МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ
Кафедра комп’ютерних наук та кібербезпеки

НЕВІРЕЦЬ ВЛАДИСЛАВ ВАСИЛЬОВИЧ

ДОСЛІДЖЕННЯ ТА РОЗРОБКА ORM-СИСТЕМИ МОВОЮ ПРОГРАМУВАННЯ JAVA

Спеціальність: 122 Комп’ютерні науки
Освітньо-професійна програма: Комп’ютерні науки та інформаційні технології
Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр»

Науковий керівник:
МАМЧИЧ ТЕТЯНА ІВАНІВНА,
кандидат фізико-математичних наук,
доцент кафедри комп'ютерних наук
та кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ
Протокол №_____
засідання кафедри комп’ютерних наук
та кібербезпеки
від _____ 2025 р.
Завідувач кафедри
(_____) Гришанович Т. О.

ЛУЦЬК – 2025

ЗМІСТ

ВСТУП

Актуальність теми. У нинішніх реаліях розробка програмного забезпечення потребує дедалі більше зусиль, адже вимоги до його якості, масштабованості та швидкодії постійно зростають. Незважаючи на це, реляційні бази даних і надалі залишаються одним із найпоширеніших способів зберігання даних. Із розширенням проєкту інтеграція з базами даних стає все складнішою та витратнішою, оскільки доводиться витрачати значну кількість часу на написання стандартних SQL-запитів. У зв’язку з цим виникає потреба у більш ефективному способі взаємодії з реляційними базами, і одним із таких рішень є використання ORM.

Об'єктно-реляційне відображення (ORM) - це техніка, яка використовується для створення "моста" між об'єктно-орієнтованими програмами і, в більшості випадків, реляційними базами даних[1]. Її реалізації, такі як Hibernate, Entity Framework, Doctrine, дозволяють автоматизувати значну частину рутинного коду, що прискорює процес розробки програмного забезпечення. Саме це є головною особливістю ORM-систем — відсутність необхідності використовувати SQL-команди в коді, що дозволяє швидко створювати мінімально життєздатний продукт (MVP).

Мета і завдання кваліфікаційної роботи. Метою кваліфікаційної роботи є розробка власної ORM-системи, яка забезпечуватиме зручну та ефективну взаємодію між об’єктно-орієнтованою моделлю та релаційною базою даних, включаючи дослідження принципів роботи ORM, огляд існуючих підходів для роботи з реляційними базами даних, а також аналіз готових реалізацій ORM.

Для досягнення поставленої мети, потрібно виконати наступні завдання:

- дослідження поняття ORM, та основні принципи цього шаблону;
- аналіз існуючих програмних реалізацій;
- вибір моделі розробки;
- програмна реалізація;
- тестування програмного продукту;

- розробка рекомендацій щодо впровадження та використання програмного засобу.

Об’єкт дослідження. Об’єктом дослідження є процес взаємодії між об’єктно-орієнтовною моделлю та реляційною базою даних. Однією з основних проблем є неповна відповідність об’єктно-реляційних моделей(object-relational impedance mismatch). Простими словами, структура даних в об’єктно-орієнтовній моделі відрізняється від реляційної моделі. Це ускладнює використання об’єктів для взаємодії з базою даних.

Предмет дослідження. Предметом дослідження є розробка та використання ORM-системи для інтеграції об’єктно-орієнтовної моделі з реляційними базами даних.

РОЗДІЛ 1.
ПОНЯТТЯ ORM ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1 Поняття та основні принципи ORM

Object-Relational mapping – технологія, яка дозволяє розробнику взаємодіяти з базою даних, як із звичайним об’єктом в програмному коді. Завдяки їй, розробник не повинен використовувати SQL для взаємодії з реляційною базою даних, оскільки він оперує об’єктами та класами, що значно спрощує та пришвидшує розробку програмного забезпечення. ORM-системи автоматично виконують трансляцію між об’єктами програми та реляційними структурами бази даних, забезпечуючи зручний і ефективний механізм збереження, оновлення та отримання даних.

ORM-система автоматично генерує та виконує SQL-запити, що значно пришвидшує розробку програмного забезпечення. Її використання значно зменшує обсяг коду, роблячи його легшим для розуміння та підтримки.

Однією з ключових переваг ORM є підтримка різних СУБД і спрощення міграції між ними, що дозволяє зосередитися на бізнес-логіці, а не на низькорівневій роботі з базами даних.

Основними принципами ORM є:

- відображення об’єктів на таблиці БД;
- абстракція від конкретної БД;
- кешування даних;
- відображення зв’язків між таблицями.

1.2. Існуючі підходи інтеграцій об‘єктних моделей з базами даних

Упродовж тривалого часу розвитку програмного забезпечення з’явилося чимало рішень для збереження даних. Серед найпоширеніших — системи SQL та NoSQL. Обидва підходи мають свої переваги й обмеження. Наприклад, SQL

краще підходить для роботи зі строго структурованими даними, тоді як NoSQL надає гнучкість у зберіганні неструктурованої інформації. Одним із ключових аспектів вибору між ними є легкість інтеграції з програмним забезпеченням.

Для роботи з NoSQL-базами даних існує багато різних фреймворків, зокрема Spring Data MongoDB. Однак головною складністю такого підходу є відсутність єдиного стандарту доступу до даних, адже кожна NoSQL-база даних має власну модель зберігання. Через це для кожного типу NoSQL-сховища зазвичай розробляється окремий інструментарій, що ускладнює впровадження нових технологій.

Зовсім інша ситуація спостерігається з реляційними базами даних. Усі вони базуються на табличній моделі: дані організовано у вигляді рядків і стовпців, а для взаємодії використовується уніфікована мова SQL. Це значно спрощує процес роботи з даними та забезпечує гнучкість у міграції між різними СУБД, адже основні принципи взаємодії залишаються незмінними.

Серед основних способів інтеграції реляційних баз даних із об’єктно-орієнтованим підходом можна виокремити три: пряме використання SQL-запитів, застосування шаблону DAO та використання ORM-систем.

