ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ

Навчально-науковий інститут денної освіти

Форма навчання денна

Кафедра комп’ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)
«___»_____202_ р.

КВАЛІФІКАЦІЙНА РОБОТА
на тему
РОЗРОБКА НАВЧАЛЬНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ТЕМИ «ОСНОВИ МОВИ ЗАПИТІВ HQL» ДИСЦИПЛІНИ «ТЕХНОЛОГІЇ ОБРОБКИ ТА АНАЛІЗУ ДАНИХ»

зі спеціальності 122 Комп’ютерні науки освітня
програма «Комп’ютерні науки» ступеня магістра
Виконавець роботи Шаповал Павло Вікторович
_____«____»_____ 202_ р.
(підпис)
Науковий керівник к. ф.-м. н., Олексійчук Юрій Федорович
_____«____»_____ 202_ р.
(підпис)

Рецензент

ПОЛТАВА 2025

ЗАТВЕРДЖУЮ
Завідувач кафедри_____Олена ОЛЬХОВСЬКА
« ___ » вересня 202_ р.

ЗАВДАННЯ ТА КАЛЕНДАРНИЙ ГРАФІК
ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему « Розробка навчального програмного забезпечення з теми «Основи мови запитів HQL» дисципліни «Технології обробки та аналізу даних»»
зі спеціальності 122 «Комп'ютерні науки»
освітня програма «Комп'ютерні науки»
ступеня магістр
Прізвище, ім'я, по батькові _Шаповал Павло Вікторович_
Затверджена наказом ректора № ____-Н від « » _____ 202_ р.
Термін подання студентом роботи «__ » _____ 202_ р.
Вихідні дані до кваліфікаційно роботи: публікації з теми, навчальні тренажери в дистанційних курсах з комп'ютерних наук.
Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Так, абсолютно вірно. У документ "Завдання" (пункт "Зміст пояснювальної записки") вставляється саме перелік розділів і підрозділів без сторінок і крапок. Ось ваш відформатований зміст, готовий до вставки:
ВСТУП
1. ПОСТАНОВКА ЗАДАЧІ
1.1. Аналіз проблематики вивчення ORM-технологій та мови HQL
1.2. Визначення вимог до функціоналу навчального програмного забезпечення
1.3. Формулювання вимог до інтерфейсу та ергономіки навчальної програми
2. ІНФОРМАЦІЙНИЙ ОГЛЯД
2.1. Огляд існуючих платформ та програм для вивчення баз даних
2.2. Аналіз недоліків існуючих рішень у контексті вивчення HQL
2.3. Обґрунтування вибору засобів розробки та технологічного стеку
3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Теоретичні основи мови запитів HQL та об'єктно-реляційного відображення
3.2. Проектування структури бази даних навчального курсу
3.3. Розробка алгоритму роботи навчальної програми та механізму перевірки відповідей
3.4. Побудова логічної структури та схеми переходів застосунку
4. ПРАКТИЧНА ЧАСТИНА
4.1. Опис процесу програмної реалізації
4.2. Опис програми
4.2.1. Реалізація моделі даних
4.2.2. Логіка ініціалізації та виконання запитів
4.3. Перевірка валідності та дослідження можливостей
4.4. Інструкція користувача
4.4.1. Робота з теоретичним матеріалом
4.4.2. Виконання практичних завдань
4.4.3. Перегляд статистики

ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК А. КОД ПРОГРАМИ
Перелік графічного матеріалу: 1 аркуш блок-схем, 2-3 аркуші графічного матеріалу, інші необхідні ілюстрації.
Консультанти розділів кваліфікаційної роботи

Розділ	ПІП, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постанова задачі	Олексійчук Ю.Ф.		
Інформаційний огляд	Олексійчук Ю.Ф.		
Теоретична частина	Олексійчук Ю.Ф.		
Практична реалізація	Олексійчук Ю.Ф.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
1. Вступ		
2. Вивчення методичних рекомендацій та стандартів та звіт керівнику		
3. Постановка задачі		
4. Інформаційний огляд джерел бібліотек та інтернету		
5. Теоретична частина		
6. Практична частина		
7. Закінчення оформлення		
8. Доповідь студента на кафедрі		
9. Доробка (за необхідністю), рецензування		

Дата видачі завдання « __» _____ 202_ р.
Здобувач вищої освіти _____ Шаповал Павло Вікторович
Науковий керівник_____ к. ф.-м. н., Олексійчук Юрій Федорович
Результати захисту кваліфікаційної роботи
Кваліфікаційна робота оцінена на _____
(балів, оцінка за національною шкалою, оцінка за ECTS)
Протокол засідання ЕК № ____ від «____» _____ 202_ р.
Секретар ЕК_____ _____
(підпис) (ініціал та прізвище)

Затверджую
Зав. кафедрою _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА
«__» _____ 202_ р.

Погоджено
Науковий керівник_____
_____
«__» _____ 202_ р.

План

кваліфікаційної роботи ступеня магістр
зі спеціальності 122 Комп'ютерні науки
освітня програма 122 Комп'ютерні науки
Шаповал Павло Вікторович
Прізвище, ім'я, по батькові

на « Розробка навчального програмного забезпечення з теми «Основи мови запитів HQL» дисципліни «Технології обробки та аналізу даних»»

ВСТУП
1. ПОСТАНОВКА ЗАДАЧІ
1.1. Аналіз проблематики вивчення ORM-технологій та мови HQL
1.2. Визначення вимог до функціоналу навчального програмного забезпечення
1.3. Формулювання вимог до інтерфейсу та ергономіки навчальної програми
2. ІНФОРМАЦІЙНИЙ ОГЛЯД
2.1. Огляд існуючих платформ та програм для вивчення баз даних
2.2. Аналіз недоліків існуючих рішень у контексті вивчення HQL
2.3. Обґрунтування вибору засобів розробки та технологічного стеку
3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Теоретичні основи мови запитів HQL та об'єктно-реляційного відображення
3.2. Проектування структури бази даних навчального курсу
3.3. Розробка алгоритму роботи навчальної програми та механізму перевірки відповідей
3.4. Побудова логічної структури та схеми переходів застосунку
4. ПРАКТИЧНА ЧАСТИНА
4.1. Опис процесу програмної реалізації
4.2. Опис програми
4.2.1. Реалізація моделі даних
4.2.2. Логіка ініціалізації та виконання запитів
4.3. Перевірка валідності та дослідження можливостей
4.4. Інструкція користувача
4.4.1. Робота з теоретичним матеріалом
4.4.2. Виконання практичних завдань
4.4.3. Перегляд статистики
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТОК А. КОД ПРОГРАМИ

Здобувач вищої освіти _____ Павло ШАПОВАЛ
«____» _____ 202_ р.

РЕФЕРАТ

Записка: 67 с., 14 рис., 15 джерел, 1 додаток.

JAVA, HIBERNATE, HQL, ORM, JAVAFX, НАВЧАЛЬНА ПРОГРАМА, БАЗА ДАНИХ, ІНТЕРАКТИВНЕ НАВЧАННЯ.

Об’єкт розробки – навчальне програмне забезпечення для вивчення основ мови запитів HQL та технології об’єктно-реляційного відображення.

Мета роботи – підвищення ефективності підготовки фахівців шляхом проектування та програмної реалізації інтерактивного засобу, який поєднує теоретичний матеріал з практичним виконанням запитів до бази даних.

Методи дослідження – системний аналіз аналогів, об’єктно-орієнтоване проектування для побудови архітектури, алгоритмічне моделювання для створення механізму валідації, тестування програмного забезпечення.

У роботі здійснено аналіз існуючих підходів до вивчення баз даних та виявлено недоліки наявних платформ у контексті вивчення ORM-технологій. Обґрунтовано вибір технологічного стеку: мова Java, бібліотека Hibernate, СКБД H2 та платформа JavaFX.

Спроектовано структуру навчальної бази даних предметної області «Університет». Розроблено алгоритм перевірки практичних завдань, який включає синтаксичний аналіз та логічну валідацію результатів запиту.

Програмно реалізовано десктопний додаток, що містить три функціональні модулі: теоретичний блок, практичний модуль із 10 завдань різної складності та панеллю навігації, а також модуль статистики успішності. Створено зручний інтерфейс користувача з редактором коду та табличною візуалізацією даних.

ЗМІСТ

ВСТУП

Стрімкий розвиток інформаційних технологій у сучасному світі диктує нові вимоги до підготовки фахівців у галузі комп'ютерних наук. Одним із ключових аспектів професійної компетенції розробника програмного забезпечення є вміння ефективно працювати з великими масивами даних та проектувати складні інформаційні системи. Особливого значення набуває вивчення дисципліни «Технології обробки та аналізу даних», яка закладає фундамент розуміння архітектури сучасних сховищ даних та методів взаємодії з ними.

У сучасній практиці розробки корпоративних додатків спостерігається стійка тенденція до використання об'єктно-реляційного відображення (ORM), де провідну роль займає фреймворк Hibernate [1]. Важливою складовою цієї технології є мова запитів HQL (Hibernate Query Language), яка дозволяє розробникам оперувати даними на рівні об'єктів класів, а не таблиць реляційної бази даних. Проте процес освоєння HQL супроводжується значними труднощами для студентів, оскільки вимагає фундаментальної зміни парадигми мислення — переходу від роботи з реляційними структурами до об'єктно-орієнтованих конструкцій.

Традиційні методи навчання, що базуються на читанні технічної документації, часто не забезпечують належного рівня практичної підготовки. В умовах цифровізації освіти та поширення дистанційних форм навчання, виникає гостра потреба у створенні спеціалізованих програмних інструментів, які б дозволяли студентам закріплювати теоретичні знання шляхом активної практичної взаємодії. Саме навчальні програми стають оптимальним рішенням, оскільки вони поєднують у собі наочність, інтерактивність та можливість миттєвого отримання зворотного зв'язку про правильність виконання завдань.

Актуальність даної теми обумовлена необхідністю модернізації освітнього процесу та створення ефективного інструментарію для вивчення складних аспектів обробки даних у рамках курсу «Технології обробки та аналізу даних». Розробка спеціалізованого застосунку дозволить

автоматизувати контроль знань, підвищити залученість студентів та забезпечити гнучкість процесу самостійного навчання.

Мета кваліфікаційної роботи полягає у проектуванні та програмній реалізації навчальної програми з теми «Основи мови запитів HQL», яка забезпечить ефективне засвоєння синтаксису мови та логіки об'єктних запитів студентами спеціальності «Комп'ютерні науки».

Об’єкт дослідження — процес вивчення сучасних технологій обробки та аналізу даних у рамках дистанційних та самостійних курсів.

Предметом дослідження є методи та програмні засоби підтримки вивчення об’єктно-орієнтованих мов запитів до баз даних.

Завдання роботи:

1) Проаналізувати особливості вивчення ORM-технологій та існуючі програмні засоби для підтримки навчання баз даних.
2) Сформулювати вимоги до функціоналу та інтерфейсу навчального програмного забезпечення для вивчення HQL.
3) Розробити структуру навчального контенту (теоретичні блоки та набір різнорівневих практичних завдань) для наповнення програми.
4) Спроектувати архітектуру програмного засобу та схему навчальної бази даних.
5) Виконати програмну реалізацію клієнтської частини та модулів валідації запитів мовою Java.
6) Провести тестування розробленого програмного забезпечення та оцінити його працездатність.

Новизна роботи полягає у розробці спеціалізованого середовища для вивчення HQL, що інтегрує в собі візуальні підказки синтаксису та систему поетапної перевірки результатів, адаптовану під вимоги навчального плану конкретної дисципліни.

Практичне значення отриманих результатів складає можливість інтеграції програмного продукту в систему дистанційного навчання університету, що дозволить покращити якість підготовки фахівців та оптимізувати час викладача на перевірку практичних робіт.

1. ПОСТАНОВКА ЗАДАЧІ

Процес розробки будь-якого програмного продукту, орієнтованого на освітню сферу, вимагає чіткого визначення проблематики, формування детальних вимог до функціонала та проектування взаємодії користувача із системою. У цьому розділі проведено аналіз предметної області, обґрунтовано необхідність створення спеціалізованого програмного забезпечення для вивчення мови запитів HQL та сформульовано технічне завдання на розробку.

1.1 Аналіз проблематики вивчення ORM-технологій та мови HQL

Сучасна парадигма розробки корпоративних інформаційних систем базується на об’єктно-орієнтованому підході, де дані представляються у вигляді об’єктів та класів. Водночас, переважна більшість промислових сховищ даних залишаються реляційними, оперуючи таблицями та зв’язками між ними. Ця розбіжність породжує так званий «об’єктно-реляційний розрив» (Object-Relational Impedance Mismatch) [4], для подолання якого використовуються ORM-фреймворки, зокрема Hibernate.

Навчання студентів технологіям ORM супроводжується значними когнітивними труднощами. Головна проблема полягає в інерції мислення: студенти, які попередньо вивчали класичну мову SQL, намагаються застосовувати реляційну логіку до об’єктних запитів. Мова HQL (Hibernate Query Language), хоча і є синтаксично подібною до SQL, має принципово іншу семантику. Вона оперує сутностями (Entities) та їх атрибутами, підтримує поліморфізм, спадкування та асоціації, що не має прямих аналогів у реляційній алгебрі.

Аналіз існуючих методичних підходів показав, що вивчення HQL часто обмежується теоретичним оглядом документації або аналізом фрагментів коду на лекціях. Відсутність інтерактивного інструментарію призводить до того, що студенти не отримують практичних навичок написання запитів до моменту початку роботи над реальними проектами, де ціна помилки є значно вищою.

Більшість доступних онлайн-тренажерів фокусуються виключно на SQL, ігноруючи специфіку Java-технологій та Hibernate.

Отже, існує нагальна потреба у створенні програмного засобу, який би імітував середовище розробки, але був спрощений для навчальних цілей. Такий додаток повинен не лише перевіряти синтаксичну правильність запиту, а й пояснювати логіку його виконання, демонструючи зв’язок між написаним кодом HQL та згенерованим SQL-запитом, що виконується в базі даних. Це дозволить сформувати у здобувачів освіти глибоке розуміння механізмів роботи ORM та підвищити якість підготовки фахівців з дисципліни «Технології обробки та аналізу даних».

1.2. Визначення вимог до функціоналу навчального програмного забезпечення

Виходячи з аналізу предметної області та потреб навчального процесу, програмний комплекс повинен реалізовувати повний цикл навчання: від ознайомлення з теорією до практичного закріплення навичок та контролю знань. Функціональна структура навчальної програми має бути побудована за модульним принципом, що забезпечить гнучкість використання та можливість подальшого розширення.

Першочерговою вимогою є реалізація теоретичного модуля. Він повинен містити структурований лекційний матеріал, поділений на логічні блоки (теми). Важливо передбачити можливість форматування тексту, відображення фрагментів коду з підсвіткою синтаксису та наявність ілюстрацій (діаграм класів, схем баз даних). Система навігації має дозволяти користувачеві вільно переміщуватися між темами, повертатися до попереднього матеріалу та використовувати пошук по ключових словах.

Центральним елементом системи є практичний модуль (програма). Він повинен надавати користувачеві інтерактивне середовище для виконання завдань. Кожне завдання має супроводжуватися чітким формулюванням умови, описом структури класів-сутностей, з якими необхідно працювати, та очікуваним результатом. Критично важливою функцією є наявність

вбудованого редактора коду, який підтримує базові функції IDE, такі як нумерація рядків та, за можливості, автодоповнення або підсвітка ключових слів HQL.

Алгоритм перевірки має базуватися на засобах валідації бібліотеки Hibernate, що дозволяє виявляти синтаксичні помилки на етапі компіляції запиту, та подальшому динамічному виконанні коду на тестовій базі даних для перевірки логічної правильності результату. Система перевірки рішень повинна працювати в режимі реального часу. У разі помилки програма повинна надавати інформативне повідомлення, яке не просто констатує факт невдачі, а вказує на ймовірну причину (наприклад, «Невідоме поле класу» або «Невірний тип повертаних даних»).

Додатковою вимогою є реалізація системи оцінювання та статистики. Програма повинна зберігати прогрес користувача, фіксуючи кількість пройдених тем, успішність виконання практичних завдань та час, витрачений на проходження тестів. Це дозволить студенту самостійно контролювати свій рівень підготовки, а викладачеві (у разі впровадження режиму адміністратора) — моніторити успішність групи.

1.3. Формулювання вимог до інтерфейсу та ергономіки навчальної програми

Ергономіка програмного забезпечення відіграє вирішальну роль у процесі навчання, оскільки незручний або перевантажений інтерфейс може відволікати студента від засвоєння матеріалу. Проектування графічного інтерфейсу користувача (GUI) повинно базуватися на принципах мінімалізму, інтуїтивності та доступності.

Вимоги до візуального оформлення включають використання стриманої кольорової гами, яка не викликає зорової втоми при тривалій роботі. Рекомендується використання темних тонів для редактора коду (аналогічно до популярних середовищ розробки, таких як IntelliJ IDEA) та світлих тонів для текстових блоків теорії. Шрифт повинен бути розбірливим, з можливістю масштабування для користувачів з вадами зору.

Компонування робочої області додатку має бути логічним та функціональним. Екран доцільно розділити на кілька зон:

1) Зона навігації: забезпечує швидкий доступ до списку завдань та теоретичних розділів.
2) Інформаційна зона: відображає умову поточної задачі та структуру даних (діаграму сутностей).
3) Робоча зона (Редактор): місце для введення HQL-запитів.
4) Зона результатів: вікно для виведення отриманої вибірки даних або повідомлень про помилки.

Важливою ергономічною вимогою є адаптивність інтерфейсу до різних роздільних здатностей екранів, оскільки студенти можуть використовувати як стаціонарні монітори в лабораторіях університету, так і ноутбуки з меншою діагоналлю екрану вдома. Всі елементи управління (кнопки «Перевірити», «Далі», «Підказка») повинні бути чітко видимими, мати зрозумілі піктограми або текстові підписи та розташовуватися у звичних для користувача місцях.

Окрему увагу слід приділити системі зворотного зв’язку. Будь-яка дія користувача (завантаження завдання, перевірка відповіді, збереження прогресу) повинна супроводжуватися візуальною індикацією, щоб уникнути відчуття «зависання» програми.

