НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
ім. Ігоря Сікорського

Навчально-науковий комплекс «Інститут прикладного системного аналізу»
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
__________ А.І.Петренко
(підпис) (ініціали, прізвище)
"___"_____________20__ р.

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки 6.050101 Комп'ютерні науки
(код і назва)

на тему: Застосування об'єктно-орієнтованих баз даних в інформаційних системах

Виконав (-ла): студент (-ка) 4 курсу, групи ДА-32
(шифр групи)

Токарський Андрій Олегович ________
(прізвище, ім'я, по батькові) (підпис)

Керівник ас., к.т.н. Свірін П.В ________
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант економічний доцент, к.е.н. Рощина Н.В. ________
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент асистент Кухарєв С.О. ________
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль старший викладач Бритов О.А. ________
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____________
(підпис)

Київ – 2017 року

Національний технічний університет України
«Київський політехнічний інститут»
ім. Ігоря Сікорського

Інститут (факультет) ННК «Інститут прикладного системного аналізу
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки 6.050101 Комп’ютерні науки
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
__________ А.І.Петренко
(підпис) (ініціали, прізвище)

«___»____________20__ р.

ЗАВДАННЯ
на дипломну роботу студенту

Токарському Андрію Олеговичу
(прізвище, ім’я, по батькові)

1. Тема роботи Застосування об’єктно-орієнтованих баз даних в інформаційних системах ,

керівник роботи ас. Свірін Павло Володимирович. ,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «10» травня 2017 р. № 1477-С

2. Термін подання студентом роботи ______________________________

3. Вихідні дані до роботи

Об’єктно-орієнтовані бази даних Cache, db4object, Perst, Volante, VelocityDb, реляційні бази даних Microsoft SQL Server, PostgreSQL

4. Зміст роботи ______________________________

Дослідження концепції об’єктно-орієнтованих баз даних, їх технічних аспектів. Аналіз існуючих реалізацій ООБД на продуктивність. Реалізація інформаційної системи, що використовує ООБД у якості сховища даних.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

1. Презентація
2. UML-діаграма класів інформаційної системи – Плакат
3. Наслідування – Плакат
4. B-дерево – Плакат

6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічно-організаційна частина праці	доцент к.е.н. Рощина Н. В		

7. Дата видачі завдання ____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	01.02.2017	
2	Збір інформації	25.02.2017	
3	Дослідження предметної області – теорії ООБД	02.03.2017	
4	Дослідження Caché та VelocityDb як ООБД	10.03.2017	
5	Розробка прикладу застосування ООБД	15.03.2017	
6	Тестування прикладу та аналіз переваг порівняно із іншими підходами	25.04.2017	
7	Аналіз сучасного використання ООБД	30.04.2017	
8	Оформлення дипломної роботи	31.05.2017	
9	Оформлення допуску до захисту та подача роботи в ДЕК	10.06.2017	

Студент ____________ А.О.Токарський
(підпис) (ініціали, прізвище)

Керівник роботи ____________ П.В. Свірін
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено керівника дипломної роботи.

АНОТАЦІЯ

бакалаврської дипломної роботи Токарського Андрія Олеговича
на тему: «Застосування об'єктно-орієнтованих баз даних в інформаційних системах»

Дана дипломна робота присвячена аналізу об'єктно-орієнтованих баз даних та приведення прикладів їх використання в сучасних технологіях.

У роботі розглянуті особливості і деякі технічні аспекти об'єктно-орієнтованих баз даних. Були показані переваги і недоліки використання ООБД в порівнянні з іншими класичними підходами. Також був показаний приклад використання ООБД на прикладі інформаційної системи бібліотеки.

Загальний обсяг роботи: 81 сторінка, 34 рисунки, 10 таблиць, 1 додаток на 20 сторінок, 8 посилань.

Ключові слова: SQL, ООБД, Дані, ООП, База даних, Порівняльна характеристика.

АННОТАЦИЯ

бакалаврской дипломной работы Токарского Андрея Олеговича
на тему:«Применение объектно-ориентированных баз данных в информационных системах»

Данная дипломная работа посвящена анализу объектно-ориентированных баз данных и приведения примеров их использования в современных технологиях.