У випадку прямого доступу до бази розробник самостійно створює SQL-запити та керує підключенням, використовуючи засоби, які надає мова програмування — наприклад, JDBC у Java. Цей метод надає максимальний контроль над логікою запитів і може покращити продуктивність, оскільки виключає проміжні шари. Однак така гнучкість приходить зі зростанням обсягу повторюваного коду, що ускладнює підтримку, особливо в масштабних проєктах.

DAO-підхід (Data access object), на відміну від прямого доступу, дозволяє створити окремий шар для роботи з базою даних, відділивши його від бізнес-логіки. Це сприяє кращій організації коду та його повторному використанню. Водночас, у порівнянні з ORM, DAO вимагає більше ручної реалізації, що збільшує обсяг написаного коду.

ORM-системи автоматизують взаємодію з базою даних, суттєво зменшуючи потребу у ручному написанні SQL. Вони покладаються на анотації, які описують структуру сутностей, після чого фреймворк автоматично генерує необхідні запити. Це спрощує розробку, знижує кількість помилок та прискорює розробку додатків. Але попри численні переваги, вони також мають недоліки, такі як N+1 проблема та об’єктно-реляційна невідповідність[2].

1.3 Огляд існуючих програмних розробок

Для створення власної ORM-системи, потрібно оцінити наявні рішення, проаналізувавши їхні функціональні можливості та способи реалізації.

Аналіз фреймворку «Hibernate»[12]**;**

Розробник: Red Hat;

Архітектура: Фреймворк;

Мова реалізації: Java;

Перелік функцій:

- управління транзакціями через JTA та JDBC;
- підтримка асоціацій;
- автоматична генерація SQL-запитів;
- автоматична генерація таблиць;
- управління сутностями.

Переваги фреймворку «Hibernate»**:**

- масштабованість;
- підтримка кешування;
- кросплатформність;
- широка підтримка баз даних.

Недоліки фреймворку «Hibernate»**:**

- складність для початківців;
- зменшення продуктивності в порівнянні із використанням SQL;
- неправильне налаштування кешу може спричинити втрату даних;

- N+1 проблема [11].

Рисунок 1.1 – Створення сутності в «Hibernate»

```
public static void main(String[] args) {
    EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "learn");
    EntityManager em = emf.createEntityManager();
    em.getTransaction().begin();
    User user = new User();
    user.setName("David");
    user.setEmail("david@mail.com");
    em.persist(user);
    List<User> users = em.createQuery( s: "select u from User u", User.class).getResultList();
    users.forEach(u->System.out.println(u.getName()));

    em.getTransaction().commit();
    em.close();
    emf.close();
}
```

Рисунок 1.2 – Приклад використання «Hibernate»

Аналіз фреймворку «EntityFramework»[3]**;**

Розробник: Microsoft;

Архітектура: Фреймворк;

Мова реалізації: C#;

Перелік функцій:

- автоматична генерація таблиць;
- підтримка транзакцій;
- підтримка асинхронних операцій;
- управління сутностями.

Переваги фреймворку «EntityFramework»**:**

- генерація міграцій;
- тісна інтеграція з .NET;
- підтримка LINQ [6].

Недоліки фреймворку «EntityFramework»**:**

- відсутність повної підтримки складних запитів;
- низька продуктивність;
- невелика гнучкість у налаштуванні схем.

```
public class Book
{
    1 usage
    public int Id { get; set; }
    3 usages
    public string Title { get; set; } = string.Empty;
    3 usages
    public string Author { get; set; } = string.Empty;
    2 usages
    public DateTime PublishedDate { get; set; }
    4 usages
    public decimal Price { get; set; }
}
```

Рисунок 1.3 – Створення моделі в «EntityFramework»

Аналіз фреймворку «SQLAlchemy»[9]**.**

Розробник: SQLAlchemy Project.

Архітектура: Бібліотека.

Мова реалізації: python.

```
static void CreateBooks()
{
    using (var context = new AppDbContext())
    {
        var books = new List<Book>
        {
            new Book { Title = "Clean Code", Author = "Robert C. Martin", PublishedDate = new DateTime( year: 2008,  month: 8,  day: 1), Price = 25.99m },
            new Book { Title = "The Pragmatic Programmer", Author = "Andrew Hunt", PublishedDate = new DateTime( year: 1999,  month: 10,  day: 20), Price = 30.99m }
        };

        context.Books.AddRange(books);
        context.SaveChanges();
        Console.WriteLine("Books created!");
    }
}
1 usage
static void ReadBooks()
{
    using (var context = new AppDbContext())
    {
        var books List<Book> = context.Books.ToList();
        Console.WriteLine("Books in the database:");
        foreach (var book in books)
        {
            Console.WriteLine($"ID: {book.Id}, Title: {book.Title}, Author: {book.Author}, Price: {book.Price}");
        }
    }
}
```

Рисунок 1.4 – Методи для взаємодії з базою даних в «EntityFramework»

Аналіз фреймворку «SQLAlchemy»[9]**.**

Розробник: SQLAlchemy Project;

Архітектура: Бібліотека;

Мова реалізації: python;

Перелік функцій:

- підтримка декларативного та імперативного стилів моделювання;
- підтримка транзакцій;
- потужний SQL-конструктор;
- підтримка асинхронного програмування.

Переваги фреймворку « SQLAlchemy »**:**

- гнучка інтеграція з існуючими SQL-запитами;
- велика спільнота та багата документація;
- висока продуктивність та контроль над SQL.