Таким чином, сформульовані вимоги до функціонала та інтерфейсу є основою для подальшого проектування архітектури та програмної реалізації навчального програмного засобу, що буде детально розглянуто у наступних розділах.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

Ефективне проектування та розробка будь-якого сучасного програмного засобу, орієнтованого на освітню сферу, є складним процесом, який неможливий без попереднього глибокого та всебічного аналізу існуючих рішень, а також детального вивчення технологічного ландшафту. Цей етап наукового дослідження дозволяє виявити сильні та слабкі сторони наявних на ринку продуктів, уникнути типових помилок при проектуванні власної системи та, що найважливіше, обгрунтовано підійти до вибору інструментарію розробки. У даному розділі проведено детальний огляд популярних платформ для вивчення баз даних, виокремлено їхні суттєві недоліки в контексті специфіки мови HQL, а також наведено розширену аргументацію щодо вибору технологічного стеку для реалізації програмного застосунку.

2.1. Огляд існуючих платформ та програм для вивчення баз даних

Сучасний ринок освітніх технологій (EdTech) пропонує широкий спектр програмних інструментів та онлайн-сервісів для вивчення програмування та роботи з даними. Детальний аналіз доступних ресурсів дозволяє класифікувати їх на кілька основних категорій: інтерактивні онлайн-курси з вбудованими редакторами коду, платформи для розв’язання алгоритмічних задач (competitive programming) та спеціалізовані SQL-тренажери.

Одним із найвідоміших та найпопулярніших ресурсів у цій сфері є платформа SQLZoo [8]. Цей веб-сервіс надає користувачам можливість інтерактивного навчання мові SQL (Structured Query Language) без необхідності встановлення будь-якого додаткового програмного забезпечення. Навчальний процес на SQLZoo побудований за принципом «від простого до складного»: користувачам пропонується набір таблиць, що імітують реальні бази даних (наприклад, база даних країн світу або база даних нобелівських лауреатів), та послідовність практичних завдань (рис. 2.1).

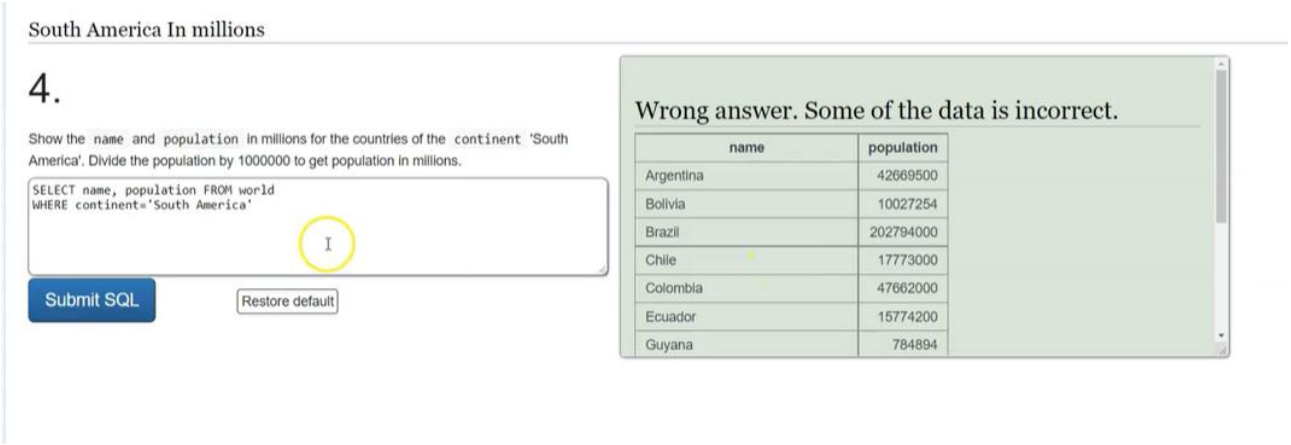

Рисунок 2.1 – Інтерфейс платформи SQLZoo

Безумовною перевагою SQLZoo є його простота, доступність та наочність: студент бачить схему бази даних, вводить запит у текстове поле і миттєво отримує результат виконання у вигляді таблиці. Проте, ключовим обмеженням цієї платформи в контексті теми нашої кваліфікаційної роботи є її виключна орієнтація на реляційну модель даних. SQLZoo вчить працювати з таблицями, рядками та стовпцями, повністю ігноруючи об’єктно-орієнтований підхід, який лежить в основі мови HQL та сучасних ORM-технологій.

Іншим популярним прикладом, який варто розглянути, є платформа LeetCode [9] (а також її численні аналоги, такі як HackerRank, CodeWars). Ці ресурси здебільшого фокусуються на підготовці розробників до технічних співбесід у великі технологічні компанії і пропонують величезну базу алгоритмічних задач, у тому числі задач на проектування та написання запитів до баз даних

Система автоматичної перевірки рішень на таких платформах є досить досконалою та високотехнологічною: вона порівнює результат виконання запиту користувача з еталонним результатом на різних наборах тестових даних, враховуючи час виконання та використання пам'яті. Проте, як і у випадку з SQLZoo, завдання тут формулюються виключно діалектами мови SQL (MySQL, PostgreSQL, Oracle SQL). Студент, який намагається опанувати бібліотеку Hibernate, не знайде тут підтримки специфічного синтаксису HQL, такого як навігація по асоціаціях через крапку, використання імен класів замість імен таблиць або робота з поліморфними запитами.

Також доцільно згадати популярний освітній ресурс W3Schools, який є своєрідним стандартом для початківців у веб-розробці. Цей ресурс містить великий розділ, присвячений мові SQL, з вбудованим редактором «Try it Yourself». Цей інструмент дозволяє експериментувати з кодом безпосередньо у браузері. Однак, W3Schools надає лише базові знання синтаксису і не заглиблюється в особливості взаємодії об'єктно-орієнтованих мов програмування (таких як Java) з базами даних. Відсутність контексту Java-розробки робить цей ресурс малокорисним для вивчення HQL.

Окремо слід виділити освітні платформи широкого профілю, такі як Udemy, Coursera або Pluralsight. Курси, представлені на цих ресурсах, часто містять глибокий теоретичний матеріал по Hibernate та HQL. Зазвичай це відеолекції, записані експертами галузі, які супроводжуються текстовими конспектами та прикладами коду. Головним недоліком такого підходу є переважно пасивний характер навчання. Студент може переглянути години відео, де викладач пише та пояснює код, але без власноручного відтворення та миттєвої автоматичної перевірки помилок рівень засвоєння матеріалу залишається недостатнім. Більшість таких курсів пропонують студентам налаштовувати середовище розробки локально на власному комп'ютері, що для початківця часто стає непереборним бар’єром через складність конфігурації серверів баз даних та систем збірки проектів.

Також варто згадати офіційну документацію Hibernate ORM. Хоча вона є першоджерелом, найбільш авторитетним ресурсом та містить вичерпну інформацію про всі можливості фреймворку, її стиль викладу розрахований насамперед на досвідчених інженерів-практиків. Велика кількість специфічних технічних термінів, складність структури та відсутність простих інтерактивних прикладів роблять офіційну документацію складною для сприйняття студентами, які тільки починають своє знайомство з технологією.

Окрім глобальних платформ, доцільно розглянути досвід розробки подібних систем у закладах вищої освіти України.

Аналіз репозитаріїв закладів вищої освіти України, зокрема Полтавського університету економіки і торгівлі, свідчить про те, що тема розробки

програмних засобів для підтримки навчання є стабільно актуальною. Серед робіт студентів ПУЕТ варто відзначити розробки систем тестування знань з використанням мобільних технологій [13], а також комплексні автоматизовані системи управління навчальним процесом [14]. Водночас, у інших технічних ЗВО (наприклад, НТУУ «КПІ») значна увага приділяється аспектам безпеки баз даних та виявленню вразливостей SQL Injection [15]. Детальний розгляд цих робіт дозволяє виділити два типові підходи до реалізації, які мають певні обмеження у контексті вивчення HQL:

1) Тестова модель. Більшість студентських навчальних програм реалізовані у вигляді класичних тестів, де користувачеві пропонується обрати правильний варіант SQL-запиту зі списку або вставити пропущене слово. Такий підхід перевіряє пасивні знання, але не формує навички написання реального коду.
2) Перевірка за шаблоном (String Matching). У випадках, коли програмний засіб дозволяє вводити код, перевірка часто зводиться до простого порівняння введеного рядка з еталонним текстом. Цей метод є негнучким, оскільки валідні запити SELECT * FROM Table та select * from Table можуть бути розцінені системою як різні через регістр або зайві пробіли.

На відміну від розглянутих аналогів, у даній кваліфікаційній роботі пропонується реалізувати підхід динамічного виконання, де введений студентом HQL-код компілюється та виконується на реальній базі даних H2 в оперативній пам'яті. Це забезпечує об'єктивність перевірки результату незалежно від форматування коду.

2.2. Аналіз недоліків існуючих рішень у контексті вивчення HQL

Узагальнюючи проведений аналіз існуючих рішень та платформ, можна виділити ряд системних недоліків, які унеможливлюють їх ефективне використання для вивчення саме мови HQL у навчальному процесі:

1) Домінування SQL-парадигми. Абсолютна більшість існуючих тренажерів вчить мислити категоріями реляційної алгебри (таблиці, зовнішні ключі, операції об'єднання JOIN). Натомість HQL вимагає принципово іншого

типу мислення — навігації по графу об’єктів. Існуючі тренажери не візуалізують критично важливий зв’язок між класом Java (Entity) та таблицею бази даних, що є ключовим для розуміння суті технології ORM.

2) Відсутність візуалізації об’єктної моделі. У класичних SQL-тренажерах користувач бачить ER-діаграму (схему таблиць та зв'язків між ними). Для ефективного вивчення HQL студенту необхідно бачити UML-діаграму класів, оскільки запити пишуться саме до імен класів та їх полів, а не до таблиць. Відсутність такої візуалізації призводить до плутанини в термінології та помилок у написанні запитів.
3) Високий поріг входження та складність налаштування оточення. Щоб почати практикуватися в HQL «вдома», студенту потрібно виконати ряд складних дій: встановити Java Development Kit (JDK), налаштувати систему автоматичної збірки (Maven або Gradle), підключити необхідні бібліотеки Hibernate, встановити та налаштувати драйвер бази даних, а також створити коректний конфігураційний файл hibernate.cfg.xml. Помилка на будь-якому з цих етапів може повністю заблокувати процес навчання ще до моменту написання першого запиту.
4) Брак зрозумілого зворотного зв’язку. При виконанні запитів у стандартних середовищах розробки помилки часто виводяться у вигляді довгих та незрозумілих стектрейсів (Stack Trace), які важко читати та аналізувати новачкам. Спеціалізований навчальний застосунок повинен "перекладати" ці технічні помилки на зрозумілу мову, надаючи студенту підказки щодо шляхів їх виправлення.

Таким чином, можна констатувати, що на сьогоднішній день існує незаповнена ніша програмних інструментів, які б дозволяли вивчати HQL в ізольованому, спрощеному та дружньому до користувача середовищі без необхідності складного попереднього налаштування інфраструктури.

2.3. Обґрунтування вибору засобів розробки та технологічного стеку

Вибір програмних засобів для реалізації кваліфікаційної роботи базувався на детальному аналізі вимог до функціональності, надійності, швидкодії та відповідності темі дослідження. Було обрано наступний технологічний стек.

Мова програмування: Java Вибір мови програмування Java є безальтернативним [3,5] та найбільш логічним рішенням для даного проекту. По-перше, мова запитів HQL (Hibernate Query Language) є невід'ємною частиною фреймворку Hibernate, який, у свою чергу, написаний на Java і призначений виключно для розробки Java-додатків. Спроба створення навчального програмного засобу HQL на будь-якій іншій мові (наприклад, C#, Python або JavaScript) вимагала б створення складних технологічних «мостів» або повної емуляції роботи парсера запитів, що є технічно нераціональним та трудомістким. По-друге, Java забезпечує сувору статичну типізацію, автоматичне управління пам'яттю (Garbage Collection) та потужні засоби об’єктно-орієнтованого програмування, що значно спрощує розробку складної внутрішньої логіки аналізу та перевірки запитів користувача.

Бібліотека ORM: Hibernate Оскільки основною метою роботи є створення навчальної програми саме по мові HQL, використання оригінальної бібліотеки Hibernate ORM є обов’язковою умовою [2]. Це дозволяє гарантувати, що запити, які виконує студент у додатку, будуть оброблятися та виконуватися абсолютно ідентично до того, як вони працювали б у реальному комерційному проекті. Використання реального ядра Hibernate дозволяє уникнути неточностей та розбіжностей, які могли б виникнути при спробі написати власний інтерпретатор мови запитів.

Система управління базами даних: H2 Database. Для забезпечення повноцінної роботи застосунку необхідна наявність реляційної бази даних. Використання "важких" серверних рішень, таких як MySQL, PostgreSQL або Oracle, суперечить вимогам до мобільності, автономності та простоти встановлення навчального програмного забезпечення. Саме тому було обрано H2 Database [7]. Це швидка реляційна СУБД, повністю написана на Java, яка може працювати в режимі "In-Memory" (безпосередньо в оперативній пам’яті комп'ютера). Ключові переваги H2 для даного проекту включають:

1) Відсутність необхідності встановлення та налаштування окремого сервера баз даних.

2) База даних створюється миттєво при кожному запуску програми і автоматично очищується при виході, що є ідеальним сценарієм для навчального тестування.
3) Висока швидкість роботи завдяки відсутності операцій вводу-виводу на жорсткий диск.
4) Повна сумісність з Hibernate та підтримка стандарту SQL.

Середовище розробки: IntelliJ IDEA Як інтегроване середовище розробки (Integrated Development Environment - IDE) обрано IntelliJ IDEA від компанії JetBrains. На сьогоднішній день це визнаний стандарт де-факто в індустрії професійної Java-розробки. Середовище надає розробнику потужні інструменти для інтелектуальної роботи з кодом, автоматичного рефакторингу, зручного відлагодження (debugging) та інтеграції з системами контролю версій (Git). Наявність вбудованих інструментів для роботи з базами даних та нативна підтримка фреймворку Hibernate значно прискорює та спрощує процес розробки додатку.

Технологія інтерфейсу користувача: JavaFX Для реалізації сучасного графічного інтерфейсу користувача (GUI) обрано платформу JavaFX [6]. На відміну від застарілої бібліотеки Swing, JavaFX дозволяє створювати сучасні, візуально привабливі та адаптивні інтерфейси з використанням декларативної мови розмітки FXML та CSS-стилізації. Використання FXML дає можливість чітко відокремити дизайн інтерфейсу від програмної логіки (реалізуючи патерн MVC/MVVM), що спрощує підтримку та розвиток коду в майбутньому. Крім того, JavaFX забезпечує багатий набір готових візуальних компонентів (таблиці, текстові поля з підтримкою форматування), які необхідні для створення зручного редактора коду та відображення результатів виконання запитів.

Таким чином, обраний технологічний стек (Java + Hibernate + H2 + JavaFX) є збалансованим, сучасним, надійним та повністю відповідає поставленим задачам, забезпечуючи створення ефективного навчального інструменту.

3. ТЕОРЕТИЧНА ЧАСТИНА

Процес розробки програмного забезпечення навчального призначення вимагає ґрунтовного теоретичного підґрунтя, яке включає аналіз предметної області, проектування структури даних та розробку алгоритмів взаємодії системи з користувачем. У даному розділі розглянуто теоретичні основи мови запитів HQL, спроектовано навчальну базу даних, яка використовуватиметься у тренажері, а також детально описано алгоритми перевірки знань та логічну структуру програмного застосунку.

3.1. Теоретичні основи мови запитів HQL та об’єктно-реляційного відображення

В основі роботи розроблюваного додатку лежить технологія об’єктно-реляційного відображення (Object-Relational Mapping — ORM). Це технологія програмування, яка дозволяє перетворювати несумісні типи моделей у об'єктно-орієнтованих мовах програмування, створюючи "віртуальну об'єктну базу даних". Для Java-середовища стандартом де-факто є бібліотека Hibernate, яка реалізує специфікацію JPA (Java Persistence API) [1].

Ключовим інструментом взаємодії з даними в Hibernate є мова запитів HQL (Hibernate Query Language) [2]. На відміну від SQL, який оперує назвами таблиць та колонок у базі даних, HQL є повністю об'єктно-орієнтованою мовою. Вона оперує назвами персистентних класів (Entity) та їхніми властивостями (полями). Принципові відмінності HQL від SQL, які повинні бути враховані при розробці навчального контенту застосунку, наведено нижче:

1) Поліморфізм: Запит HQL, адресований до батьківського класу, автоматично витягує дані з усіх таблиць, що відповідають класам-спадкоємцям.
2) Асоціації: HQL підтримує навігацію по зв'язках об'єктів. Замість явного використання оператора JOIN з умовами ON, розробник може звертатися

до пов'язаних сутностей через крапку (наприклад, student.group.name), що значно спрощує синтаксис.

3) Агрегатні функції: Хоча HQL підтримує стандартні функції (count, sum, avg), результати їх виконання повертаються як об'єкти відповідних типів Java, а не просто як набір байтів.

Загальна структура виконання HQL-запиту в системі виглядає наступним чином: додаток формує запит, передає його до Session (основний інтерфейс Hibernate), після чого ядро бібліотеки транслирує HQL у SQL-діалект конкретної бази даних (у нашому випадку — H2 Database), виконує його і відображає отриманий результат (ResultSet) назад у список об'єктів Java. Цей процес відображено на схемі архітектури (рис. 3.1).

