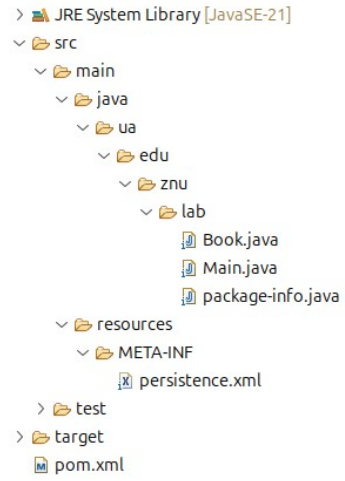
Невід’ємною частиною Jakarta Persistence є концепція конфігурації за винятком. Вона передбачає, що компоненти мають налаштування за замовчуванням, а власні налаштування вказуються лише тоді, коли потрібно їх змінити. Це зменшує кількість необхідного коду для конфігурації та робить платформу зручною для використання. Наприклад, у Jakarta Persistence ім’я таблиці за замовчуванням — це ім’я класу сутності, але можна вказати інше ім’я за допомогою анотації @Table, якщо це необхідно. Платформа автоматично обробляє типові випадки і тільки винятки корегуються. У дійсності цей підхід широко застосовується в різних частинах Jakarta EE, таких як сервлети, EJB і безпека, де за замовчуванням є стандартні налаштування, які можна перевизначити за потреби.

Проте типові значення вбудовані в API і можуть залишатися невидимими для користувачів. Це може ускладнити розуміння роботи механізму збереження, а також ускладнити налагодження або зміну поведінки програми, коли це стає необхідним. Типові значення призначені для того, щоб розробник міг швидко почати роботу із працездатним рішенням, поступово вдосконалюючи та розширюючи його функціональність у міру збільшення складності застосунку. Навіть якщо типові значення зазвичай відповідають очікуванням, важливо знати, які саме значення використовуються. Наприклад, якщо використовується типове ім'я таблиці, необхідно розуміти, яку саме таблицю очікує середовище виконання або яку таблицю буде створено при генерації схеми.

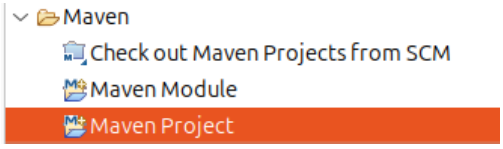
Практична частина

Створення простого застосунку з JPA

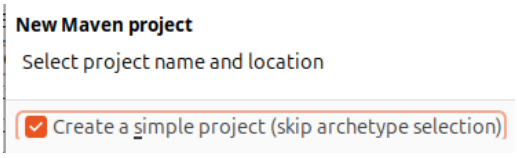
Проєкт застосунку повинен мати наступну структуру:



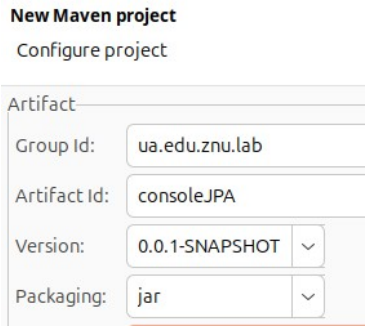
Для цього в Eclipse виконуємо File → New → Maven Project:



Обираємо опцію «Create a simple project»