Недоліки фреймворку « SQLAlchemy »**:**

- високий поріг входження для новачків;

- може вимагати багато конфігурації для складних моделей;
- асинхронність підтримується лише частково.

```python
# Базовий клас для моделей
Base = declarative_base()

# Модель User
class User(Base):  3 usages
    __tablename__ = 'users'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    email = Column(String)

    def __repr__(self):
        return f"<User(name='{self.name}', email='{self.email}')>"

# Створення SQLite-бази в пам'яті
engine = create_engine('sqlite:///example.db', echo=True)

# Створення таблиць
Base.metadata.create_all(engine)

# Сесія для роботи з БД
Session = sessionmaker(bind=engine)
session = Session()

# Додавання нових користувачів
user1 = User(name='Іван', email='ivan@example.com')
user2 = User(name='Олена', email='olena@example.com')

session.add(user1)
session.add(user2)
session.commit()

# Запит до БД
users = session.query(User).all()
for user in users:
    print(user)

# Закриття сесії
session.close()
```

Рисунок 1.5 – Приклад використання «SQLAlchemy»

Аналіз фреймворку «Sequelize»[8].

Розробник: Sequelize Contributors.

Архітектура: Бібліотека.

Мова реалізації: JavaScript/TypeScript.

Перелік функцій:

- асинхронна робота з базою даних;
- підтримка транзакцій;
- зв'язки між моделями;
- підтримка валідаторів.

Переваги фреймворку « Sequelize »:

- генерація міграцій;
- підтримка різних СУБД;
- генерація з популярними фреймворками (Express, NestJS).

Недоліки фреймворку « Sequelize »:

- обмежена підтримка складних SQL-запитів;
- іноді неочевидна обробка помилок;
- менш гнучкий, ніж Knex.js або TypeORM у налаштуваннях.

```
const User = sequelize.define('User', {
  username: {
    type: DataTypes.STRING,
    allowNull: false
  },
  age: {
    type: DataTypes.INTEGER
  }
}, {
  timestamps: true
});
```

Рисунок 1.6 – Створення сутності в « Sequelize »

```
(async () => {
  try {
    await sequelize.authenticate();
    console.log('Підключення до бази даних успішне.');

    // Синхронізація моделі з БД (створить таблицю, якщо її нема)
    await sequelize.sync({ force: true });

    // Створення нового користувача
    const user = await User.create({ username: 'Ivan', age: 25 });
    console.log('Користувача створено:', user.toJSON());

    // Отримання всіх користувачів
    const users = await User.findAll();
    console.log('Всі користувачі:', users.map(u => u.toJSON()));
  } catch (error) {
    console.error('Помилка:', error);
  } finally {
    await sequelize.close();
  }
})();
```

Рисунок 1.7 – Приклад використання « Sequelize »

Аналіз фреймворку «GORM»[4].

Розробник: JetBrains.

Архітектура: Фреймворк.

Мова реалізації: Go.

Перелік функцій:

- автоматичне управління схемою;
- управління сутностями;
- налаштування поведінки запитів;
- запити з фільтрацією та сортуванням;
- робота із зв’язками.

Переваги фреймворку «GORM»**:**

- широка підтримка баз даних;
- активна спільнота розробників;
- підтримка транзакцій;

- автоматична міграція.

Недоліки фреймворку «GORM»:

- важкий у налаштуванні;
- малий контроль над запитами;
- менша гнучкість у порівнянні з іншими ORM;
- проблеми з масштабуванням;
- N+1 проблема [11].

```go
type User struct {
    ID    uint   `gorm:"primaryKey"`
    Name  string `gorm:"type:varchar(100);not null"`
    Email string `gorm:"type:varchar(100);unique"`
}
```

Рисунок 1.8 – Створення моделі в «GORM»

```go
// Створення нового користувача
user := User{Name: "John Doe", Email: "john.doe@example.com"}
db.Create(&user)

// Вибірка користувача за ID
var retrievedUser User
if err := db.First(&retrievedUser, 1).Error; err != nil {
    fmt.Println("Помилка вибірки користувача:", err)
    return
}

// Виведення знайденого користувача
fmt.Printf("Знайдений користувач: %v\n", retrievedUser)
```

Рисунок 1.9 – Методи для взаємодії з базою даних в «GORM»

Огляд наявних реалізацій технології ORM дав змогу зробити низку важливих висновків, які стануть у пригоді під час створення власного програмного продукту. Кожна ORM-система має свої переваги та недоліки.

Одним із ключових аспектів є архітектура реалізації ORM. У більшості випадків аналізовані ORM-системи побудовані у форматі фреймворку. Але також є реалізації побудовані у вигляді бібліотек Основна відмінність між цими підходами полягає в тому, що бібліотека надає набір інструментів, якими розробник користується на власний розсуд, тоді як фреймворк зазвичай визначає структуру та спосіб взаємодії з базою даних. Таким чином, бібліотека краще підходить для невеликих проєктів, тоді як фреймворк — для побудови масштабних додатків.

Важливо звертати увагу на те, наскільки ORM-фреймворк дотримується загальноприйнятих стандартів, таких як JPA (Java Persistence API) у Java чи Active Record у Ruby. Сумісність зі сторонніми інструментами, бібліотеками та базами даних значно спрощує інтеграцію у великі проєкти.

Не менш важливим є аналіз переваг і недоліків існуючих рішень. Такий аналіз дозволяє визначити, який функціонал повинен мати ORM, а також врахувати слабкі місця інших реалізацій і уникнути повторення їхніх помилок. Це сприятиме створенню більш ефективної та зручної ORM-системи, яка відповідатиме сучасним вимогам розробників.

РОЗДІЛ 2.
ПРОЄКТУВАННЯ ТА РОЗРОБКА ORM-СИСТЕМИ

2.1. Постановка задачі, призначення та вимоги до програмного засобу “MapFlow”

Основною задачею бакалаврської роботи є створення ORM-системи, яка надає розробникам зручний та ефективний інструмент для взаємодії з реляційними базами даних. Основна мета програмного засобу є автоматизація написання SQL-команд для пришвидшення розробки програмного забезпечення, за рахунок уніфікації запитів, та зменшення ручної роботи при взаємодії з БД.

Важливо реалізувати наступний функціонал в програмному засобі “MapFlow”:

- автоматична генерація SQL-запитів;
- автоматична генерація запиту для створення таблиці;
- менеджер для роботи з сутностями;
- механізм для відображення сутності на таблицю в БД;
- механізм кешування;
- механізм логування;
- механізм ігнорування поля класу при створенні таблиці в БД;
- генерація ідентифікатора сутності;
- можливість зручного налаштування ORM.