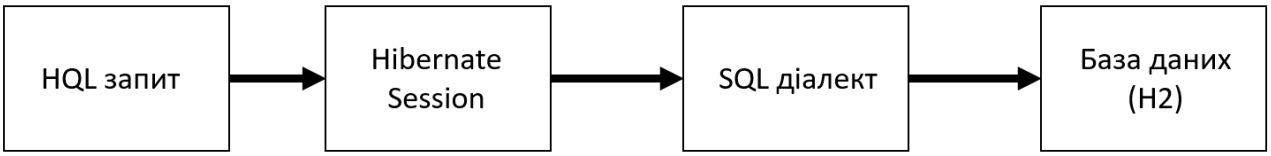

Рисунок 3.1 – Схема трансляції HQL запиту в SQL через Hibernate

Розуміння цього процесу є критично важливим для студента, тому теоретичний блок тренажера повинен містити детальні пояснення механізму трансляції, щоб уникнути поширеної проблеми "N+1 запитів" та неоптимального використання ресурсів бази даних.

3.2. Проектування структури бази даних навчального курсу

Фундаментальним етапом розробки будь-якої інформаційної системи є проектування моделі даних, яка визначає правила зберігання, організації та маніпулювання інформацією. Для забезпечення ефективного навчального процесу в рамках навчальної програми було обрано предметну область «Університет». Вибір саме цієї доменної моделі обумовлений її інтуїтивною

зрозумілістю для цільової аудиторії — студентів, що дозволяє їм зосередитися на вивченні синтаксису HQL, не витрачаючи час на розуміння специфіки даних (як це буває з абстрактними прикладами на кшталт «департаменти та працівники» [11]).

Спроектована архітектура бази даних базується на об'єктно-реляційному підході та складається з чотирьох взаємопов'язаних сутностей (Entities). Програмна реалізація моделі здійснена мовою Java з використанням специфікації JPA. Нижче наведено детальний опис кожної сутності, їх атрибутивного складу та використаних змінних.

1) Сутність Student (Студент). Це центральний клас системи, що відображає інформацію про здобувачів освіти. У програмному коді він представлений класом org.example.Student. Атрибутивний склад класу включає наступні змінні:
   a) private Long id — унікальний ідентифікатор запису (первинний ключ). Для автоматичної генерації значень використано стратегію GenerationType.IDENTITY, що дозволяє базі даних самостійно керувати інкрементом ключів.
   b) private String firstName — змінна рядкового типу для зберігання імені студента.
   c) private String lastName — змінна рядкового типу для зберігання прізвища. Розділення імені та прізвища на два окремих поля дозволяє реалізовувати завдання на складне сортування та пошук.
   d) private Double rating — змінна типу числа з плаваючою комою, що зберігає рейтинг успішності студента. Використання типу-обгортки Double дозволяє виконувати точні математичні операції та використовувати агрегатні функції AVG, MIN, MAX у запитах HQL.
   e) private Group group — об'єктна змінна, що реалізує зв'язок із сутністю Group. Використання анотації @ManyToOne вказує на те, що багато студентів можуть належати до однієї групи. У базі даних цей зв'язок реалізується через колонку зовнішнього ключа group_id.

2) Сутність Group (Академічна група). Клас org.example.Group виконує роль контейнера для об'єднання студентів. Його структура є мінімалістичною для спрощення навчальних прикладів:
   a) private Long id — унікальний ідентифікатор групи.
   b) private String name — змінна рядкового типу, що містить назву групи (наприклад, "CS-21"). Саме до цього поля студенти звертаються при формуванні умов фільтрації у запитах (наприклад, WHERE s.group.name = '...').
   c) private List<Student> students — колекція, що реалізує зворотний бік зв'язку «Один-до-Багатьох» (@OneToMany). Наявність цієї змінної дозволяє демонструвати навігацію по графу об'єктів від групи до списку її студентів.
3) Сутність Course (Навчальний курс). Клас org.example.Course описує навчальні дисципліни, які вивчаються в університеті.
   a) private Long id — ідентифікатор курсу.
   b) private String title — змінна для зберігання назви дисципліни (наприклад, "Java Programming", "Databases"). Ця сутність є незалежним довідником і використовується для формування аналітичних запитів.
4) Сутність Grade (Оцінка). Це асоціативна сутність, яка розгортає зв'язок «Багато-до-Багатьох» між студентами та курсами у два зв'язки «Один-до-Багатьох». У коді представлена класом org.example.Grade:
   a) private Long id — ідентифікатор запису про оцінку.
   b) private Student student — змінна-посилання на об'єкт студента, який отримав оцінку.
   c) private Course course — змінна-посилання на об'єкт курсу, за який отримано оцінку.
   d) private Integer value — цілочисельна змінна, що зберігає саме значення оцінки (бали).

Графічне представлення розробленої структури класів та зв'язків між ними наведено на рисунку 3.2. Ця UML-діаграма є основною візуальною підказкою для студента під час роботи з тренажером.

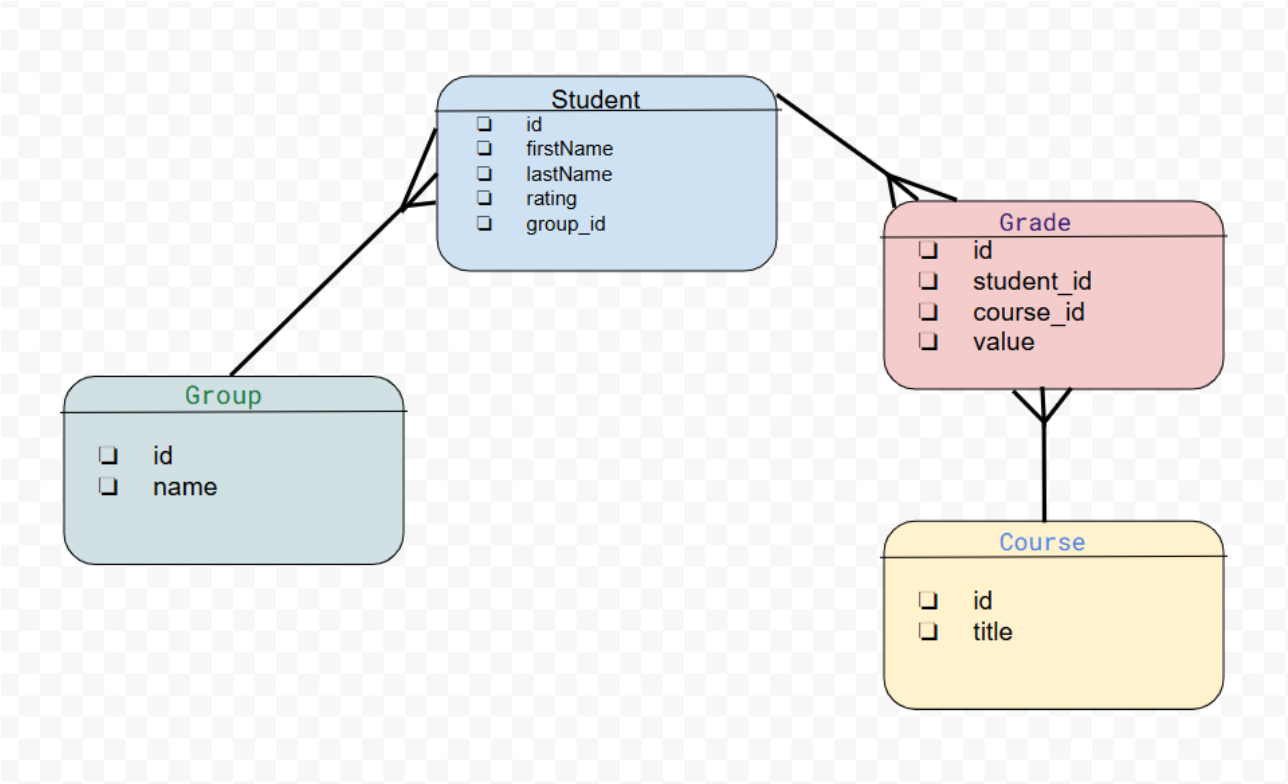

Рисунок 3.2 – UML-діаграма класів предметної області «Університет»

Кожному описаному класу у додатку відповідає таблиця в реляційній базі даних H2, яка створюється автоматично при запуску програми. Налаштування відображення (ORM Mapping) здійснюється за допомогою стандартних анотацій JPA: @Entity для оголошення сутності, @Table для вказання імені таблиці в БД, @Column для налаштування параметрів стовпців.

Така деталізована структура даних дозволяє сформулювати широкий спектр практичних завдань: від простих вибірок ("Знайти всіх студентів з іменем Ivan", звертаючись до змінної firstName) до складних аналітичних запитів з використанням об'єднання таблиць ("Знайти середній бал по групі", використовуючи навігацію через змінні student та rating у сутності Grade)

3.3. Розробка алгоритму роботи навчальної програми та механізму перевірки відповідей

Ключовим елементом архітектури розроблюваного програмного комплексу є підсистема інтерактивної взаємодії з користувачем та автоматизованої перевірки програмного коду. На відміну від статичних

тестових систем, де перевірка здійснюється шляхом простого порівняння вибраних варіантів, даний тренажер реалізує повноцінний цикл виконання HQL-запитів, що наближує навчальний процес до реальних умов розробки програмного забезпечення. Алгоритмічна база додатку побудована на принципах подійно-орієнтованого програмування, що забезпечує гнучку реакцію системи на будь-які дії користувача в режимі реального часу.

Загальна логіка функціонування практичного модуля, графічно представлена на рисунку 3.3, демонструє ітеративний підхід до виконання завдань. Ініціалізація навчального процесу розпочинається з етапу завантаження контексту завдання, під час якого система звертається до внутрішнього сховища даних для отримання формулювання умови поточної задачі, а також еталонного HQL-запиту, який слугуватиме критерієм істини при використанні підказок. Після успішної ініціалізації програма переходить у стан очікування введення, надаючи користувачеві можливість аналізувати умову та формувати власний варіант запиту у відповідному текстовому полі редактора.

Варто зазначити, що розроблений алгоритм передбачає розгалуження сценаріїв поведінки системи залежно від обраної користувачем стратегії вирішення проблеми. Зокрема, у випадках, коли студент стикається з труднощами у формулюванні синтаксичних конструкцій, передбачено механізм використання інтелектуальної підказки. При активації відповідного елемента інтерфейсу система автоматично заповнює робочу область редактора коректним програмним кодом, що дозволяє користувачеві проаналізувати структуру правильного рішення. Після цього управління знову повертається до користувача, дозволяючи йому запустити отриманий код на виконання та наочно побачити результат його роботи.

Основний цикл перевірки знань ініціюється виключно після натискання користувачем кнопки виконання запиту. Цей процес є багатоступеневим і включає в себе декілька рівнів валідації. На першому етапі відбувається передача введеного тексту до ядра ORM-бібліотеки Hibernate, де здійснюється спроба компіляції HQL-запиту. Цей етап є критично важливим, оскільки саме тут відбувається синтаксичний аналіз коду. У разі виявлення синтаксичних

помилок, таких як неправильне написання ключових слів або звернення до неіснуючих полів класів, механізм Hibernate генерує виключення (Exception). Алгоритм програми спроектовано таким чином, щоб перехоплювати ці виключні ситуації, запобігаючи аварійному завершенню роботи додатку. Замість цього система трансформує технічне повідомлення про помилку у зрозумілий для користувача формат та виводить його на екран, після чого повертає процес на етап редагування коду.

У випадку успішного проходження синтаксичного контролю, алгоритм переходить до етапу безпосереднього виконання запиту на підключеній базі даних H2. Отриманий результат, який представляє собою список об'єктів, передається до компонента візуалізації — таблиці результатів. Саме на цьому етапі відбувається оновлення інтерфейсу, що дозволяє студенту побачити реальні дані, які були витягнуті з бази даних за допомогою його запиту.

Фінальним етапом алгоритму є логічна валідація отриманого результату. Система проводить аналіз вибірки даних на відповідність очікуваним критеріям успішності. Важливо підкреслити, що успішним вважається не просто синтаксично правильний запит, а лише той, що повертає релевантний набір даних, який відповідає семантиці поставленого завдання. Якщо результат виконання запиту є пустим або не відповідає умові, система інформує користувача про необхідність повторної спроби, не блокуючи при цьому можливість подальшого редагування коду. Лише за умови повної відповідності отриманих даних еталонним показникам, система фіксує успішне виконання завдання, оновлює статистичні показники успішності студента та активує елементи навігації для переходу до наступного рівня складності.

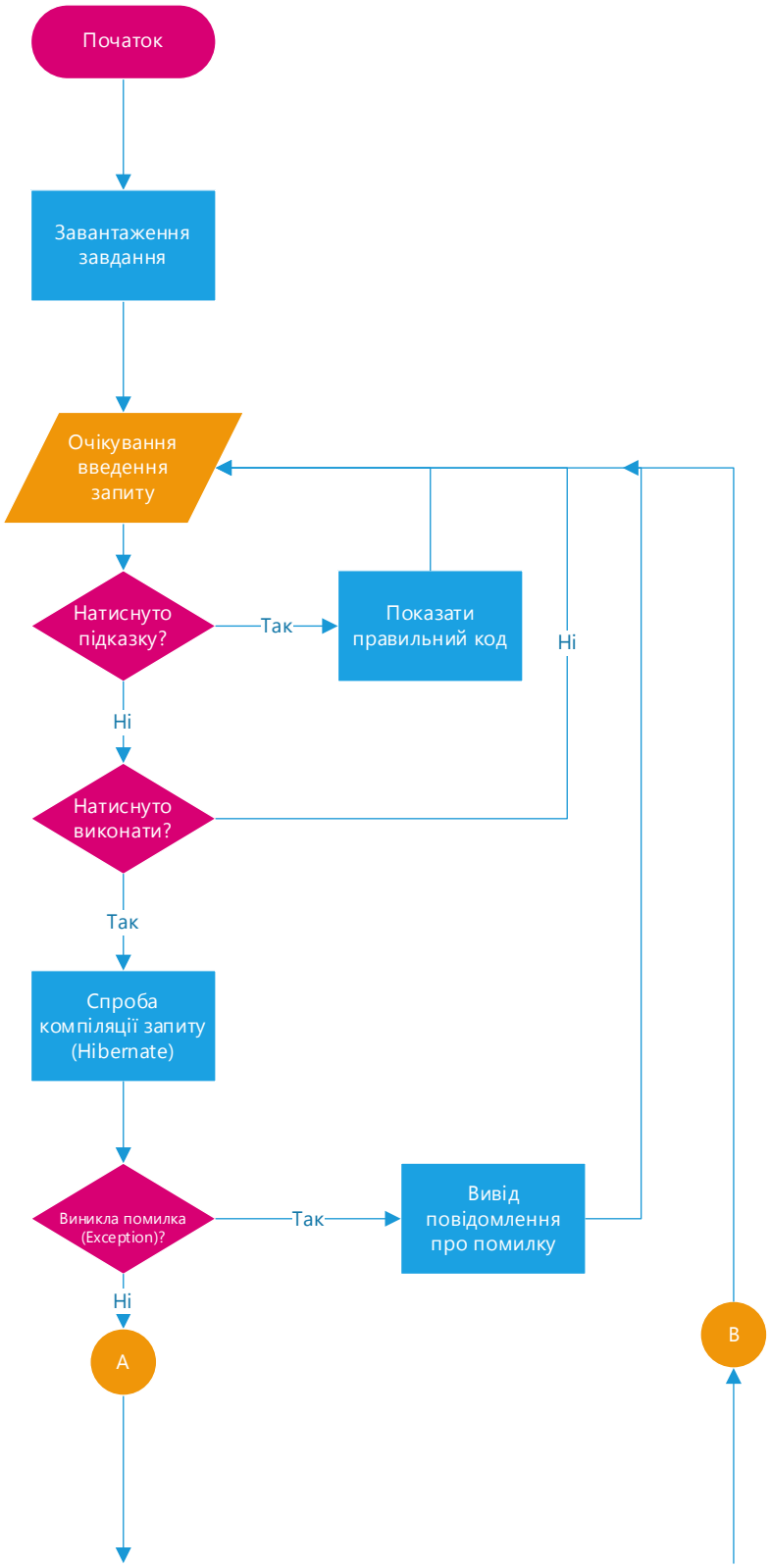

Рисунок 3.3 – Деталізований алгоритм роботи практичного модуля (початок)

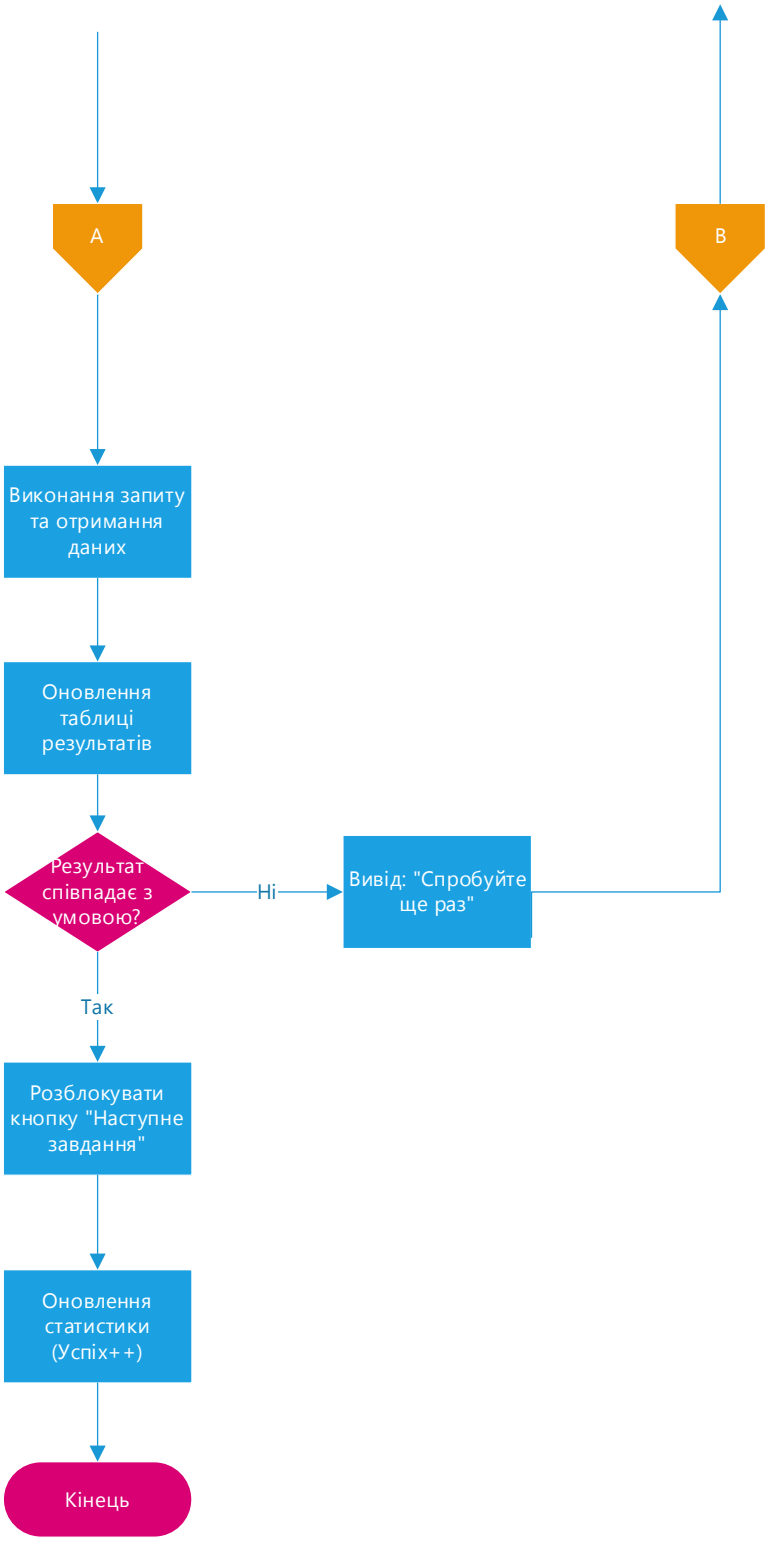

Рисунок 3.3 – Деталізований алгоритм роботи практичного модуля (продовження)

Така циклічна структура алгоритму забезпечує високу ефективність навчального процесу, дозволяючи студенту багаторазово експериментувати з кодом, отримувати миттєвий зворотний зв'язок як щодо синтаксичних помилок,

так і щодо логічної правильності побудованих запитів, що в кінцевому підсумку сприяє глибшому засвоєнню матеріалу.