В работе рассмотрены особенности и некоторые технические аспекты объектно-ориентированных баз данных. Были показаны преимущества и недостатки использования ООБД по сравнению з другими классическими подходами. Также был показан пример использования ООБД на примере информационной системы библиотеки.

Общий объем работы: 81 страница, 34 рисунков, 10 таблиц, 1 приложение на 20 страниц, 8 ссылок.

Ключевые слова: SQL, ООБД, Данные, ООП, База данных, Сравнительная характеристика.

ANNOTATION

a bachelor's degree work of Tokarskyi Andrii

entitled " Use of object-oriented databases in information systems"

This diploma is dedicated to analysis of object-oriented databases and giving examples of using them in modern technologies.

This project considers to overall features and some technical aspects of object-oriented databases. The advantages and disadvantages of OODBMS were compared with the other classic approaches. Also was shown the example of using OODB on the instance of information system of library.

Total volume of work: 81 pages, 34 figures, 10 tables, 1 appendix, 8 references.

Keywords: SQL, OODB, Data, OOP, Database, Comparative characteristics.

ЗМІСТ

ВСТУП 10
1. КОНЦЕПЦІЯ ООБД 12
1.1 Визначення об'єктно-орієнтованої бази даних 12
1.2 Об'єктно-орієнтовані СУБД 15
1.3 Порівняння концепцій ООБД і реляційних БД 16
1.4 Приклад використання ООБД 19
1.5 Висновок 23
2. ТЕХНІЧНІ АСПЕКТИ ООБД 24
2.1 Ідентифікатори об'єктів 24
2.1.1 Зв'язний список 24
2.1.2 Бінарне дерево 24
2.1.3 AVL-дерево 26
2.1.4 B-дерево 26
2.2 Вирішення проблеми наслідування 30
2.3 Висновок 35
3. ІСНУЮЧІ РЕАЛІЗАЦІЇ 36
3.1 Intersystems Caché 36
3.1.1 Використання Caché у космічному напрямку 40
3.1.2 Використання Caché у медицині 42
3.2 База даних VelocityDb 44
3.3 Тестування баз даних на продуктивність 46
3.3.1 Результати тестування реляційних баз даних 48

3.3.2 Результат тестування об’єктно-орієнтованих баз даних 50
3.4 Висновок 51
4. РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ 52
4.1 Опис інформаційної системи 52
4.2 Висновок 57
5. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ 58
5.1 Постановка задачі техніко-економічного аналізу 59
5.1.1 Обґрунтування функцій програмного продукту 60
5.1.2 Варіанти реалізації основних функцій 60
5.2 Обґрунтування системи параметрів ПП 63
5.2.1 Опис параметрів 63
5.2.2 Кількісна оцінка параметрів 64
5.2.3 Аналіз експертного оцінювання параметрів 66
5.3 Аналіз рівня якості варіантів реалізації функцій 70
5.4 Економічний аналіз варіантів розробки ПП 72
5.5 Вибір кращого варіанта ПП техніко-економічного рівня 77
5.6 Висновок 77
ВИСНОВКИ 79
ПЕРЕЛІК ПОСИЛАНЬ 81
ДОДАТОК А 82

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

1. БД – база даних
2. ООБД – об’єктно-орієнтована база даних
3. СУБД – система управління базами даних
4. ПП – програмний продукт

ВСТУП

Більшість інформаційних систем сьогодні зберігають дані в сховищах, що називаються базами даних. Найбільш популярні бази даних - Microsoft SQL Server, Oracle, PostgreSQL або MySQL - використовують реляційну модель представлення даних. У цих базах даних сутності представляються у вигляді рядків таблиць, їх параметри - у вигляді стовпців, а зв'язок між сутностями різних типів здійснюється за допомогою відносин один-до-одного або один-до-багатьох.

Типи даних, що можуть мати стовпці, обмежуються базовими типами (integer, double, varchar, text). Системи керування базами даних, як правило, не дозволяють розширити систему типів шляхом додавання власного типу даних.