Програмний засіб “MapFlow” призначений для розробників, які прагнуть скоротити час розробки програмного забезпечення. Завдяки “MapFlow” взаємодія з базою даних відбувається за допомогою об’єктів та методів мови програмування, що дозволяє уникнути необхідності ручного написання SQL-запитів. А також використання даного програмного засобу знижує кількість запитів до бази даних за рахунок механізму кешування;

2.2. Вибір моделі розробки програмного засобу “MapFlow”

Важливим етапом перед початком розробки програмного забезпечення є вибір моделі розробки, адже саме від цього залежить ефективність, масштабованість і якість кінцевого продукту. Правильно підібрана модель дозволяє зменшити ризики, покращити керованість процесом і досягти кращих результатів у коротші строки.

Однією з таких моделей є ітеративна модель, у якій процес розробки розбивається на послідовні цикли (ітерації). Кожна ітерація включає планування, розробку, тестування та перегляд результатів. Після завершення кожного циклу продукт доповнюється новим функціоналом, і на кожному етапі існує працездатна версія системи. Це дає можливість регулярно перевіряти якість продукту та вчасно вносити корективи.

Ітеративна модель забезпечує гнучкість у розробці: спочатку команда зосереджується на реалізації базової функціональності, а потім поступово розширює систему відповідно до потреб користувачів. Такий підхід особливо корисний у випадках, коли важливо швидко випустити першу робочу версію продукту (Minimum Viable Product), яка вже вирішує основні проблеми, а подальші оновлення вдосконалюють систему.

Рисунок 2.1 – Ітеративна модель розробки

Для розробки програмного засобу “MapFlow” було обрано ітеративну модель з кількох ключових причин. Перш за все, цей підхід дає змогу поступово додавати функціональні можливості, що дозволяє проводити регулярне тестування кожного компонента та швидко отримувати зворотний зв’язок від користувачів. Завдяки цьому можливе своєчасне виявлення та усунення помилок, а також вдосконалення продукту до потреб користувачів ще на ранніх етапах розробки.

Другою важливою перевагою ітеративного підходу є зниження ризиків. Після кожної ітерації користувачі мають доступ до оновленої версії програмного засобу, що дозволяє розробнику краще зрозуміти очікування цільової аудиторії. Це дає змогу зосередитись на розробці необхідного функціоналу, а також заощадити ресурси та створити стабільний, зручний і практичний продукт.

2.3. Загальний опис проєкту “MapFlow”

Основними компонентами програмного засобу “MapFlow” є: EntityManager, QueryBuilder, CacheManager, IdGenerator, PropetryManager та LogManager, а також анотації, які служать для того, щоб пов’язати поля об’єктів з стовпцями в таблиці бази даних.

EntityManager – це інтерфейс, який відповідає за управління життєвим циклом сутностей в контексті бази даних. Він містить методи для маніпулювання сутністю (create, read, update, delete) та створює відповідну до сутності таблицю, якщо вона відсутня.

QueryBuilder – інструмент, який на основі анотованої сутності генерує відповідний SQL-запит до бази даних.

CacheManager – інтерфейс, який відповідає за управління кешем. Він має дві реалізації: InMemoryCacheManager та RedisCacheManager. Завдяки цьому оптимізується взаємодія з базою даних.