3.4. Побудова логічної структури та схеми переходів застосунку

Програмна реалізація застосунку виконана з дотриманням принципів об’єктно-орієнтованого проектування та архітектурного патерну, що розділяє логіку відображення (UI) та бізнес-логіку. Структура проекту організована за пакетами:

1) Пакет org.example (Model & View):
    a. Main.java: Головний клас додатка, який відповідає за ініціалізацію графічного інтерфейсу (JavaFX), побудову сцен (Scenes), обробку подій користувача (натискання кнопок, навігація меню) та відображення даних у таблицях і діаграмах.
    b. Сутності (Entities): Класи Student, Group, Course, Grade. Вони є POJO-класами (Plain Old Java Objects), анотованими специфікаціями JPA для відображення у базі даних.
2) Пакет org.example.service (Controller / Service Layer):
    a. DatabaseService.java: Клас, що інкапсулює роботу з бібліотекою Hibernate. Він відповідає за створення фабрики сесій (SessionFactory), ініціалізацію підключення до H2 Database, наповнення бази тестовими даними при старті та виконання довільних HQL-запитів, що надходять від користувача.
    b. TaskService.java: Клас, що містить логіку навчального процесу. Він зберігає список об'єктів Task (завдання), кожен з яких містить умову, еталонний запит та логіку валідації. Валідація реалізована з використанням функціональних інтерфейсів Java (Predicate), що дозволяє перевіряти як текст запиту (на наявність ключових слів), так і отриманий результат (на кількість записів, відповідність значень тощо).

Така архітектура забезпечує слабку зв'язність компонентів (Loose Coupling). Наприклад, зміна логіки перевірки завдань у TaskService не потребує змін у класі інтерфейсу Main.

Взаємодія компонентів відбувається наступним чином: користувач взаємодіє з Main, який передає введені дані у DatabaseService. Отриманий результат повертається у Main для відображення та паралельно передається у TaskService для перевірки правильності рішення.

Для забезпечення інтуїтивно зрозумілої навігації та зручності користування (Usability) було розроблено логічну структуру застосунку, яка базується на ієрархічному принципі. Програма складається з кількох функціональних блоків, перехід між якими здійснюється через головне меню.

Основні структурні елементи програмного засобу:

1) Головне меню: Центральний хаб, що надає доступ до трьох основних модулів: «Теорія», «Практика», «Статистика».
2) Модуль «Теорія»: Містить список лекційних тем. При виборі теми відкривається вікно перегляду контенту з можливістю навігації «Назад»/«Далі».
3) Модуль «Практика»: Містить список рівнів складності. При виборі рівня відкривається вікно тренажера. З вікна додатку можливий перехід до наступного завдання або повернення до списку.
4) Модуль «Статистика»: Відображає прогрес користувача (кількість пройдених тем, відсоток правильних відповідей).

Схема переходів між екранними формами (Screen Flow Diagram) зображена на рисунку 3.4. Вона демонструє всі можливі шляхи користувача в системі та логіку взаємодії інтерфейсних компонентів.

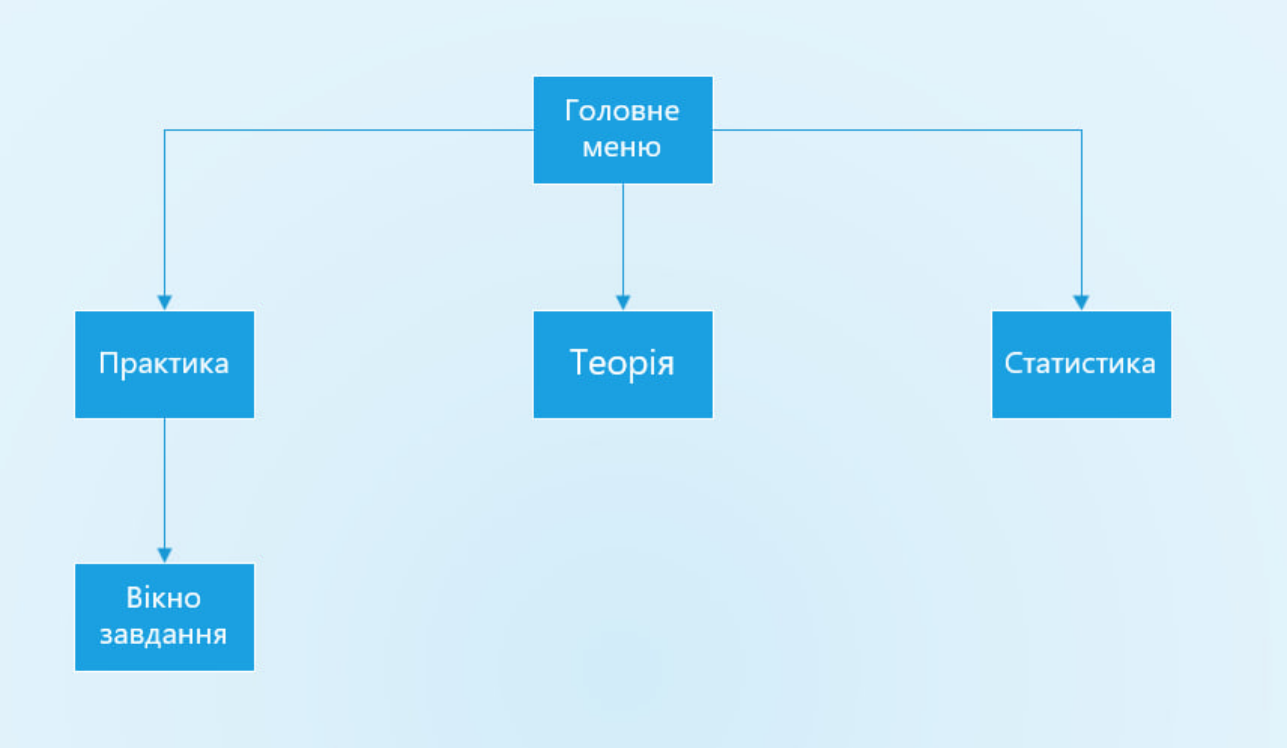

Рисунок 3.4 – Схема переходів між екранами програмного застосунку

Розроблена логічна структура забезпечує замкненість циклу навчання: студент може ознайомитися з теорією, одразу перейти до практики, перевірити свої знання і, в разі невдачі, повернутися до теоретичного матеріалу для повторення. Така архітектура сприяє кращому засвоєнню матеріалу та підвищує мотивацію до навчання.

4. ПРАКТИЧНА ЧАСТИНА

4.1. Опис процесу програмної реалізації

Реалізація програмного комплексу «HQL Trainer» виконувалася відповідно до принципів об’єктно-орієнтованого програмування [10] з використанням мови Java. Вибір цього інструментарію обумовлений необхідністю нативної підтримки технології Hibernate ORM, яка є ключовим об’єктом вивчення в рамках розроблюваного тренажера.

В якості інтегрованого середовища розробки (IDE) було обрано IntelliJ IDEA Community Edition. Це середовище надає розширені можливості для роботи з системою збірки Maven, управління залежностями та налагодження коду. Структура проекту була організована відповідно до стандартів Maven, що забезпечує чітке розділення вихідного коду, ресурсів та конфігураційних файлів.

На рисунку 4.1 зображено загальну структуру проекту в середовищі IntelliJ IDEA. Коренева директорія містить файл опису залежностей pom.xml, а вихідний код розміщено в директорії src/main/java.

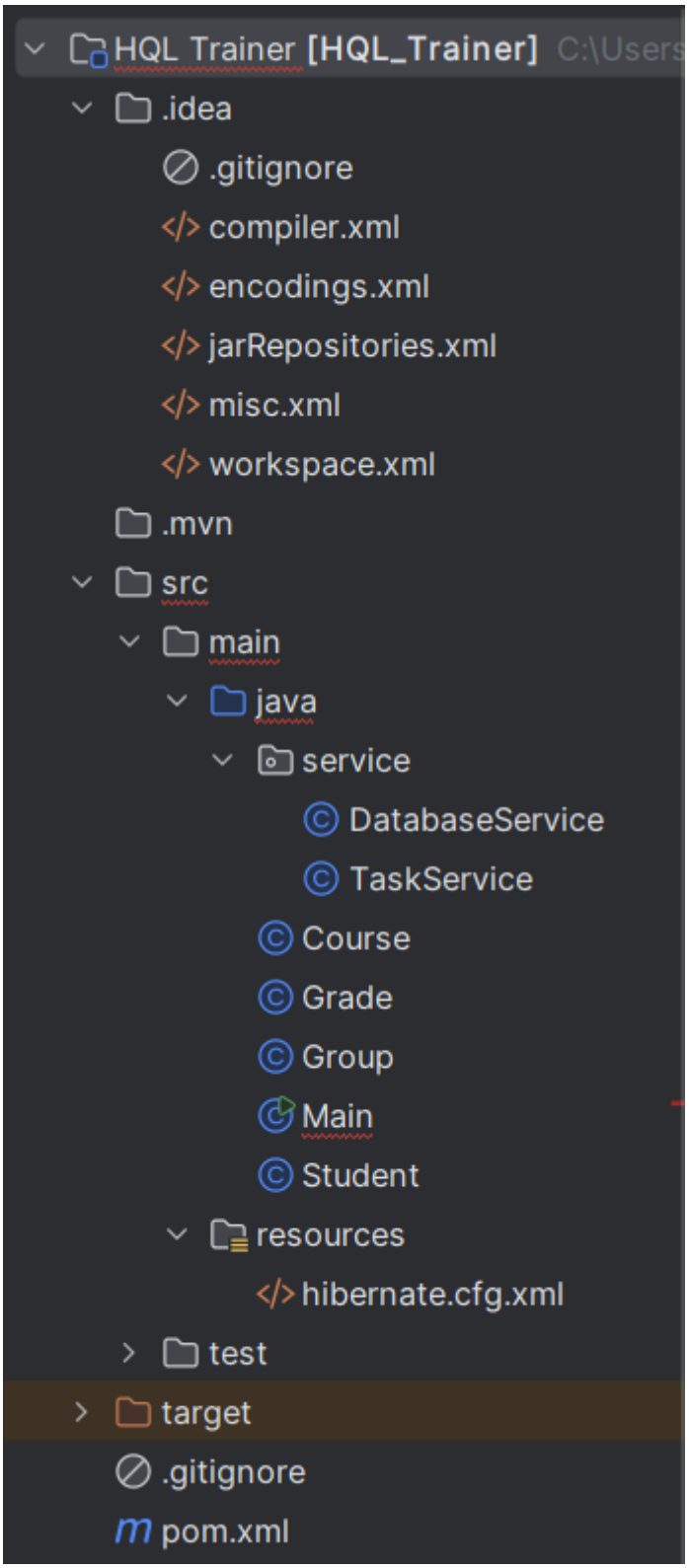

Рисунок 4.1 – Структура проекту в середовищі IntelliJ IDEA

Управління бібліотеками та залежностями реалізовано через файл pom.xml. Для забезпечення функціонування додатку було підключено наступні критично важливі залежності:

1) javafx-controls та javafx-fxml – для побудови графічного інтерфейсу користувача;
2) hibernate-core (версія 6.4.4) – для реалізації ORM-логіки та обробки HQL-запитів;
3) h2 – драйвер вбудованої бази даних для зберігання навчальних даних у пам’яті;
4) jfoenix – для стилізації елементів управління.

Конфігурація підключення до бази даних винесена у файл hibernate.cfg.xml, що знаходиться в каталозі ресурсів. Це дозволяє гнучко змінювати параметри підключення без перекомпіляції коду. Як видно з лістингу налаштувань, використовується режим create-drop, що забезпечує автоматичне створення чистої схеми бази даних при кожному запуску додатку:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    >     <property name="connection.driver_class">org.h2.Driver</property>
    <property name="connection.url">jdbc:h2:mem:testdb;DB_CLOSE_DELAY=-
1</property>
    <property name="connection.username">sa</property>
    <property name="connection.password"></property>
>     <property name="show_sql">true</property>
>     <property name="hbm2ddl.auto">create-drop</property>
>     <mapping class="org.example.Student"/>
<mapping class="org.example.Group"/>
<mapping class="org.example.Course"/>
<mapping class="org.example.Grade"/>
```

4.2. Опис програми

Програмна реалізація базується на архітектурному патерні, що розділяє логіку представлення (User Interface) та бізнес-логіку (Data Access Layer).

4.2.1. Реалізація моделі даних

Відповідно до спроектованої діаграми класів, було створено чотири сутності (Entity), які відображають предметну область «Навчальний процес». Ключовим класом є Student, який містить інформацію про здобувачів освіти. Програмна реалізація цього класу з використанням анотацій JPA представлена кодом нижче:

```
@Entity
@Table(name = "students")
public class Student {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

@Column(name = "first_name")
  private String firstName;

@Column(name = "last_name")
  private String lastName;

@ManyToOne(fetch = FetchType.EAGER)
@JoinColumn(name = "group_id")
  private Group group;

  private Double rating;

// Конструктори, геттери та сеттери
  public Student() {}

  public Student(String fName, String lName, Group group, Double rating) {
  this.firstName = fName;
  this.lastName = lName;
  this.group = group;
  this.rating = rating;
  }
  }
```

Використання анотації @ManyToOne для поля group дозволяє реалізувати зв’язок «багато-до-одного», що дає можливість студентам практикувати складні запити з об’єднанням даних (JOIN) через об’єктну навігацію. Аналогічним чином реалізовані класи Course, Group та Grade, що дозволяє формувати запити для аналізу успішності.

4.2.2. Логіка ініціалізації та виконання запитів

При запуску програми метод initDatabase() виконує початкове наповнення бази даних тестовими даними. Це критично важливий етап, оскільки H2 Database працює в оперативній пам’яті і очищується після закриття програми. Код ініціалізації створює об’єкти груп, курсів та студентів, після чого зберігає їх у персистентному контексті Hibernate. Логіка початкового наповнення бази даних тестовими записами інкапсульована у класі DatabaseService, фрагмент якого наведено у коді:

```
public void initData() {
    try (Session session = sessionFactory.openSession()) {
        session.beginTransaction();

    // 1. Створення груп
    Group g1 = new Group("CS-21");
    Group g2 = new Group("F1-24");
    session.persist(g1);
    session.persist(g2);
```

```
        // 2. Створення студентів
        session.persist(new Student("Ivan", "Petrenko", g1, 95.0));
        session.persist(new Student("Maria", "Koval", g1, 92.5));
        session.persist(new Student("Max", "Verstappen", g2, 99.9));

        // 3. Додавання оцінок
        Course c1 = new Course("Java Core");
        session.persist(c1);
        session.persist(new Grade(s1, c1, 95));

        session.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();
    }

    }
```

Обробка запитів користувача здійснюється в методі runQuery(). Цей метод зчитує текст з поля вводу, створює об’єкт Query та намагається виконати його. У разі успіху результати відображаються у таблиці TableView. Особливістю реалізації є динамічне формування колонок таблиці в залежності від типу повертаних даних, що дозволяє відображати як цілі об’єкти, так і окремі поля (проекції). Механізм перевірки правильності виконання завдань реалізовано у класі TaskService з використанням функціональних інтерфейсів наведено нижче у коді:

```
public class TaskService {
    private List<Task> tasks = new ArrayList<>();

private void initTasks() {
    // Завдання на фільтрацію
    tasks.add(new Task(
        "Виберіть студентів з рейтингом вище 90.",
        "SELECT s FROM Student s WHERE s.rating > 90",
        // Перевірка 1: Наявність ключового слова WHERE
        hql -> hql.toLowerCase().contains("where"),
        // Перевірка 2: Результат має містити 3 записи
        results -> results.size() == 3
    ));
}