Реляційні БД мають власний математичний апарат. Для реляційних БД свого часу Едгар Кодд заклав фундамент математичного апарату реляційної алгебри. Цей математичний апарат пояснює, як повинні виконуватися основні операції над відносинами в базі даних.

У 1986 році був прийнятий перший стандарт SQL-86, який визначив всю долю реляційних БД. Після прийняття стандарту всі розробники реляційних СУБД зобов'язані були слідувати йому. Так, реляційні бази даних, зазвичай, використовують мову SQL як мови запитів.

Реляційні бази даних дуже успішні на ринку, і написано дуже багато програм, які в якості сховища даних використовують саме ці бази даних. До багатьох з цих продуктів вкладені великі гроші, і замовники будуть далі вкладати гроші в їх розвиток.

Але традиційні бази даних не завжди ідеально підходять для зберігання даних. Так, часом буває складно використовувати реляційні БД в інформаційних системах зі складною об'єктно-орієнтованою архітектурою. Програма може мати, наприклад, типи даних для великих і неструктурованих

об'єктів мультимедіа, географічних інформаційних систем або Computer Aided Design / Computer Aided Manufacturing. Складність полягає в тому, що, по-перше, потрібно розробляти не тільки архітектуру класів додатку, але і також схему бази даних. По-друге, на розробку функціоналу, що зможе представити табличні дані у вигляді об'єктів класів і навпаки, теж потрібно витратити час.

1. КОНЦЕПЦІЯ ООБД

1.1 Визначення об’єктно-орієнтованої бази даних

На початку 1980-х років почалися перші дослідження і розробка концепцій об'єктно-орієнтованих баз даних.

Об'єктно-орієнтовані бази даних - бази даних, в яких інформація представлена у вигляді об'єктів, як в об'єктно-орієнтованих мовах програмування.

Причиною появи систем об'єктно-орієнтованих баз даних була потреба в більш адекватному уявленні і моделюванні сутностей реального світу, оскільки ООБД забезпечують набагато більш розвинену модель даних, ніж традиційні реляційні бази даних. Парадигма ООБД грунтується на ряді базових понять, таких як об'єкт, ідентифікатор об'єкта, клас, наслідування, перевантаження і відкладене зв'язування. Будь-який об'єкт при своєму створенні отримує генерований системою унікальний ідентифікатор, який пов'язаний з об'єктом у весь час його існування і не змінюється при зміні стану об'єкта. Кожен об'єкт має стан і поведінку.

1. Стан об'єкта - набір значень його атрибутів.
2. Значення атрибута об'єкта - це теж певний об'єкт або безліч об'єктів.
3. Поведінка об'єкта - набір методів (програмний код), що оперують над станом об'єкта.

Об'єктно-орієнтовані бази даних зазвичай рекомендовані для тих випадків, коли потрібна високопродуктивна обробка даних, що мають складну структуру.

Основні труднощі об'єктно-орієнтованого моделювання даних є наслідком того, що такого розвиненого математичного апарату, на який могла б спиратися загальна об'єктно-орієнтована модель даних, не існує.

Розгляд особливостей ООБД привели до визначення ООБД, яке було представлено на Першій міжнародній конференції з дедуктивним і об'єктно-орієнтованим баз даних у вигляді маніфесту 1989 року. У ньому представлені можливості об'єктно-орієнтованих баз даних.

Обов'язкові можливості ООБД:

1. Підтримка складних об'єктів

Механізм складних об'єктів дозволяє об'єкту мати атрибут, що можуть теж бути об'єктами. Атрибутами можуть бути масиви, списки, хеш-таблиці тощо.

2. Ідентифікація об'єктів

Кожна сутність в базі даних має унікальний ідентифікатор OID. Він є властивістю об'єкта, що відрізняє його від інших об'єктів і існує протягом усього життєвого циклу об'єкта. Ідентифікатор об'єкта повинен бути незалежним від значень його атрибутів

3. Інкапсуляція

Об'єктно-орієнтована модель представлення даних нав'язує інкапсуляцію і приховування інформації. Це означає, що стан об'єкта може ховатися для заборони доступу до внутрішньої логіки роботи об'єкта.