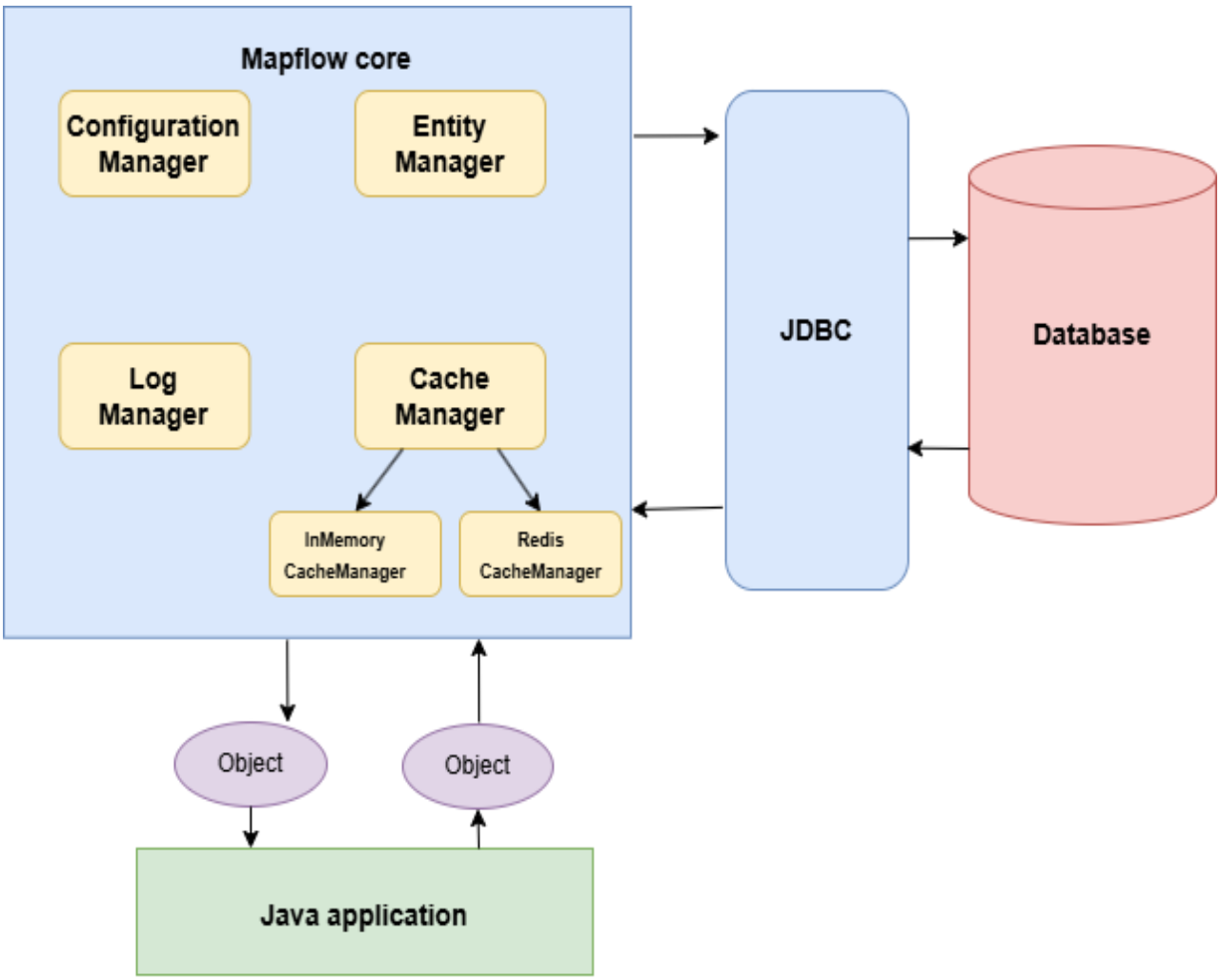

Рисунок 2.2 – Архітектура програмного засобу “MapFlow”

IdGenerator – механізм який генерує ідентифікатор для сутностей.

PropertyManager – інструмент, який дозволяє зручно налаштовувати ORM-систему, завдяки використанню application.properties файлу. Це значно зменшує кількість коду, який потрібен для налаштування ORM.

LogManager – механізм, який дозволяє відслідковувати дії, які робить ORM-система.

2.4. Обґрунтування вибору інструментальних засобів розробки “MapFlow”

Для розробки програмного засобу “MapFlow” було обрано мову програмування Java. Вона є поширеним вибором для розробки різноманітних

проєктів в яких використовуються бази даних, відповідно інструмент для взаємодії буде досить доречним. Також вона є об’єктно-орієнтовною мовою програмування, та має зручний та зрозумілий синтаксис.

Також в Java є ряд технологій, таких як ReflectionApi та анотації, що полегшать процес розробки ORM-системи.

2.5. Особливості програмної реалізації програмного засобу“MapFlow”

В основі програмного засобу “MapFlow” є анотації. Вони слугують для того, щоб визначити, як класи та поля об'єктів у коді відповідають таблицям та стовпцям у базі даних.

Даний програмний засіб має 5 анотацій: @Entity, @Table, @Id, @Column, @Transient.

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface Entity {
}
```

Рисунок 2.3 – Анотація @Entity

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface Table {
    String name();
}
```

Рисунок 2.4 – Анотація @Table

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface Id {
}
```

Рисунок 2.5 – Анотація @Id

```
@Retention(RetentionPolicy.RUNTIME)  8 usages
@Target(ElementType.FIELD)
public @interface Column {
    String name() default "";
    String type() default "";
    boolean primaryKey() default false;  1 usage
    boolean nullable() default true;  1 usage
}
```

Рисунок 2.6 – Анотація @Column

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface Transient {
}
```

Рисунок 2.7 – Анотація @Transient

Анотації слугують для отримання даних про об’єкт під час формування SQL-запиту.QueryBuilder отримує назву та тип поля, а також чи це поле є первиним ключем та чи може воно містити null значення в базі даних.

Також важливо врахувати момент, коли для бізнес логіки потрібно мати певне поле, але його не обов’язково зберігати в базі даних. Для цього і потрібна анотація @Transient. Завдяки їй QueryBuilder розуміє, що певне поле не повинно бути збережене в базу даних, відповідно воно не буде використано під час формування SQL-запиту.

```
public static String insertQuery(Object entity) {  1 usage
    String entityName;
    //Отримання інформації про клас сутності
    Class<?> clazz = entity.getClass();
    //Створення запиту
    StringBuilder query = new StringBuilder("INSERT INTO ");
    //Отримання назви таблиці бд та додавання її до запиту
    Table table = clazz.getAnnotation(Table.class);
    if (table == null) {
        entityName = clazz.getSimpleName();
    }else {
        entityName = table.name();
    }
    query.append(entityName).append(" (");

    Field[] fields = clazz.getDeclaredFields();
    List<String> columns = new ArrayList<>();
    List<String> values = new ArrayList<>();

    //Отримання назв колонок із сутності та отримання данних на вставку з Object entity
    for (Field field : fields) {
        field.setAccessible(true);
        if (field.isAnnotationPresent(Transient.class)) {
            continue;
        }
        columns.add(EntityUtilties.getColumnName(field));

        try {
            var value = field.get(entity);
            if (value instanceof Number) {
                values.add(value.toString());
            }else if(value==null) {
                values.add(null);
            }else {
                values.add("'" + value.toString() + "'");
            }
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }


    }
    //додавання назв колонок та данних в запит
    query.append(String.join( delimiter: ", ", columns));
    query.append(") VALUES (");
    query.append(String.join( delimiter: ", ", values));
    query.append(");");

    return query.toString();
}
```

Рисунок 2.8 – Метод insertQuery() для генерації запиту на додавання нового об’єкта в базу даних

Програмний засіб “MapFlow” має клас QueryBuilder, основна задача якого формувати запити, такі як create, read, update, delete, а також createTableIfNotExist.

Даний метод отримує на вхід об’єкт типу Object, який містить об’єкт, який потрібно зберегти в базі даних. З цього об’єкта функція отримує об’єкт типу Class. Це необхідно для того, щоб за допомогою рефлексії дізнатись інформацію про поля, анотації і т.д. Після отримання необхідної інформації, завдяки StringBuilder формується SQL-запит.