}
```

4.3. Перевірка валідності та дослідження можливостей

Етап тестування мав на меті підтвердити стабільність роботи програмного засобу та коректність обробки HQL-запитів. Перевірка проводилася шляхом моделювання дій користувача.

Тест 1. Обробка коректних запитів**.** Було перевірено виконання запитів на вибірку, фільтрацію та сортування. Система коректно інтерпретує синтаксис HQL та повертає очікувані результати. Наприклад, при запиті на вибірку

студентів з рейтингом вище 90, програма відображає лише релевантні записи (рис. 4.2).

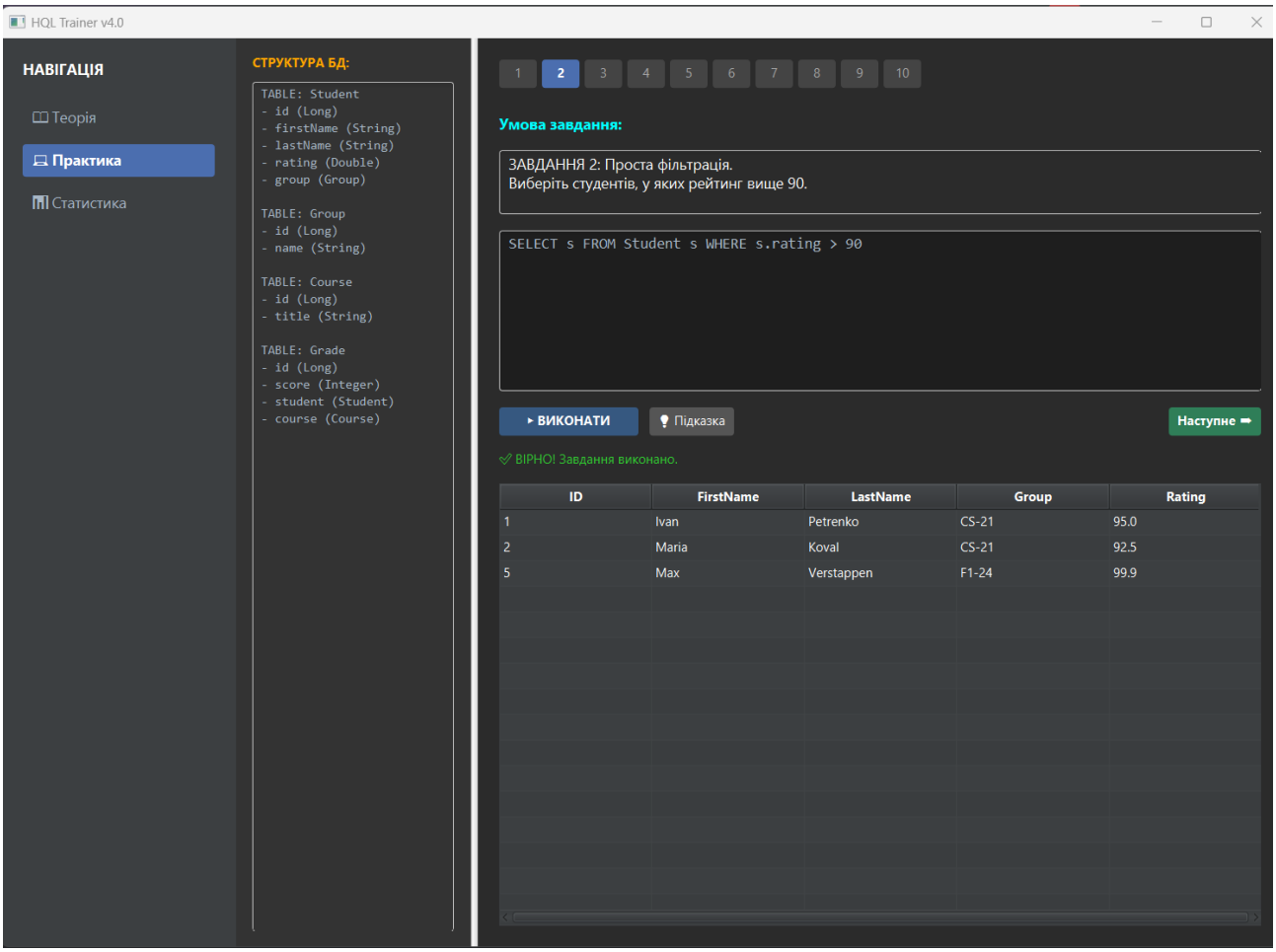

Рисунок 4.2 – Вивід інформації згідно HQL запиту

Тест 2. Обробка помилок. Важливим аспектом навчального ПЗ є реакція на помилки. При введенні некоректного запиту (наприклад, з друкарською помилкою в ключовому слові SELEKT), механізм перехоплення виключень (try-catch) блокує падіння програми і виводить повідомлення про помилку червоним кольором у статусний рядок (рис. 4.3).

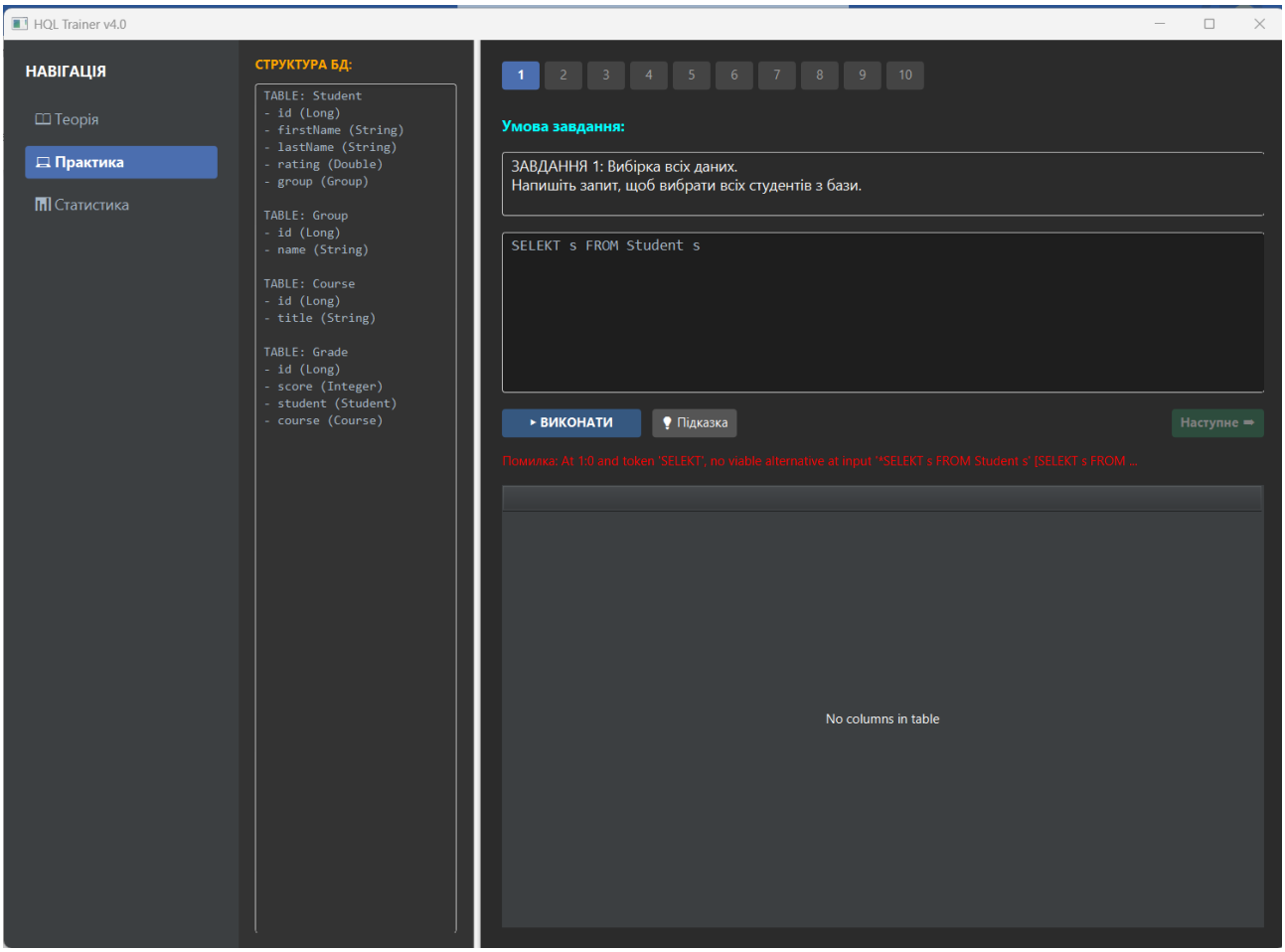

Рисунок 4.3 – Реакція системи на синтаксичну помилку в запиті

4.4. Інструкція користувача

Для забезпечення ефективного використання додатку розроблено детальну інструкцію користувача. Інтерфейс програми інтуїтивно зрозумілий та розділений на логічні блоки.

4.4.1. Робота з теоретичним матеріалом

Після запуску програми користувач може перейти до розділу «Теорія» за допомогою навігаційного меню зліва. Цей розділ містить структуровані лекційні матеріали. Користувач обирає тему зі списку (наприклад, «Вступ до HQL»), після чого в основній області з’являється текст лекції (рис. 4.4).

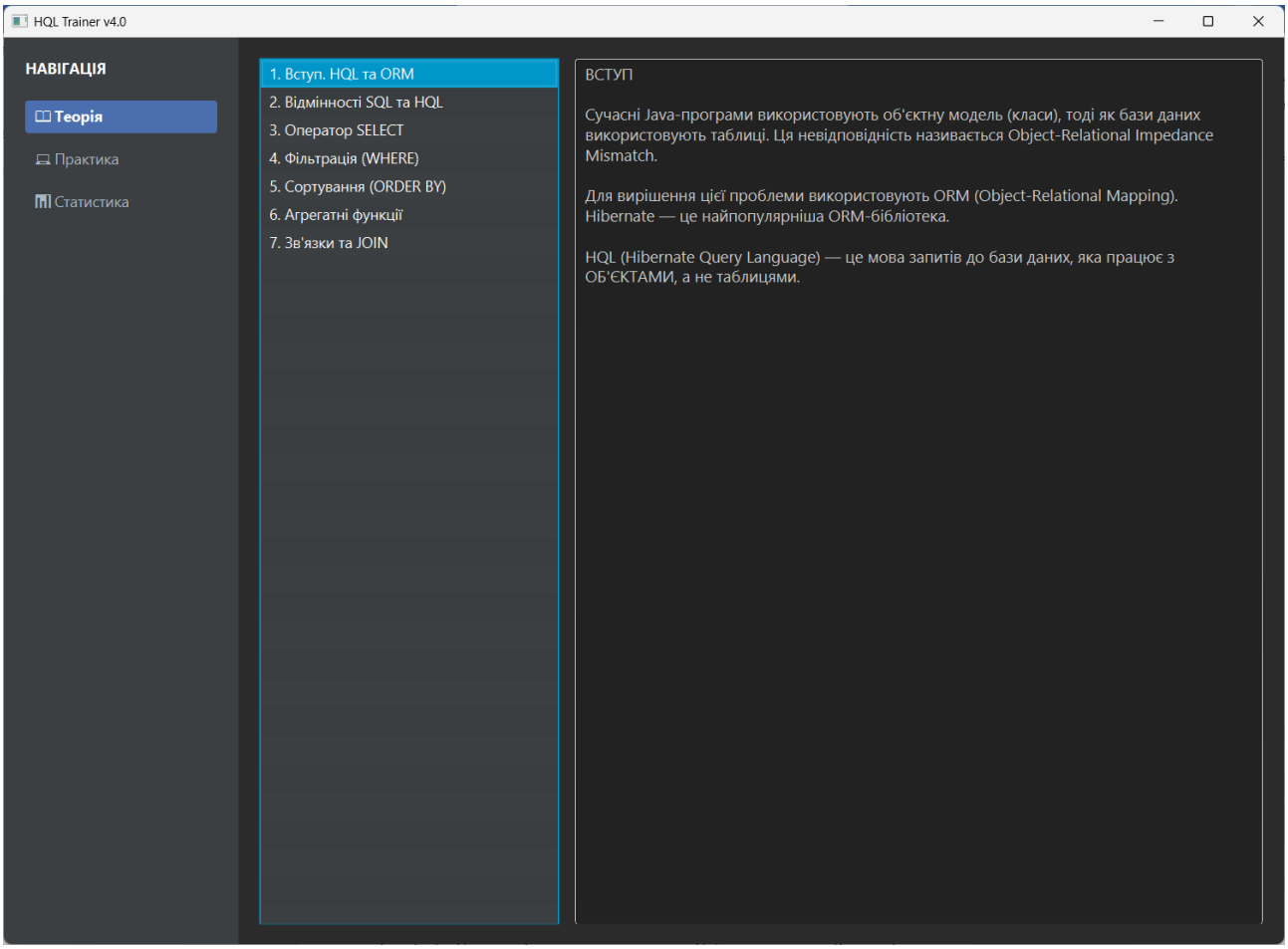

Рисунок 4.4 – Перегляд теоретичного матеріалу (Вступ)

Для поглибленого вивчення доступні інші теми, наприклад, про фільтрацію даних. Перемикання між темами відбувається миттєво (рис. 4.5).

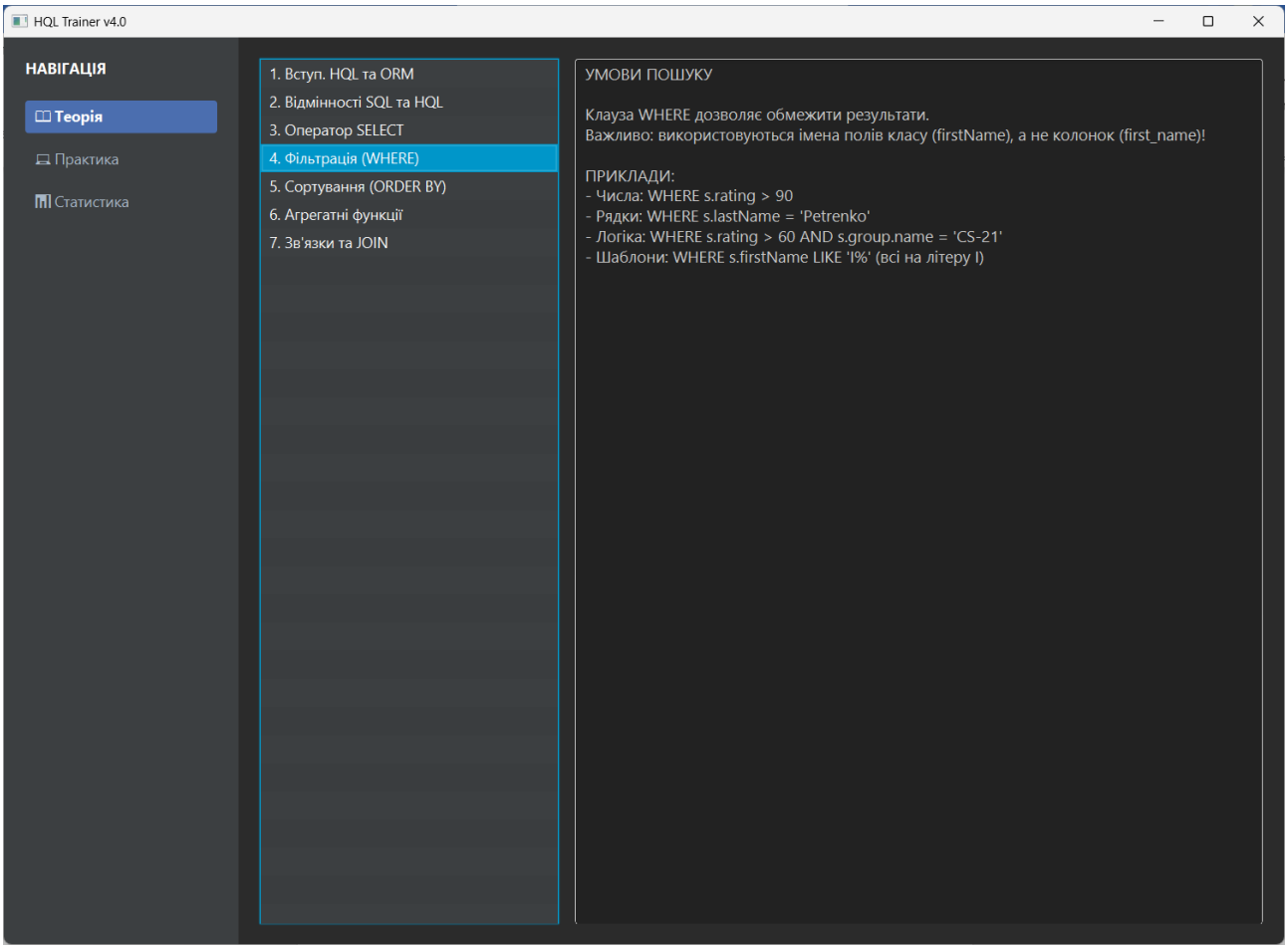

Рисунок 4.5 – Вивчення теми «Фільтрація даних»

4.4.2. Виконання практичних завдань

Розділ «Практика» є основним робочим середовищем. Зверху відображається умова поточного завдання. Для полегшення процесу написання коду та зниження когнітивного навантаження на студента, інтерфейс практичного модуля оснащено інформаційною панеллю «Структура БД», яка розміщена у лівій частині вікна. Цей блок у режимі реального часу відображає актуальну схему класів, з якими працює навчальна програма. Окрім базової сутності Student, система оперує пов’язаними класами Group (академічна група), Course (навчальна дисципліна) та Grade (оцінка). На панелі наведено перелік полів кожного класу та їх типи даних, що дозволяє користувачеві швидко орієнтуватися в моделі даних та коректно конструювати складні запити (наприклад, фільтрацію оцінок за назвою курсу) без необхідності запам'ятовувати назви всіх атрибутів.

Користувач повинен ввести код запиту у текстове поле і натиснути кнопку «ВИКОНАТИ». Навчальний курс, реалізований у програмі, складається з 10 практичних завдань різного рівня складності: від простих вибірок даних до використання агрегатних функцій та навігації по зв'язках об'єктів. Для забезпечення гнучкості навчального процесу у верхній частині робочої області реалізовано панель пагінації (шкала з кнопками 1–10). Цей елемент навігації дозволяє студенту миттєво переходити до будь-якого завдання, довільно змінювати порядок їх виконання або повертатися до попередніх етапів для повторення матеріалу, що робить процес навчання більш адаптивним та комфортним (рис. 4.6).

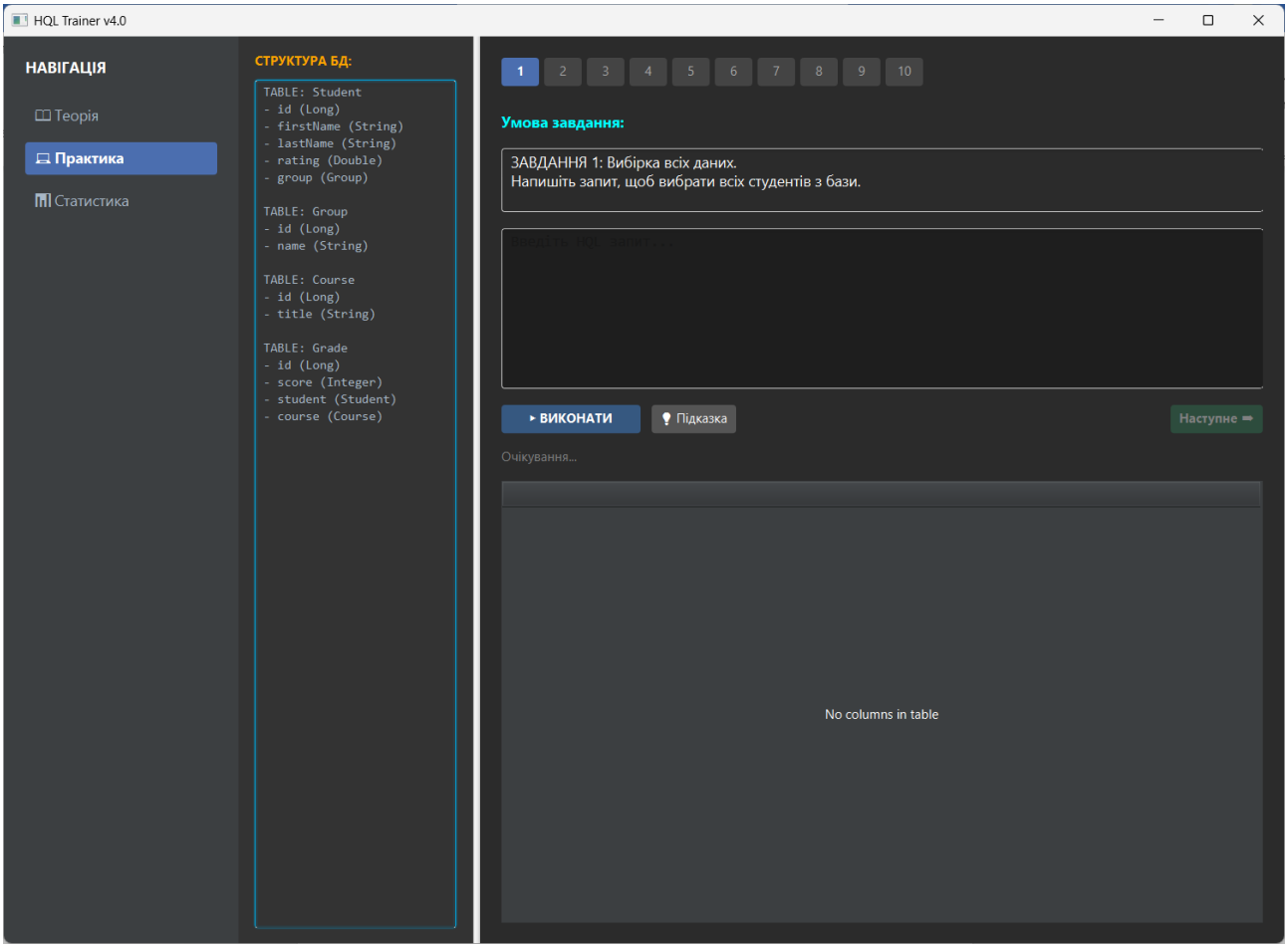

Рисунок 4.6 – Вікно практичного завдання (початковий стан)

У разі успішного виконання запиту, результати відображаються у таблиці знизу, а статус виконання змінюється на «Успішно». Кнопка «Наступне завдання» стає активною (рис. 4.7).

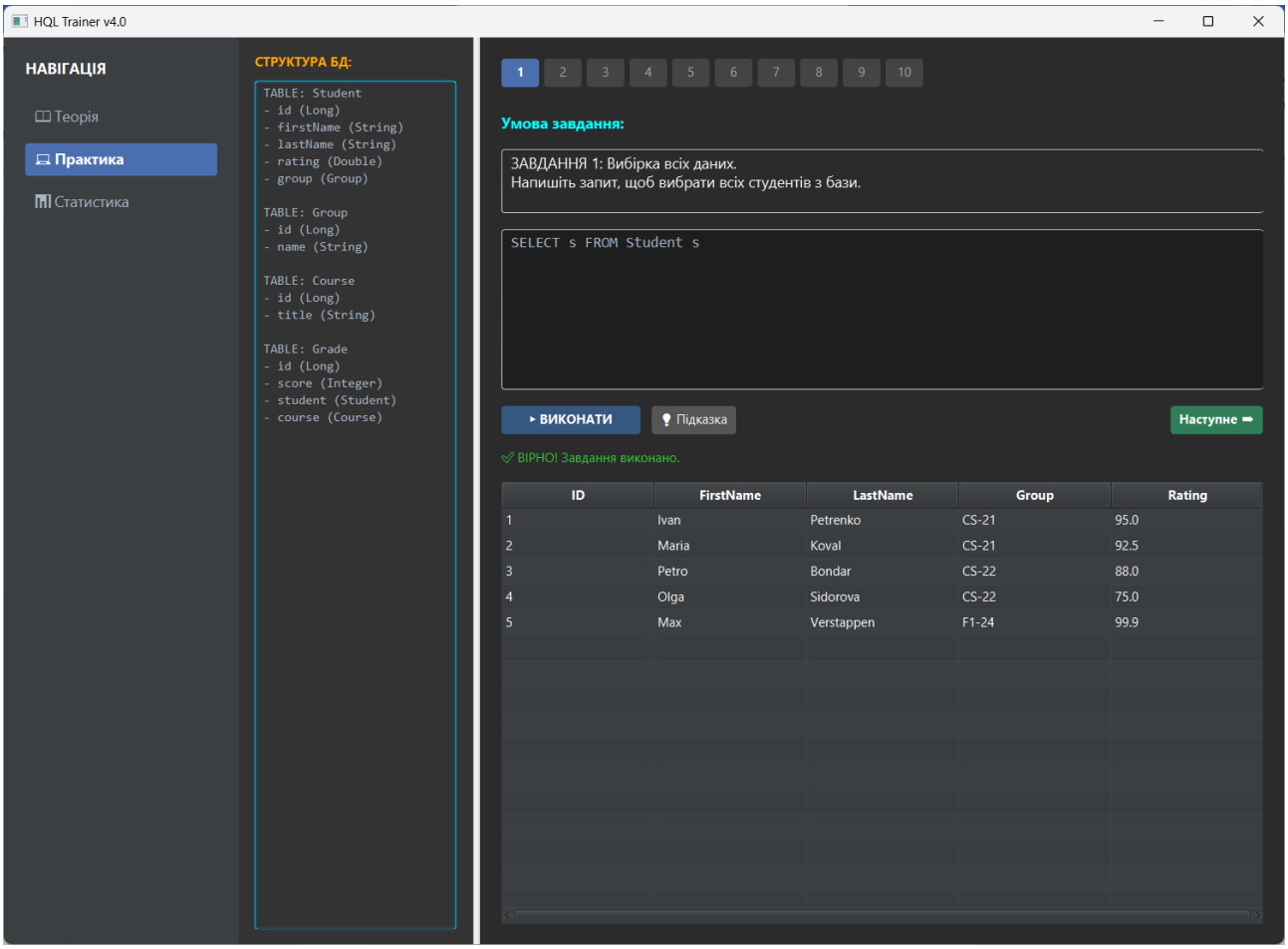

Рисунок 4.7 – Успішне виконання завдання №1

Якщо користувач стикається з труднощами, він може скористатися системою підказок. Натискання кнопки «Показати відповідь» автоматично заповнює редактор правильним кодом, що дозволяє проаналізувати логіку рішення (рис. 4.8).

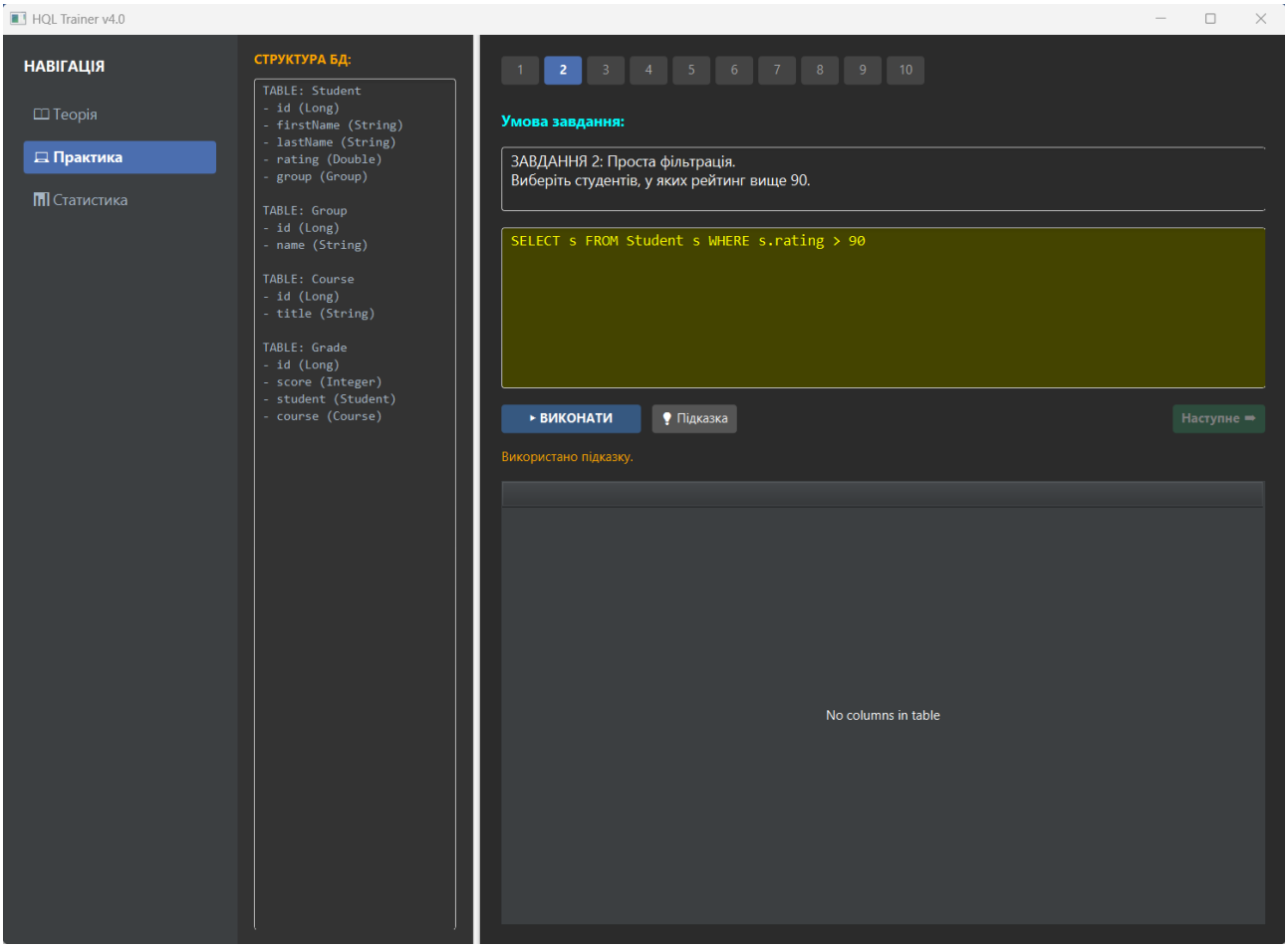

Рисунок 4.8 – Використання функції підказки у Завданні №2

4.4.3. Перегляд статистики

Для контролю власної успішності реалізовано модуль «Статистика». Він візуалізує співвідношення успішних та помилкових спроб виконання запитів за допомогою кругової діаграми. Зелений сектор відповідає успішним запитам, червоний – помилковим (рис. 4.9).

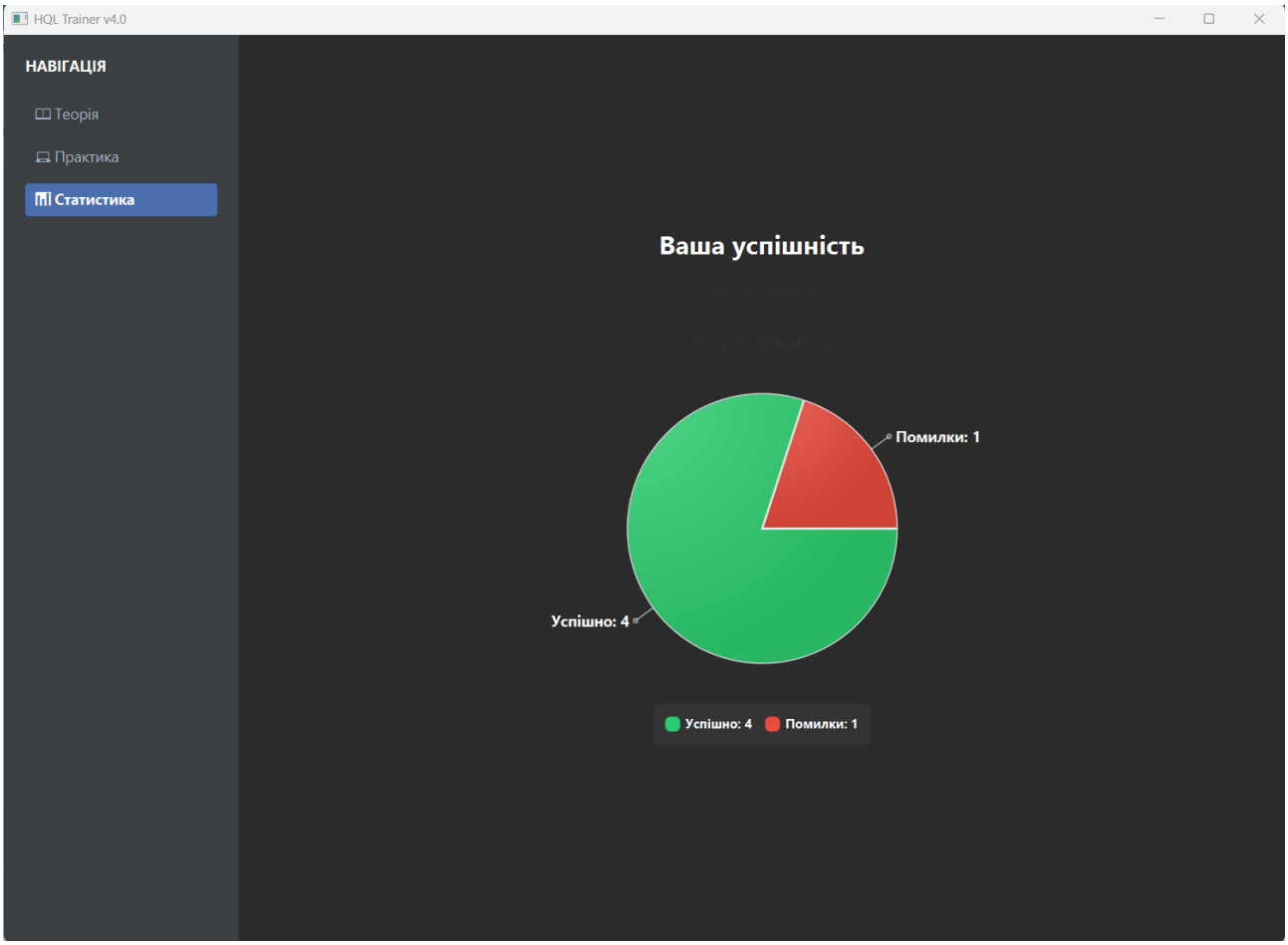

Рисунок 4.9 – Візуалізація статистики успішності користувача