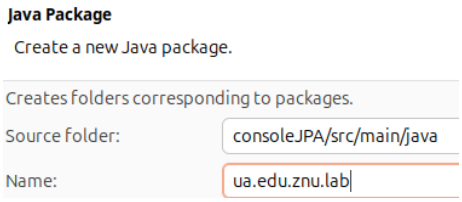


На наступному кроці вказуємо

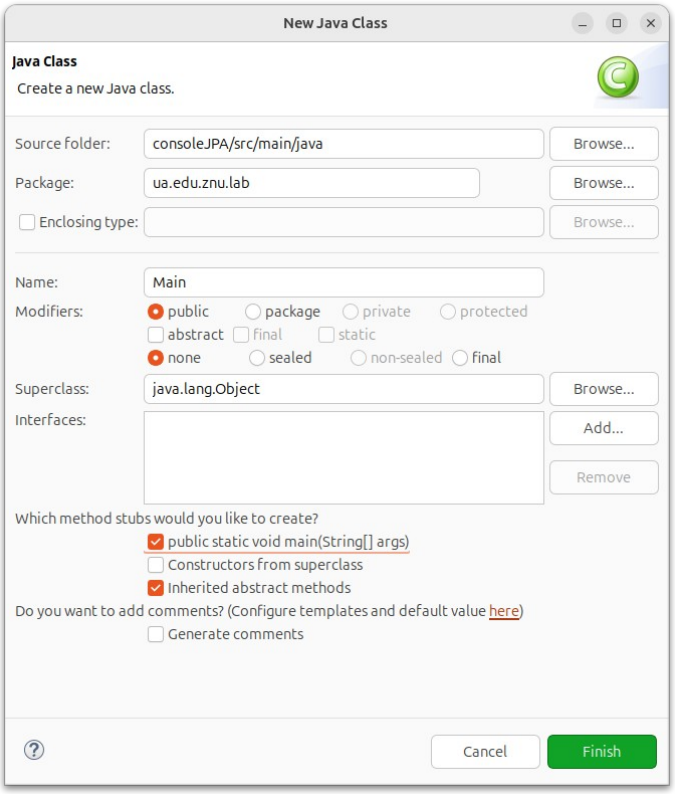


Створена структура потребує додаткових корегувань. Насамперед слід змінити версію SE: натискаємо праву кнопку миші на JRE System Library -> Properties і далі обираємо необхідну версію SE, наприклад:

Необхідно впевнитись, що в структурі проєкту створено пакет ua.edu.znu.lab. Якщо його немає, то додати до проєкту вручну

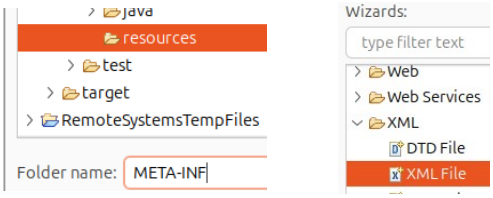


Додати до проєкту в створений пакет файл Main.java з методом main():



а також клас Book.java для створення сутностей.

У каталогу resources необхідно створити каталог META-INF, до якого додати файл persistence.xml.



Наступним налаштовуємо файл pom.xml, який використовується Maven при збирані проєкту. Його вміст для даного проєкту наступний:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>ua.edu.znu.lab</groupId>
    <artifactId>consoleJPA</artifactId>
    <version>1.0</version>
    <packaging>jar</packaging>
    <name>consoleJPA-1.0</name>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <jakartaee>10.0.0</jakartaee>
    </properties>

    <dependencies>
        <dependency>
            <groupId>jakarta.platform</groupId>
            <artifactId>jakarta.jakartaee-api</artifactId>
            <version>${jakartaee}</version>
            <scope>provided</scope>
        </dependency>
        <!-- JPA провайдер (EclipseLink, Hibernate тощо) -->
        <dependency>
            <groupId>org.eclipse.persistence</groupId>
            <artifactId>eclipselink</artifactId>
            <version>4.0.5</version>
        </dependency>
        <!-- Apache Derby -->
        <dependency>
            <groupId>org.apache.derby</groupId>
            <artifactId>derby</artifactId>
            <version>10.17.1.0</version>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.derby</groupId>
            <artifactId>derbytools</artifactId>
            <version>10.17.1.0</version>
            <scope>runtime</scope>
        </dependency>
        <!-- Логування -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>2.0.12</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <source>21</source>
                    <target>21</target>
                </configuration>
            </plugin>
      </plugins>
   </build>
</project>
```

Для перевірки правильності та актуальності версій залежностей слід використовувати ресурс https://repo.maven.apache.org/maven2/.

Тепер відредагуємо вміст файлу Book.class, який буде слугувати основою сутності. Звичайні Java-класи можна перетворити на сутності шляхом додавання відповідних анотацій. Достатньо додати кілька анотацій, щоб майже будь-який клас із конструктором без аргументів став сутністю. Головним програмним артефактом сутності є клас сутності, який повинен відповідати таким вимогам:

- анотований за допомогою @jakarta.persistence.Entity;
- має публічний або захищений конструктор без аргументів (додаткові конструктори допускаються);
- не може бути оголошений як final, його методи або змінні екземпляра, що зберігаються, не можуть мати оголошення final;
- повинен реалізовувати інтерфейс Serializable, що дозволить передавати сутність за значенням як від'єднаний об'єкт через віддалений бізнес-інтерфейс session bean;
- може розширювати як сутності, так і звичайні класи;
- змінні екземпляра, що зберігаються, оголошені як private, protected або без модифікатора доступу (package-private). Вони можуть бути доступні лише через методи класу-сутності.

Дотримання цих вимог гарантує правильну інтеграцію сутностей із механізмом збереження Jakarta Persistence та ефективне управління їхнім життєвим циклом.

Приклад класу-сутності Book:

```
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Book {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
   private String title;
   private String author;

   // get та set методи
   public Long getId() { return id; }

   public String getTitle() { return title; }
   public void setTitle(String title) { this.title = title; }

   public String getAuthor() { return author; }
   public void setAuthor(String author) { this.author = author; }
       }
```

Анотація @Entity не має аргументів, тому сутності цього типу будуть автоматично зберігатись в таблиці BOOK. Кожне поле або властивість сутності мають бути зіставлені з конкретною колонкою в цій таблиці. Якщо розробник не задає явного зіставлення, то імена колонок також визначаються за замовчуванням - вони будуть збігатись з назвами відповідних полів або властивостей класу. У даному випадку дані будуть збережені в таблиці BOOK, а стовпці матимуть такі назви:

ID для поля id
TITLE для поля title
AUTHOR для поля author

Для виконання операцій над сутностями потрібні відповідні виклики API. Для керування персистентністю сутностей використовується *Entity Manager*, який майже повністю інкапсульований у єдиному інтерфейсі: jakarta.persistence.EntityManager. Коли менеджер сутностей отримує посилання на сутність через параметр методу або в результаті читання з бази даних, ця сутність стає керованою (*managed*). Контекст персистентності (*persistence context*) – це набір керованих сутностей в одному екземплярі EntityManager. У контексті персистентності не може існувати двох екземплярів однієї сутності з однаковим ідентифікатором. Наприклад, якщо у persistence context вже є сутність Book з ID 158, то жоден інший об'єкт Book з таким самим ID не може бути присутнім.

У данному проєкті Менеджер сутностей реалізовано у файлі Main:

```
import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.Persistence;
import java.sql.DriverManager;
import java.sql.SQLException;


public class Main {
    public static void main(String[] args) {
        try {
            EntityManagerFactory emf = Persistence.createEntityManagerFactory("my-persistence-unit");
            EntityManager em = emf.createEntityManager();

            Book book = new Book();
            book.setTitle("Java Persistence with Derby");
            book.setAuthor("Hibernate Team");

            em.getTransaction().begin();
            em.persist(book);
            em.getTransaction().commit();

            Book savedBook = em.find(Book.class, book.getId());
            System.out.println("Saved book: " + savedBook.getTitle() + " by " + savedBook.getAuthor());

            em.close();
            emf.close();

        } finally {
            // Завершення роботи Derby
            try {
                DriverManager.getConnection("jdbc:derby:testDB;shutdown=true");
            } catch (SQLException e) {
                // Очікуваний виняток при shutdown
                System.out.println("Derby shutdown: " + e.getMessage());
            }
        }
    }
       }
```

Для використання технології Jakarta Persistence необхідно додати відповідні налаштування до файлу persistence.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="3.0"
    xmlns="https://jakarta.ee/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
    https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd">

    <persistence-unit name="my-persistence-unit" transaction-type="RESOURCE_LOCAL">
        <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
        <class>ua.edu.znu.lab.Book</class>
        <properties>
            <property name="jakarta.persistence.jdbc.driver"
value="org.apache.derby.jdbc.EmbeddedDriver"/>
            <property name="jakarta.persistence.jdbc.url" value="jdbc:derby:testDB;create=true"/>
            <property name="jakarta.persistence.jdbc.user" value="app"/>
            <property name="jakarta.persistence.jdbc.password" value="app"/>
            <!-- Автоматичне створення таблиць -->
            <!-- property name="jakarta.persistence.schema-generation.database.action" value="drop-and-
create"/ -->
            <property name="jakarta.persistence.schema-generation.database.action" value="create"/>
            <property name="eclipselink.logging.level" value="FINE"/>
            <property name="eclipselink.target-database" value="Derby"/>

        </properties>
    </persistence-unit>
       </persistence>
```

Завдання

1. Реалізуйте приклад наданий у методичних вказівках.
2. Наведіть у звіті скриншоти отриманого результату.
3. За допомогою інструментів Derby перевірте створення відповідної таблиці та полів.
4. Модифікуйте проєкт, представлений у лабораторній роботі 5 на використання технології Jakarta Persistence.
5. Підготуйте звіт.