Після формування, запит виконується в класі EntityManagerImpl. Цей клас управляє життєвим циклом сутності. Він дозволяє читати, додавати, оновлювати та видаляти записи з таблиці бази даних.

```
public void create(T entity) {
    EntityUtilties.validateEntityClass(clazz);
    Field identifierField = EntityUtilties.getIdField(clazz);
    try {
        identifierField.setAccessible(true);
        Object identifierValue = identifierField.get(entity);

        if (identifierValue == null || (identifierValue instanceof Number && ((Number) identifierValue).longValue() == 0)) {
            IdGenerator generator = new IdGenerator(connection, EntityUtilties.getTableName(clazz));
            Long newId = generator.nextId();
            identifierField.set(entity, newId);
        }
    } catch (IllegalAccessException e) {
        throw new RuntimeException("Cannot access @Id field in " + clazz.getSimpleName(), e);
    }
    String query = CrudQueryBuilder.insertQuery(entity);
    LogManager.log( message: "Entity manager",query);
    try (Statement statment = connection.createStatement()) {

        statment.executeUpdate(query);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Рисунок 2.9 – Метод для отримання даних з БД

Цей клас використовує узагальнення (generics), для **забезпечення** типобезпеки та зручності роботи з різними типами сутностей. Це гарантує що клас буде працювати з об’єктами певного типу.

Також важливим елементом ORM-системи є механізм кешування об’єктів. В програмному засобі “MapFlow” є інтерфейс CacheManager, який містить такі методи як: put, get, remove. Цей інтерфейс реалізують два класи: InMemoryCacheManager та RedisCacheManager. Вони дозволяють зберігати кеш в пам’яті та в Redis відповідно.

В якості структури даних для кешу, який зберігається в оперативній пам’яті, було використано Map. В якості ключа, вона приймає об’єкт типу CacheKey, а в якості значення - об’єкт класу CacheEntry.

CacheKey має два поля: id об’єкта, та clazz. Це гарантує унікальність ідентифікації кешованого об’єкта, що є важливим при роботі з кількома сутностями.

CacheEntry – клас, який містить об’єкт, який потрібно кешувати, та час життя об’єкта в кеші.

Для більш комфортного використання програмного засобу, було реалізовано клас MapFlowProperties. Його задача полягає в тому, щоб отримувати конфігурацію ORM. Це дозволяє розробнику швидше почати використовувати програмний продукт.

Даний клас отримує з файлу “appilication.properties” такі налаштування як:

- mapflow.cache.type - отримання типу кешування;
- mapflow.cache.ttl - час життя об’єкту в кеші;
- mapflow.redis.host - отримання хосту Redis;
- mapflow.redis.port - отримання порта Redis;
- mapflow.redis.password - отримання паролю до Redis;
- mapflow.application.logging - виводити логи в консоль.

```
public class InMemoryCacheManager<T> implements CacheManager<T> {  1 usage

    private static class CacheEntry<T> {  3 usages
        private final T value;  2 usages
        private final long expireAt;  2 usages

        CacheEntry(T value, long expireAt) {  1 usage
            this.value = value;
            this.expireAt = expireAt;
        }
    }

    private final Map<CacheKey, CacheEntry<T>> cache = new ConcurrentHashMap<>();  4 usag

    @Override  1 usage
    public void put(CacheKey key, T value, int ttlInSeconds) {
        long expireAt = System.currentTimeMillis() + ttlInSeconds * 1000L;
        cache.put(key, new CacheEntry<>(value, expireAt));
    }

    @Override  1 usage
    public T get(CacheKey key) {
        CacheEntry<T> entry = cache.get(key);
        if (entry == null || System.currentTimeMillis() > entry.expireAt) {
            cache.remove(key);
            return null;
        }
        return entry.value;
    }

    @Override  no usages
    public void remove(CacheKey key) {
        cache.remove(key);
    }
}
```

Рисунок 2.10 – Клас InMemoryCacheManager

Також в програмному засобі “MapFlow” присутня система логування. Вона дозволяє розробнику бачити які дії виконує ORM в певний проміжок часу.

За логування відповідальний клас LogManager. Він містить два статичних методи “log()”, один з яких приймає строку, а інший строку та об’єкт, який потрібно логувати

Систему логування можна використовувати, задавши параметру mapflow.application.logging значення “true”.

```
public class LogManager {  7 usages
    private static Boolean isLoggingEnabled;  3 usages

    static {
        MapflowProperty property = new MapflowProperty();
        isLoggingEnabled = Boolean.parseBoolean(property.getIsLoggingEnabled());
    }
    public static void log(String message) {  2 usages
        if(isLoggingEnabled){
            System.out.println(message);
        }
    }
    public static void log(String message, Object value) {  4 usages
        if(isLoggingEnabled){
            System.out.println(message+" : "+value);
        }
    }
}
```

Рисунок 2.11 – Клас “LogManager”

Одним з основних класів в “MapFlow” є “ConfigurationManager”. Він відповідає за автоматичне з’єднання ORM з реляційною базою даних. Він містить два конструктора та єдиний метод “getConnection()”, який повертає з’єднання з базою даних.

```java
public class ConfigurationManager {  no usages
    private final String url;  3 usages
    private final String username;  3 usages
    private final String password;  3 usages
    private final String driver;  3 usages


    public ConfigurationManager() {  no usages

        Properties properties = new Properties();
        //Отримання конфігурації з application.properties
        try (InputStream input = getClass().getClassLoader().getResourceAsStream( name: "application.properties")) {
            this.url = properties.getProperty("mapflow.datasource.url");
            this.username = properties.getProperty("mapflow.datasource.username");
            this.password = properties.getProperty("mapflow.datasource.password");
            this.driver = properties.getProperty("mapflow.datasource.driver");
            if (input == null) {
                throw new RuntimeException("File 'application.properties' not found in resources!");
            }
            properties.load(input);


        } catch (IOException e) {
            throw new RuntimeException("Failed to load database configuration", e);
        }
    }

    public ConfigurationManager(String url, String username, String password, String driver) {  no usages
        this.url = url;
        this.username = username;
        this.password = password;
        this.driver = driver;

    }


    public Connection getConnection() throws SQLException {  no usages
        try {
            Class.forName(driver);
        } catch (ClassNotFoundException e) {
            throw new SQLException("Database driver not found", e);
        }
        return DriverManager.getConnection(url, username, password);
    }