Таким чином, розроблений програмний засіб забезпечує повний цикл навчання, від теорії до практики та контролю результатів, відповідаючи всім вимогам технічного завдання.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне науково-практичне завдання, яке полягає у підвищенні ефективності підготовки фахівців з комп’ютерних наук шляхом розробки спеціалізованого програмного забезпечення для вивчення мови запитів HQL (Hibernate Query Language).

У ході виконання роботи було отримано наступні основні результати:

1) Проведено аналіз предметної області та існуючих методів навчання. Встановлено, що однією з проблем при вивченні дисципліни «Технології обробки та аналізу даних» є складність переходу студентів до об’єктно-орієнтованої логіки запитів після роботи з реляційними базами даних. Аналіз існуючих аналогів підтвердив доцільність розробки власного програмного продукту, який поєднує теоретичний матеріал з практичним виконанням коду.
2) Обґрунтовано вибір технологічного стеку. Для реалізації системи обрано мову Java та бібліотеку Hibernate ORM, що забезпечує відповідність навчального середовища реальним умовам розробки. Використання H2 Database у режимі In-Memory дозволило забезпечити автономність програми.
3) Спроектовано архітектуру та структуру даних. Розроблено навчальну модель «Університет» (класи Student, Group, Course, Grade), яка дозволяє формулювати різнопланові завдання для відпрацювання навичок роботи зі зв'язками та агрегатними функціями.
4) Розроблено алгоритм роботи програми. Створено алгоритм перевірки практичних завдань, який базується на валідації результатів виконання HQL-запиту, що дозволяє оцінювати не лише синтаксис, а й логічну правильність отриманих даних.
5) Здійснено програмну реалізацію навчального засобу. Створено десктопний додаток із трьома модулями: «Теорія», «Практика» та «Статистика». Реалізовано редактор коду з підсвічуванням синтаксису, панель навігації по завданням та візуалізацію результатів у вигляді таблиці.

6) Проведено тестування. Перевірка підтвердила стабільність роботи додатку, коректність виконання запитів та відповідність інтерфейсу вимогам зручності використання.

Практичне значення роботи полягає у створенні інструменту, який може бути використаний у навчальному процесі кафедри при викладанні дисциплін, пов'язаних з базами даних та Java-розробкою.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bauer C., King G., Gregory G. Java Persistence with Hibernate. 5th ed. Manning Publications, 2015. 888 p.
2. Hibernate ORM 6.4 User Guide [Електронний ресурс]. URL: https://docs.jboss.org/hibernate/orm/6.4/userguide/html_single/Hibernate_User_Guide.html (дата звернення: 10.12.2025).
3. Horstmann C. S. Core Java, Volume I: Fundamentals. 12th ed. Oracle Press, 2021. 944 p.
4. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. 560 p.
5. Schildt H. Java: The Complete Reference. 12th ed. McGraw-Hill Education, 2021. 1280 p.
6. JavaFX 17 Documentation [Електронний ресурс]. URL: https://openjfx.io/javadoc/17/ (дата звернення: 11.12.2025).
7. H2 Database Engine [Електронний ресурс]. URL: https://www.h2database.com/html/main.html (дата звернення: 11.12.2025).
8. SQLZoo: Interactive SQL Tutorial [Електронний ресурс]. URL: https://sqlzoo.net/ (дата звернення: 05.12.2025).
9. LeetCode – The World's Leading Online Programming Learning Platform [Електронний ресурс]. URL: https://leetcode.com/ (дата звернення: 05.12.2025).
10. Bloch J. Effective Java. 3rd ed. Addison-Wesley Professional, 2018. 416 p.
11. Гончаров О. А. Бази даних: навчальний посібник. Київ: КПІ ім. Ігоря Сікорського, 2018. 156 с.
12. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. Київ: ДП «УкрНДНЦ», 2016. 26 с.
13. Лелюх В. О. Розробка системи тестування знань студентів з використанням мобільних технологій: магістерська робота. Полтава: ПУЕТ, 2023. URL:

http://dspace.puet.edu.ua/bitstream/123456789/14806/1/Магістерська_Лелюх.pdf (дата звернення: 12.12.2025).

14. Недаєва К. О. Автоматизована система управління навчальним процесом: магістерська робота. Полтава: ПУЕТ, 2021. URL: http://dspace.puet.edu.ua/bitstream/123456789/10341/1/Nedaieva_EK21m_.pdf (дата звернення: 12.12.2025).
15. Вісловух В. В. Система виявлення SQL Injection у мережевому трафіку: дипломна робота бакалавра. Київ: КПІ ім. Ігоря Сікорського, 2024. URL: https://ela.kpi.ua/bitstreams/35f66818-c144-4ca4-8754-762f09439be0/download (дата звернення: 12.12.2025).

ДОДАТОК А. КОД ПРОГРАМИ

Програмний код навчального засобу:

Main.java