}
```

Рисунок 2.12 – Клас “ConfigurationManager”

Налаштувати з’єднання з базою даних можна використовуючи “application.properties” файл, та передавши налаштування в конструктор даного класу.

Метод getConnection() створює та повертає з’єднання з базою даних, використовуючи JDBC (Java Database Conectivity).

2.6. Організація тестування та налагодження програмного засобу “MapFlow”

В результаті тестування програмного засобу “MapFlow” було перевірено роботу всіх CRUD-функцій, систему кешування та логування. А також автоматичне створення таблиці в базі даних. Результат тестування є позитивним, оскільки програмний засіб успішно справився з основними функціями роботи з базами даних.

Основними недоліками для повноцінної роботи “MapFlow” є відсутність функціоналу для роботи зі зв’язками між сутностями.

2.7. Рекомендація по використанню та впровадженню програмного засобу “MapFlow”

Цей програмний засіб допомагає розробнику делегувати управління базою даних, дозволяючи більше зосередитись на реалізації бізнес-логіки. “MapFlow” особливо ефективний для невеликих проєктів, де необхідна швидка та ефективна взаємодія з базою даних. Крім того, він є корисним інструментом для навчання, оскільки дозволяє наочно демонструвати основні принципи роботи ORM.

Для використання програмного засобу “MapFlow” потрібно у проєкті створити конфігураційний файл “application.properties”, який повинен містити наступні параметри для з’єднання з базою даних:

- mapflow.datasource.url- URL для підключення до бази даних;

- mapflow.datasource.username-ім’я користувача, для аутентифікації при підключенні до БД;
- mapflow.datasource.password- пароль для аутентифікації;
- mapflow.datasource.driver-JDBC драйвер для роботи з БД (org.postgresql.Driver, com.mysql.cj.jdbc.Driver і тд.).

```
mapflow.datasource.url=jdbc:postgresql://localhost:5432/your_database
mapflow.datasource.username=username
mapflow.datasource.password=password
mapflow.datasource.driver = org.postgresql.Driver
```

Рисунок 2.13 – Приклад файлу конфігурації

Також конфігураційний файл може містити параметри для управління логуванням, кешуванням, та підключенням до Redis.

Після налаштування конфігураційного файлу можна починати працювати з ORM. Спочатку потрібно створити об’єкт класу “ConfigurationManager”, а також об’єкт “EntityManager”,Який повинен приймати метод getConnection() з “ConfigurationManager”, а також об’єкт типу “Class”.

```
ConfigurationManager manager = new ConfigurationManager();  1 usage
EntityManager<Users> em = new EntityManagerImpl<>(manager.getConnection(), Users.class);  1 usage
```

Рисунок 2.13 – Приклад створення об’єкту “EntityManager”

На цьому етапі ORM розробник може використовувати ORM для розробки програмного забезпечення.

ВИСНОВОК

У сучасній розробці програмного забезпечення ефективна взаємодія з базами даних є надзвичайно важливою, особливо в контексті великих і масштабованих проєктів. Одним із провідних підходів до такої взаємодії є використання ORM-систем, які дозволяють мінімізувати ручне написання SQL-запитів та зосередитися на бізнес-логіці застосунку. У межах даної дипломної роботи було проведено аналіз існуючих ORM-рішень, виявлено їх переваги й недоліки, серед яких — спрощення розробки, але водночас і певне зниження продуктивності, зокрема через проблему N+1 запитів та не завжди оптимальні SQL-операції.

Зважаючи на проведене дослідження, було поставлено практичне завдання — створити власну ORM-систему під назвою “MapFlow”, яка забезпечує автоматичне перетворення об’єктів у реляційні представлення та навпаки. Система має на меті зробити взаємодію з базою даних інтуїтивно зрозумілою та максимально зручною для розробника. Під час реалізації “MapFlow” використовувалися такі технології, як Reflection API для динамічного доступу до полів і анотацій класів, StringBuilder для генерації SQL-запитів, а також механізми роботи з конфігураційними файлами для налаштування структури системи.

У процесі розробки було виконано всі поставлені завдання: досліджено принципи ORM і DAO, проаналізовано існуючі програмні реалізації, обрано модель розробки системи, реалізовано функціональну частину з підтримкою CRUD-операцій та основних типів зв’язків між сутностями, проведено тестування працездатності “MapFlow”, сформовано рекомендації щодо використання програмного продукту.

Розробка “MapFlow” дозволила не лише створити приклад власної ORM-системи, але й поглибити розуміння принципів об’єктно-реляційного

відображення, особливостей роботи з базами даних та внутрішніх механізмів мови Java. Створена система може слугувати базою для подальшого розширення, зокрема додавання кешування, оптимізаторів запитів, підтримки транзакцій, а також інтеграції з іншими модулями програмних систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Abba I. What is an ORM – the meaning of object relational mapping database tools. *freeCodeCamp.org*. URL: https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/ (дата звернення: 06.03.2025).
2. Bhushan S. Object relational impedance mismatch. *Medium*. URL: https://medium.com/booleanbhushan/object-relational-impedance-mismatch-28fc2440dee4 (дата звернення: 06.03.2025).
3. Entity Framework documentation hub. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/ef/ (дата звернення: 06.03.2025).
4. GORM guides. *GORM*. URL: https://gorm.io/docs/index.html (дата звернення: 06.03.2025).
5. Java JDBC API. *Moved*. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/ (дата звернення: 06.03.2025).
6. Language integrated query (LINQ) - C#. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/dotnet/csharp/linq/ (дата звернення: 06.03.2025).
7. Rachev P. My issue with orms. *Preslav Rachev*. URL: https://preslav.me/2023/05/15/my-issue-with-orm/ (дата звернення: 06.03.2025).
8. Sequelize. *Sequelize | Feature-rich ORM for modern TypeScript & JavaScript*. URL: https://sequelize.org/ (дата звернення: 27.05.2025).
9. SQLAlchemy. *SQLAlchemy - The Database Toolkit for Python*. URL: https://www.sqlalchemy.org/ (дата звернення: 06.03.2025).
10. Use Object-Relational Mapping (ORM) in software development! Cleverly in software development!. *Rock the Prototype - Softwareentwicklung & Prototyping*. URL: https://rock-the-prototype.com/en/software-development/object-relational-

mapping-orm/#:~:text=Basic%20concepts:%20The%20core%20principles,data%20manipulation%20and%20better%20maintainability. (дата звернення: 06.03.2025).

11. What is the N+1 query problem and how to solve it? Â planetscale. *PlanetScale - the worldâs fastest and most reliable relational database*. URL: https://planetscale.com/blog/what-is-n-1-query-problem-and-how-to-solve-it (дата звернення: 06.03.2025).
12. Your relational data. objectively. - hibernate ORM. *Hibernate*. URL: https://hibernate.org/orm/ (дата звернення: 15.03.2025).

ДОДАТКИ

Додаток А

ТЕХНІЧНЕ ЗАВДАННЯ

На розробку ORM-системи мовою програмування Java

ВСТУП

Програмний засіб призначений для розробників. Він дозволяє значно зменшити час розробки ПЗ, та зменшити кількість коду, оскільки не потрібно писати рутинні запити до БД.

ПІДСТАВИ ДЛЯ РОЗРОБКИ

Роботи з розробки програмного засобу «MapFlow» здійснюються на підставі написання курсової роботи.

ПРИЗНАЧЕННЯ РОЗРОБКИ

Програмний засіб призначений для розробників, мета яких – скоротити час розробки програмного забезпечення, шляхом використання ефективного інструменту для інтеграції з реляційними базами даних.

ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Вимоги до функціональних характеристик:

- управління сутностями за допомогою анотацій;
- зручна конфігурація підключення до БД;
- автоматична генерація SQL-запитів;
- компонент, що відповідає за маніпулювання сутностями;
- механізм кешування;
- механізм логування;
- генерація ідентифікатора;
- автоматична генерація запиту для створення таблиці.

Вимоги до надійності: Програмний засіб повинен ефективно працювати при збільшенні навантаження.

Умови експлуатації: програмний засіб сумісний із будь якою СУБД.

Додаток В

КЕРІВНИЦТВО КОРИСТУВАЧУ ПРОГРАМНОГО ЗАСОБУ “MapFlow”

1. Загальні відомості

Програмний засіб “MapFlow”– це інструмент, який надає можливість взаємодіяти з реляційними базами даних без написання SQL-запитів.

2. Функціональне призначення

Програмний засіб призначений для зменшення коду, та пришвидшення розробки програмного забезпечення. Розробник може створювати, отримувати, оновлювати та видаляти дані з бази даних, без знання SQL.

3. Умови застосування програми

Програмний засіб використовує мову програмування java. Тому умовою його застосування є наявність IDE для створення java проєкту, та наявність реляційної бази даних (PostgreSQL, MySQL тощо).

4. Опис роботи програми

Для початку розробник повинен створити файл “application.properties”. Він відповідає за конфігурацію бази даних. В ньому розробник повинен вказати параметри для з’єднання з БД:

- mapflow.datasource.url;
- mapflow.datasource.username;
- mapflow.datasource.password;
- mapflow.datasource.driver.

Наступним кроком буде створення об’єкту класу ConfigurationManager. Він створює з’єднання, використовуючи параметри, які розробник вказав у файлі “application.properties”. Останнім кроком буде створення EntityManager якому потрібно передати створений об’єкт ConfigurationManager. Після цього можна використовувати функцій EntityManager для маніпулювання таблицею БД.

Також є можливість налаштування кешування та логування.

Для того щоб використовувати кешування, потрібно налаштувати наступні параметри в “application.properties” файлі:

- mapflow.cache.type - отримує тип кешування (IN_MEMORY, REDIS, REDIS_DEFAULT)
- mapflow.cache.ttl - час зберігання об’єкту в системі кешування
- mapflow.redis.host - хост redis-сервера, для REDIS-кешування
- mapflow.redis.port -порт redis-сервера, для REDIS-кешування
- mapflow.redis.password - пароль redis-сервера, для REDIS-кешування.

Для того, щоб увімкнути механізм логування, потрібно передати параметр true для mapflow.application.logging.

АНОТАЦІЯ

Невірець Владислав Васильович – Дослідження та розробка ORM-системи – Рукопис.

Кваліфікаційна робота за спеціальністю 122 Комп'ютерні науки. – Волинський національний університет імені Лесі Українки, Луцьк. – 2025 р.

Робота присвячена дослідженню принципів побудови ORM-систем (Object-Relational Mapping) та розробці власної ORM-бібліотеки мовою програмування Java. У роботі розглянуто актуальність використання ORM-технологій для спрощення взаємодії між об'єктно-орієнтованими програмами та реляційними базами даних. Проаналізовано особливості сучасних ORM-фреймворків, таких як Hibernate, та виявлено їх сильні та слабкі сторони. Метою роботи є створення власної легкої та гнучкої ORM-системи, яка забезпечує базові CRUD-операції, кешування даних в Redis та inMemory, а також мінімальну конфігурацію з боку розробника. Реалізація системи здійснена з дотриманням принципів чистої архітектури та SOLID-підходу. Розробку та тестування програмного забезпечення виконано в середовищі IntelliJ IDEA з використанням мови Java та СУБД PostgreSQL. У процесі реалізації було також приділено увагу серіалізації даних, управлінню транзакціями та роботі з анотаціями. Завдання, поставлені у кваліфікаційній роботі, включають аналіз існуючих рішень, проєктування архітектури власної ORM-системи, реалізацію основних функціональних модулів, забезпечення підтримки зв'язків між об'єктами та оцінку ефективності системи в тестових середовищах.

Ключові слова: ORM, Java, реляційна база даних, Hibernate, CRUD, анотації, PostgreSQL, EntityManager, мапінг сутностей, зв'язки між таблицями, автоматизація збереження даних.