```
package org.example;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.beans.property.SimpleStringProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.geometry.Side;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.chart.PieChart;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import org.example.service.DatabaseService;
import org.example.service.TaskService;

import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

public class Main extends Application {

    private DatabaseService dbService = new DatabaseService();
    private TaskService taskService = new TaskService();

    private TextArea codeArea;
    private TableView<Object> resultTable;
    private Label statusLabel;
    private TextArea taskDescriptionArea;
    private Button btnNextTask;
    private BorderPane root;

    // Змінні стану
    private List<TaskService.Task> tasks;
    private int currentTaskIndex = 0;
    private int totalAttempts = 0;
    private int successfulQueries = 0;
    private Map<String, String> theoryContent = new LinkedHashMap<>();

    // Кнопки меню
    private Button btnTheory;
    private Button btnPractice;
    private Button btnStats;
    private VBox sidebar;

    @Override
    public void start(Stage primaryStage) {
        dbService.init();
```

```
        tasks = taskService.getTasks();
        initTheoryData();

        root = new BorderPane();
        root.setStyle("-fx-background-color: #2B2B2B;");

        // --- ЛІВЕ МЕНЮ ---
        sidebar = new VBox(10);
        sidebar.setPadding(new Insets(20));
        sidebar.setStyle("-fx-background-color: #3C3F41; -fx-pref-width: 220;");

        Label lblMenu = new Label("НАВІГАЦІЯ");
        lblMenu.setTextFill(Color.WHITE);
        lblMenu.setFont(Font.font("Segoe UI", FontWeight.BOLD, 14));
        lblMenu.setPadding(new Insets(0, 0, 10, 0));

        btnTheory = createStyledButton("   Теорія");
        btnPractice = createStyledButton("   Практика");
        btnStats = createStyledButton("   Статистика");

        btnTheory.setOnAction(e -> {
            setActiveButton(btnTheory);
            showTheoryView();
        });

        btnPractice.setOnAction(e -> {
            setActiveButton(btnPractice);
            showPracticeView();
        });

        btnStats.setOnAction(e -> {
            setActiveButton(btnStats);
            showStatisticsView();
        });

        sidebar.getChildren().addAll(lblMenu, btnTheory, btnPractice, btnStats);
        root.setLeft(sidebar);

        // Старт з практики
        setActiveButton(btnPractice);
        showPracticeView();

        Scene scene = new Scene(root, 1200, 850);
        primaryStage.setTitle("HQL Trainer v4.0");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    private void setActiveButton(Button activeBtn) {
        resetButtonStyle(btnTheory);
        resetButtonStyle(btnPractice);
        resetButtonStyle(btnStats);
        activeBtn.setStyle("-fx-background-color: #4B6EAF; -fx-text-fill: white;
-fx-alignment: CENTER_LEFT; -fx-font-size: 14px; -fx-cursor: hand; -fx-font-
weight: bold;");
    }

    private void resetButtonStyle(Button btn) {
        btn.setStyle("-fx-background-color: transparent; -fx-text-fill: #A9B7C6;
-fx-alignment: CENTER_LEFT; -fx-font-size: 14px; -fx-cursor: hand;");
    }

    // --- ВІКНО ПРАКТИКИ ---
    private void showPracticeView() {
        SplitPane splitPane = new SplitPane();
```

```
        splitPane.setStyle("-fx-background-color: #2B2B2B; -fx-padding: 0;");

        // ЛІВА ПАНЕЛЬ
        VBox dbInfoBox = new VBox(10);
        dbInfoBox.setPadding(new Insets(15));
        dbInfoBox.setStyle("-fx-background-color: #333333; -fx-border-color:
#444; -fx-border-width: 0 1 0 0;");
        dbInfoBox.setMinWidth(220);
        dbInfoBox.setMaxWidth(280);

        Label lblDbStruct = new Label("СТРУКТУРА БД:");
        lblDbStruct.setTextFill(Color.ORANGE);
        lblDbStruct.setFont(Font.font("Segoe UI", FontWeight.BOLD, 12));

        TextArea dbStructArea = new TextArea(getDbStructureSummary());
        dbStructArea.setEditable(false);
        dbStructArea.setStyle("-fx-control-inner-background: #333333; -fx-text-
fill: #A9B7C6; -fx-font-family: 'Consolas'; -fx-font-size: 12px;");
        VBox.setVgrow(dbStructArea, Priority.ALWAYS);

        dbInfoBox.getChildren().addAll(lblDbStruct, dbStructArea);

        // ПРАВА ПАНЕЛЬ
        VBox practiceBox = new VBox(15);
        practiceBox.setPadding(new Insets(20));
        practiceBox.setStyle("-fx-background-color: #2B2B2B;");

        // Шкала навігації
        HBox paginationBar = createPaginationBar();

        Label lblHeader = new Label("Умова завдання:");
        lblHeader.setTextFill(Color.CYAN);
        lblHeader.setFont(Font.font("Segoe UI", FontWeight.BOLD, 14));

        taskDescriptionArea = new TextArea();
        taskDescriptionArea.setEditable(false);
        taskDescriptionArea.setWrapText(true);
        taskDescriptionArea.setPrefHeight(60);
        taskDescriptionArea.setStyle("-fx-control-inner-background: #2B2B2B; -
fx-text-fill: white; -fx-font-size: 14px; -fx-font-family: 'Segoe UI';");

        codeArea = new TextArea();
        codeArea.setPromptText("Введіть HQL запит...");
        codeArea.setStyle("-fx-control-inner-background: #1E1E1E; -fx-text-fill:
#A9B7C6; -fx-font-family: 'Consolas'; -fx-font-size: 14px;");
        codeArea.setPrefHeight(150);
        codeArea.setOnKeyTyped(e -> codeArea.setStyle("-fx-control-inner-
background: #1E1E1E; -fx-text-fill: #A9B7C6; -fx-font-family: 'Consolas'; -fx-
font-size: 14px;"));

        HBox buttonBox = new HBox(10);
        Button btnRun = new Button("▶ ВИКОНАТИ");
        btnRun.setStyle("-fx-background-color: #365880; -fx-text-fill: white; -
fx-font-weight: bold; -fx-cursor: hand;");
        btnRun.setPrefWidth(130);
        btnRun.setOnAction(e -> runQuery());

        Button btnHint = new Button("   Підказка");
        btnHint.setStyle("-fx-background-color: #555555; -fx-text-fill: white; -
fx-cursor: hand;");
        btnHint.setOnAction(e -> {
            codeArea.setText(tasks.get(currentTaskIndex).getCorrectHql());
            codeArea.setStyle("-fx-control-inner-background: #444400; -fx-text-
fill: #FFFF00; -fx-font-family: 'Consolas'; -fx-font-size: 14px;");
            statusLabel.setText("Використано підказку.");
```

```
            statusLabel.setTextFill(Color.ORANGE);
        });

        btnNextTask = new Button("Наступне ➡");
        btnNextTask.setStyle("-fx-background-color: #2F7E58; -fx-text-fill:
white; -fx-font-weight: bold; -fx-cursor: hand;");
        btnNextTask.setOnAction(e -> loadNextTask());
        btnNextTask.setDisable(true);

        buttonBox.getChildren().addAll(btnRun, btnHint, new Region(),
btnNextTask);
        HBox.setHgrow(buttonBox.getChildren().get(2), Priority.ALWAYS);

        statusLabel = new Label("Очікування...");
        statusLabel.setTextFill(Color.GRAY);
        statusLabel.setFont(Font.font("Segoe UI", 12));

        resultTable = new TableView<>();
        resultTable.setStyle("-fx-background-color: #3C3F41; -fx-base: #3C3F41;
-fx-control-inner-background: #3C3F41;");
        resultTable.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
        VBox.setVgrow(resultTable, Priority.ALWAYS);

        practiceBox.getChildren().addAll(paginationBar, lblHeader,
taskDescriptionArea, codeArea, buttonBox, statusLabel, resultTable);
        splitPane.getItems().addAll(dbInfoBox, practiceBox);
        splitPane.setDividerPositions(0.22);
        splitPane.lookupAll(".split-pane-divider").forEach(div ->
div.setStyle("-fx-background-color: #2B2B2B;"));

        root.setCenter(splitPane);
        loadCurrentTaskData();
    }

    private HBox createPaginationBar() {
        HBox bar = new HBox(5);
        bar.setPadding(new Insets(0, 0, 10, 0));
        for (int i = 0; i < tasks.size(); i++) {
            final int index = i;
            Button btn = new Button(String.valueOf(i + 1));
            btn.setPrefWidth(35);
            if (i == currentTaskIndex) {
                btn.setStyle("-fx-background-color: #4B6EAF; -fx-text-fill:
white; -fx-font-weight: bold; -fx-cursor: hand;");
            } else {
                btn.setStyle("-fx-background-color: #444444; -fx-text-fill:
#AAAAAA; -fx-cursor: hand;");
            }
            btn.setOnAction(e -> {
                currentTaskIndex = index;
                showPracticeView();
            });
            bar.getChildren().add(btn);
        }
        return bar;
    }

    private void runQuery() {
        totalAttempts++;
        String hql = codeArea.getText();
        statusLabel.setText("Виконую...");
        statusLabel.setTextFill(Color.YELLOW);

        try {
            List<?> results = dbService.executeQuery(hql);
            updateTableStructure(results);
```

```
            statusLabel.setText("Знайдено записів: " + results.size());
            statusLabel.setTextFill(Color.LIGHTBLUE);

            TaskService.Task currentTask = tasks.get(currentTaskIndex);
            if (currentTask.validate(hql, results)) {
                statusLabel.setText("□ ВІРНО! Завдання виконано.");
                statusLabel.setTextFill(Color.LIMEGREEN);
                successfulQueries++;
                btnNextTask.setDisable(false);
            } else {
                statusLabel.setText("□□ Запит виконано, але результат не
відповідає умові.");
                statusLabel.setTextFill(Color.ORANGE);
            }
        } catch (Exception e) {
            String msg = e.getCause() != null ? e.getCause().getMessage() :
e.getMessage();
            if (msg != null && msg.length() > 100) msg = msg.substring(0, 100) +
"...";
            statusLabel.setText("Помилка: " + msg);
            statusLabel.setTextFill(Color.RED);
        }
    }

    private void updateTableStructure(List<?> results) {
        resultTable.getColumns().clear();
        resultTable.getItems().clear();
        if (results.isEmpty()) return;

        Object firstRow = results.get(0);
        if (firstRow instanceof Student) {
            createColumn("ID", "id");
            createColumn("FirstName", "firstName");
            createColumn("LastName", "lastName");
            TableColumn<Object, String> groupCol = new TableColumn<>("Group");
            groupCol.setCellValueFactory(d -> new
SimpleStringProperty(((Student)d.getValue()).getGroup().getName()));
            resultTable.getColumns().add(groupCol);
            createColumn("Rating", "rating");
        } else if (firstRow instanceof Group) {
            createColumn("ID", "id");
            createColumn("GroupName", "name");
        } else if (firstRow instanceof Grade) {
            createColumn("ID", "id");
            createColumn("Score", "score");
            TableColumn<Object, String> studCol = new TableColumn<>("Student");
            studCol.setCellValueFactory(d -> new
SimpleStringProperty(((Grade)d.getValue()).getStudent().getLastName()));
            resultTable.getColumns().add(studCol);
            TableColumn<Object, String> courseCol = new TableColumn<>("Course");
            courseCol.setCellValueFactory(d -> new
SimpleStringProperty(((Grade)d.getValue()).getCourse().getTitle()));
            resultTable.getColumns().add(courseCol);
        } else if (firstRow instanceof Object[]) {
            Object[] row = (Object[]) firstRow;
            for (int i = 0; i < row.length; i++) {
                final int index = i;
                TableColumn<Object, String> col = new TableColumn<>("Col " + (i
+ 1));
                col.setCellValueFactory(data -> {
                    Object[] rowData = (Object[]) data.getValue();
                    return new SimpleStringProperty(rowData[index] != null ?
rowData[index].toString() : "NULL");
                });
                resultTable.getColumns().add(col);
            }
```

```
        } else {
            TableColumn<Object, String> col = new TableColumn<>("Result");
            col.setCellValueFactory(data -> new
SimpleStringProperty(data.getValue().toString()));
            resultTable.getColumns().add(col);
        }
        resultTable.getItems().addAll(results);
    }

    private void createColumn(String header, String property) {
        TableColumn<Object, Object> col = new TableColumn<>(header);
        col.setCellValueFactory(new PropertyValueFactory<>(property));
        resultTable.getColumns().add(col);
    }

    // --- ВІКНО ТЕОРІЇ ---
    private void showTheoryView() {
        HBox theoryLayout = new HBox(15);
        theoryLayout.setPadding(new Insets(20));

        ListView<String> topicList = new ListView<>();
        topicList.getItems().addAll(theoryContent.keySet());
        topicList.setPrefWidth(280);
        topicList.setStyle("-fx-control-inner-background: #3C3F41; -fx-text-
fill: white; -fx-font-size: 14px;");

        TextArea contentArea = new TextArea("Оберіть тему для початку
навчання...");
        contentArea.setWrapText(true);
        contentArea.setEditable(false);
        contentArea.setStyle("-fx-control-inner-background: #222222; -fx-text-
fill: #CCCCCC; -fx-font-size: 15px; -fx-font-family: 'Segoe UI';");
        HBox.setHgrow(contentArea, Priority.ALWAYS);

        topicList.getSelectionModel().selectedItemProperty().addListener((obs,
oldVal, newVal) -> {
            if (newVal != null) contentArea.setText(theoryContent.get(newVal));
        });

        theoryLayout.getChildren().addAll(topicList, contentArea);
        root.setCenter(theoryLayout);
    }

    private void initTheoryData() {
        theoryContent.put("1. Вступ. HQL та ORM",
                "ВСТУП\n\n" +
                        "Сучасні Java-програми використовують об'єктну модель
(класи), тоді як бази даних використовують таблиці. Ця невідповідність
називається Object-Relational Impedance Mismatch.\n\n" +
                        "Для вирішення цієї проблеми використовують ORM (Object-
Relational Mapping).\n" +
                        "Hibernate — це найпопулярніша ORM-бібліотека.\n\n" +
                        "HQL (Hibernate Query Language) — це мова запитів до
бази даних, яка працює з ОБ'ЄКТАМИ, а не таблицями.");

        theoryContent.put("2. Відмінності SQL та HQL",
                "ПОРІВНЯННЯ:\n\n" +
                        "1. SQL працює з таблицями, HQL — з класами.\n" +
                        "   SQL: SELECT * FROM t_student\n" +
                        "   HQL: FROM Student\n\n" +
                        "2. HQL чутливий до регістру імен класів.\n" +
                        "   'from Student' — вірно.\n" +
                        "   'from student' — помилка (якщо класу student
немає).\n\n" +
                        "3. HQL підтримує поліморфізм (вибірка батьківського
класу автоматично підтягує спадкоємців).");
```

```
        theoryContent.put("3. Оператор SELECT",
                "ВИБІРКА ДАНИХ\n\n" +
                        "Оператор SELECT вказує, які дані ми хочемо
отримати.\n\n" +
                        "1. Повна вибірка об'єктів:\n" +
                        "   FROM Student\n" +
                        "   (Це аналог SELECT * FROM students)\n\n" +
                        "2. Використання аліасу (псевдоніму):\n" +
                        "   SELECT s FROM Student s\n" +
                        "   (s - це посилання на поточний об'єкт студента)\n\n"
+
                        "3. Вибірка окремих полів:\n" +
                        "   SELECT s.firstName, s.lastName FROM Student s");

        theoryContent.put("4. Фільтрація (WHERE)",
                "УМОВИ ПОШУКУ\n\n" +
                        "Клауза WHERE дозволяє обмежити результати.\n" +
                        "Важливо: використовуються імена полів класу
(firstName), а не колонок (first_name)!\n\n" +
                        "ПРИКЛАДИ:\n" +
                        "- Числа: WHERE s.rating > 90\n" +
                        "- Рядки: WHERE s.lastName = 'Petrenko'\n" +
                        "- Логіка: WHERE s.rating > 60 AND s.group.name = 'CS-
21'\n" +
                        "- Шаблони: WHERE s.firstName LIKE 'I%' (всі на літеру
I)");

        theoryContent.put("5. Сортування (ORDER BY)",
                "ВПОРЯДКУВАННЯ\n\n" +
                        "Синтаксис: ORDER BY поле [ASC|DESC]\n" +
                        "ASC - зростання (за замовчуванням)\n" +
                        "DESC - спадання\n\n" +
                        "ПРИКЛАД:\n" +
                        "   FROM Student s ORDER BY s.rating DESC\n" +
                        "   (від найвищого балу до найнижчого)");

        theoryContent.put("6. Агрегатні функції",
                "ОБЧИСЛЕННЯ\n\n" +
                        "HQL підтримує функції, що повертають одне значення для
групи рядків:\n" +
                        "- count(*): кількість записів\n" +
                        "- sum(x): сума\n" +
                        "- avg(x): середнє арифметичне\n" +
                        "- min(x), max(x): мінімум/максимум\n\n" +
                        "ПРИКЛАД:\n" +
                        "   SELECT avg(s.rating) FROM Student s");

        theoryContent.put("7. Зв'язки та JOIN",
                "НАВІГАЦІЯ ПО ЗВ'ЯЗКАХ\n\n" +
                        "Найбільша сила HQL — це робота зі зв'язками.\n\n" +
                        "1. Неявний JOIN (Implicit Join):\n" +
                        "   SELECT s FROM Student s WHERE s.group.name = 'CS-
21'\n" +
                        "   (Ми йдемо через крапку: студент -> група ->
назва)\n\n" +
                        "2. Явний JOIN:\n" +
                        "   SELECT s FROM Student s JOIN s.group g WHERE g.name
= 'CS-21'\n" +
                        "   (Використовується, коли треба контролювати тип
з'єднання: INNER, LEFT, RIGHT).");
    }

    // --- ВІКНО СТАТИСТИКИ (ВИПРАВЛЕНА ПРОБЛЕМА "БІЛЕ НА БІЛОМУ") ---
    private void showStatisticsView() {
        VBox statsBox = new VBox(20);
```

```
        statsBox.setPadding(new Insets(30));
        statsBox.setAlignment(Pos.CENTER);

        Label header = new Label("Ваша успішність");
        header.setTextFill(Color.WHITE);
        header.setFont(Font.font("Segoe UI", FontWeight.BOLD, 24));

        int failures = totalAttempts - successfulQueries;

        ObservableList<PieChart.Data> data = FXCollections.observableArrayList(
                new PieChart.Data("Успішно: " + successfulQueries,
successfulQueries),
                new PieChart.Data("Помилки: " + failures, failures));

        PieChart chart = new PieChart(data);
        chart.setTitle("Результативність");
        chart.setLabelsVisible(true);
        chart.setLegendVisible(true);
        chart.setLegendSide(Side.BOTTOM);

        statsBox.getChildren().addAll(header, new Label("Всього запитів: " +
totalAttempts), chart);
        root.setCenter(statsBox);

        // ФАРБУВАННЯ ДІАГРАМИ ТА ЛЕГЕНДИ ---
        Platform.runLater(() -> {
            for (PieChart.Data d : data) {
                // Фарбуємо сектори: Зелений та Червоний
                String color = d.getName().contains("Успішно") ? "#2ecc71" :
"#e74c3c";
                d.getNode().setStyle("-fx-pie-color: " + color + ";");
            }

            //  фон легенди
            Node legend = chart.lookup(".chart-legend");
            if (legend != null) {
                legend.setStyle("-fx-background-color: #333333; -fx-padding:
10px; -fx-background-radius: 5px;");
            }

            // 2.  текст на самій діаграмі
            for (Node node : chart.lookupAll(".chart-pie-label")) {
                if (node instanceof Text) {
                    ((Text) node).setFill(Color.WHITE);
                    ((Text) node).setStyle("-fx-font-size: 14px; -fx-font-
weight: bold;");
                }
            }

            // 3. текст у легенді знизу
            for (Node node : chart.lookupAll(".chart-legend-item")) {
                if (node instanceof Label) {
                    Label label = (Label) node;
                    label.setTextFill(Color.WHITE); // Робимо текст білим
                    label.setStyle("-fx-text-fill: white; -fx-font-size: 12px; -
fx-font-weight: bold;");


                    if (label.getText().contains("Успішно")) {
                        label.getGraphic().setStyle("-fx-background-color:
#2ecc71; -fx-background-radius: 5px;");
                    } else {
                        label.getGraphic().setStyle("-fx-background-color:
#e74c3c; -fx-background-radius: 5px;");
                    }
                }
```

```
            }
        });
    }

    private Button createStyledButton(String text) {
        Button btn = new Button(text);
        btn.setStyle("-fx-background-color: transparent; -fx-text-fill: #A9B7C6;
-fx-alignment: CENTER_LEFT; -fx-font-size: 14px; -fx-cursor: hand;");
        btn.setPrefWidth(180);

        btn.setOnMouseEntered(e -> {
            if (!btn.getStyle().contains("#4B6EAF")) {
                btn.setStyle("-fx-background-color: #333333; -fx-text-fill:
white; -fx-alignment: CENTER_LEFT; -fx-font-size: 14px; -fx-cursor: hand;");
            }
        });

        btn.setOnMouseExited(e -> {
            if (!btn.getStyle().contains("#4B6EAF")) {
                btn.setStyle("-fx-background-color: transparent; -fx-text-fill:
#A9B7C6; -fx-alignment: CENTER_LEFT; -fx-font-size: 14px; -fx-cursor: hand;");
            }
        });
        return btn;
    }

    private String getDbStructureSummary() {
        return """
               TABLE: Student
               - id (Long)
               - firstName (String)
               - lastName (String)
               - rating (Double)
               - group (Group)

               TABLE: Group
               - id (Long)
               - name (String)

               TABLE: Course
               - id (Long)
               - title (String)

               TABLE: Grade
               - id (Long)
               - score (Integer)
               - student (Student)
               - course (Course)
               """;
    }

    private void loadCurrentTaskData() {

taskDescriptionArea.setText(tasks.get(currentTaskIndex).getDescription());
        codeArea.clear();
        codeArea.setStyle("-fx-control-inner-background: #1E1E1E; -fx-text-fill:
#A9B7C6; -fx-font-family: 'Consolas'; -fx-font-size: 14px;");
        btnNextTask.setDisable(true);
        resultTable.getItems().clear();
        resultTable.getColumns().clear();
        statusLabel.setText("Очікування...");
        statusLabel.setTextFill(Color.GRAY);
    }

    private void loadNextTask() {
        if (currentTaskIndex < tasks.size() - 1) {
```

```
            currentTaskIndex++;
            showPracticeView();
        } else {
            Alert alert = new Alert(Alert.AlertType.INFORMATION);
            alert.setTitle("Вітаємо!");
            alert.setHeaderText(null);
            alert.setContentText("Ви успішно пройшли всі завдання курсу!");
            alert.showAndWait();
        }
    }

    public static void main(String[] args) { launch(args); }
}
```

Student.java

```
      package org.example;

import jakarta.persistence.*;

@Entity
@Table(name = "students")
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String firstName;
    private String lastName;


    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "group_id")
    private Group group;

    private Double rating;

    public Student() {}

    // Конструктор приймає об'єкт Group
    public Student(String firstName, String lastName, Group group, Double
rating) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.group = group;
        this.rating = rating;
    }

    public Long getId() { return id; }
    public String getFirstName() { return firstName; }
    public String getLastName() { return lastName; }

    // Головний метод
    public Group getGroup() { return group; }

    public Double getRating() { return rating; }

    @Override
    public String toString() {
        return id + " | " + firstName + " " + lastName + " (" + group.getName()
+ ")";
    }
}
```

DatabaseService.java

```
package org.example.service;

import org.example.Course;
import org.example.Grade;
import org.example.Group;
import org.example.Student;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;

import java.util.List;

public class DatabaseService {
    private SessionFactory sessionFactory;

    // Ініціалізація підключення. Роблю це один раз при старті.
    public void init() {
        try {
            sessionFactory = new
Configuration().configure().buildSessionFactory();
            initData();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // Універсальний метод для виконання будь-якого HQL запиту
    public List<?> executeQuery(String hql) {
        try (Session session = sessionFactory.openSession()) {
            Query<?> query = session.createQuery(hql, Object.class);
            return query.list();
        }
    }

    // Наповнюю базу тестовими даними, якщо вона порожня
    private void initData() {
        try (Session session = sessionFactory.openSession()) {
            session.beginTransaction();

            // Перевіряюмо, чи є вже дані, щоб не дублювати
            Long count = session.createQuery("SELECT COUNT(s) FROM Student s",
Long.class).uniqueResult();
            if (count > 0) return;

            // Створюємо групи
            Group g1 = new Group("CS-21");
            Group g2 = new Group("CS-22");
            Group g3 = new Group("F1-24");
            session.persist(g1);
            session.persist(g2);
            session.persist(g3);

            // Створюємо курси
            Course c1 = new Course("Java Core");
            Course c2 = new Course("Databases");
            Course c3 = new Course("Algorithms");
            session.persist(c1);
            session.persist(c2);
            session.persist(c3);

            // Додаємо студентів
            Student s1 = new Student("Ivan", "Petrenko", g1, 95.0);
            Student s2 = new Student("Maria", "Koval", g1, 92.5);
            Student s3 = new Student("Petro", "Bondar", g2, 88.0);
            Student s4 = new Student("Olga", "Sidorova", g2, 75.0);
```

```
            Student s5 = new Student("Max", "Verstappen", g3, 99.9);
            session.persist(s1);
            session.persist(s2);
            session.persist(s3);
            session.persist(s4);
            session.persist(s5);

            // Додаємо оцінки (використовуємо нове поле score)
            session.persist(new Grade(s1, c1, 95));
            session.persist(new Grade(s1, c2, 90));
            session.persist(new Grade(s2, c1, 88));
            session.persist(new Grade(s3, c3, 75));

            session.getTransaction().commit();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

TaskService.java

```
package org.example.service;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Predicate;

public class TaskService {
    private List<Task> tasks = new ArrayList<>();

    public TaskService() {
        initTasks();
    }

    public List<Task> getTasks() { return tasks; }

    private void initTasks() {
        // --- РІВЕНЬ 1: ОСНОВИ ---
        tasks.add(new Task(
                "ЗАВДАННЯ 1: Вибірка всіх даних.\nНапишіть запит, щоб вибрати
всіх студентів з бази.",
                "SELECT s FROM Student s",
                hql -> hql.toLowerCase().contains("student"),
                results -> results.size() >= 5
        ));

        tasks.add(new Task(
                "ЗАВДАННЯ 2: Проста фільтрація.\nВиберіть студентів, у яких
рейтинг вище 90.",
                "SELECT s FROM Student s WHERE s.rating > 90",
                hql -> hql.toLowerCase().contains("where"),
                results -> results.size() == 3
        ));

        tasks.add(new Task(
                "ЗАВДАННЯ 3: Пошук за рядком.\nЗнайдіть групу з назвою 'CS-
22'.",
                "SELECT g FROM Group g WHERE g.name = 'CS-22'",
                hql -> hql.toLowerCase().contains("group"),
                results -> !results.isEmpty()
        ));

        tasks.add(new Task(
                "ЗАВДАННЯ 4: Логічне 'АБО' (OR).\nВиберіть студентів, яких звати
```

```
'Ivan' АБО у яких рейтинг = 88.",
                "SELECT s FROM Student s WHERE s.firstName = 'Ivan' OR s.rating
= 88",
                hql -> hql.toLowerCase().contains("or"),
                results -> results.size() >= 2
        ));

        tasks.add(new Task(
                "ЗАВДАННЯ 5: Сортування (ORDER BY).\nВиведіть усіх студентів,
відсортувавши їх за прізвищем (lastName) в алфавітному порядку.",
                "SELECT s FROM Student s ORDER BY s.lastName ASC",
                hql -> hql.toLowerCase().contains("order by"),
                results -> results.size() >= 5
        ));

        // --- РІВЕНЬ 2: ЗВ'ЯЗКИ ТА АГРЕГАЦІЯ ---

        tasks.add(new Task(
                "ЗАВДАННЯ 6: Навігація по зв'язках.\nВиберіть студентів, які
навчаються в групі 'F1-24'. (Використовуйте s.group.name)",
                "SELECT s FROM Student s WHERE s.group.name = 'F1-24'",
                hql -> hql.toLowerCase().contains("group.name"),
                results -> results.size() == 1
        ));

        tasks.add(new Task(
                "ЗАВДАННЯ 7: Агрегатна функція COUNT.\nПорахуйте загальну
кількість студентів у базі.",
                "SELECT count(s) FROM Student s",
                hql -> hql.toLowerCase().contains("count"),
                results -> !results.isEmpty() &&
results.get(0).toString().equals("5")
        ));

        tasks.add(new Task(
                "ЗАВДАННЯ 8: Агрегатна функція AVG.\nЗнайдіть середній рейтинг
(avg) усіх студентів.",
                "SELECT avg(s.rating) FROM Student s",
                hql -> hql.toLowerCase().contains("avg"),
                results -> !results.isEmpty()
        ));

        tasks.add(new Task(
                "ЗАВДАННЯ 9: Робота з оцінками (Grade).\nВиберіть усі оцінки,
які дорівнюють 95 балам.",
                "SELECT g FROM Grade g WHERE g.score = 95",
                hql -> hql.toLowerCase().contains("grade"),
                results -> !results.isEmpty()
        ));

        tasks.add(new Task(
                "ЗАВДАННЯ 10: Складний запит (Chaining).\nЗнайдіть оцінки, які
отримав студент з прізвищем 'Petrenko'. (Підказка: g.student.lastName)",
                "SELECT g FROM Grade g WHERE g.student.lastName = 'Petrenko'",
                hql -> hql.toLowerCase().contains("student.lastname"),
                results -> results.size() >= 2
        ));
    }

    public static class Task {
        String description;
        String correctHql;
        Predicate<String> hqlValidator;
        Predicate<List<?>> resultValidator;

        public Task(String d, String c, Predicate<String> hv, Predicate<List<?>>
```

```
rv) {
            description = d;
            correctHql = c;
            hqlValidator = hv;
            resultValidator = rv;
        }

        public String getDescription() { return description; }
        public String getCorrectHql() { return correctHql; }

        public boolean validate(String hql, List<?> results) {
            if (hql == null || results == null) return false;
            return hqlValidator.test(hql) && resultValidator.test(results);
        }
    }
}
```

Group.java

```
package org.example;

import jakarta.persistence.*;
import java.util.List;

@Entity
@Table(name = "groups")
public class Group {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "group_name")
    private String name;

    @OneToMany(mappedBy = "group", fetch = FetchType.EAGER)
    private List<Student> students;

    public Group() {}

    public Group(String name) {
        this.name = name;
    }

    public Long getId() { return id; } // Додано!
    public String getName() { return name; }
}
```

Course.java

```
package org.example;

import jakarta.persistence.*;

@Entity
@Table(name = "courses")
public class Course {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;

    public Course() {}
```

```
    public Course(String title) {
        this.title = title;
    }

    public Long getId() { return id; } // Додано!
    public String getTitle() { return title; }
}
```

Grade.java

```
package org.example;

import jakarta.persistence.*;

@Entity
@Table(name = "grades")
public class Grade {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "student_id")
    private Student student;

    @ManyToOne
    @JoinColumn(name = "course_id")
    private Course course;

    @Column(name = "score_val")
    private Integer score;

    public Grade() {}

    public Grade(Student student, Course course, Integer score) {
        this.student = student;
        this.course = course;
        this.score = score;
    }

    // --- ГЕТТЕРИ
    public Long getId() { return id; }
    public Student getStudent() { return student; }
    public Course getCourse() { return course; }
    public Integer getScore() { return score; }
}
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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>org.example</groupId>
    <artifactId>HqlTrainer</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
```

```
        <javafx.version>17.0.6</javafx.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-controls</artifactId>
            <version>${javafx.version}</version>
        </dependency>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-fxml</artifactId>
            <version>${javafx.version}</version>
        </dependency>

        <dependency>
            <groupId>org.hibernate.orm</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>6.4.4.Final</version>
        </dependency>

        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <version>2.2.224</version>
        </dependency>

        <dependency>
            <groupId>com.jfoenix</groupId>
            <artifactId>jfoenix</artifactId>
            <version>9.0.10</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.11.0</version>
                <configuration>
                    <source>17</source>
                    <target>17</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.openjfx</groupId>
                <artifactId>javafx-maven-plugin</artifactId>
                <version>0.0.8</version>
                <configuration>
                    <mainClass>org.example.Main</mainClass>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

hibernate.cfg.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
```

```
        <property name="connection.driver_class">org.h2.Driver</property>
        <property name="connection.url">jdbc:h2:mem:testdb;DB_CLOSE_DELAY=-
1</property>
        <property name="connection.username">sa</property>
        <property name="connection.password"></property>

        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">create-drop</property> <mapping
class="org.example.Student"/>

        <mapping class="org.example.Group"/>
        <mapping class="org.example.Student"/>
        <mapping class="org.example.Course"/>
        <mapping class="org.example.Grade"/>
    </session-factory>
</hibernate-configuration>
```