4. Підтримка структур і класів

В ООП структура об'єднує загальні характеристики набору сутностей. Наприклад, структура вектора об'єднує координати X, Y і Z.

Концепція класу схожа на структуру, але вона більше пов'язана з виділенням пам'яті під об'єкти класів під час виконання програми.

5. Наслідування і поліморфізм

Наслідування - одне з базових понять в об'єктно-орієнтованому програмуванні. Пов'язано з тим, що клас може мати класа-спадкоємця, який має ті ж властивості, що і базовий клас, але також і мати нові властивості.

У класі-спадкоємці може бути перевизначений якийсь метод базового класу. Це властивість називається поліморфізмом.

6. Обчислювальна повнота

SQL не має всіх можливостей звичайних мов програмування. Мови типу Pascal, C, C# або Java можна назвати обчислювально повними, так як вони використовують всі обчислювальні можливості комп'ютера. SQL можна назвати реляційно повним, так як в ньому реалізована вся логіка реляційної алгебри. Так, будь-який код, написаний на SQL, можна переписати на C++, але далеко не будь-який код на C++ можна переписати на SQL.

Тому більшість програм, що використовують реляційні бази даних, включають використання вбудованих SQL-запитів в рамках звичайної мови програмування.

Це означає, що мова маніпулювання даними повинна бути мовою програмування загального призначення.

7. Паралелізм

8. Відновлення

9. Спеціальна система запитів

Реляційні бази даних використовують SQL як мови запитів.

Засіб для відправки запитів на базу даних повинно бути і в об'єктно-орієнтованих баз даних. База даних повинна забезпечувати високорівневе,

незалежне від додатка засіб для відправлення запитів. Це не обов'язково може бути мова запитів типу SQL.

10. Можливість додавання нових типів даних

Набір типів даних повинен бути розширюваним. Користувач повинен мати засоби створення нових типів даних на основі набору визначених системних типів. Більш того, між способами використання системних і призначених для користувача типів даних не повинно бути ніяких відмінностей.

1.2 Об'єктно-орієнтовані СУБД

В принципі, ООСУБД - це об'єктно-орієнтована база даних, що надає можливості СУБД об'єктам, що були створені з використанням об'єктно-орієнтованої мови програмування.

Прикладні програми, що використовують об'єктно-орієнтовану базу даних, пишуться на об'єктно-орієнтованої мови програмування, а можливості СУБД існують завдяки спеціальному API, в якому є клас бази даних або базовий клас Persistent, в яких реалізовані функції для роботи з даними.

ООСУБД дозволяє працювати з об'єктами баз даних так само, як з об'єктами в програмуванні на об'єктно-орієнтованих мовах. ООСУБД розширює мови програмування, прозоро вводячи довгострокові дані, управління паралелізмом, відновлення даних, асоціативні запити й інші можливості. Деякі об'єктно-орієнтовані бази даних розроблені для щільної взаємодії з такими об'єктно-орієнтованими мовами програмування як Python, Java, C #, Visual Basic .NET, C ++, Objective-C і Smalltalk; інші мають свої власні мови програмування. ООСУБД використовують точно таку ж модель, що і об'єктно-орієнтовані мови програмування.

Переваги використання ООСУБД:

1. Відсутня проблема невідповідності моделі даних в додатку і БД (impedance mismatch). Всі дані зберігаються в БД в тому ж вигляді, що і в моделі програми.
2. Не потрібно окремо підтримувати модель даних на стороні СУБД.
3. Всі об'єкти на рівні джерела даних строго типізовані. Більше ніяких строкових імен колонок!

1.3 Порівняння концепцій ООБД і реляційних БД

Таблиця 3.1 – Співвідношення між базовими поняттями ООБД і РБД

ООБД	Реляційні БД
Об'єкт	Рядок таблиці
Поле	Стовпчик таблиці
Ієрархія класів	Схема бази даних
Наслідування	Один з окремих випадків відносини один-до-одного, де основний ключ таблиці-потомка є одночасно зовнішнім ключем на основний ключ таблиці-предка
Метод	-
Поліморфізм	-
Інкапсуляція	-

У таблиці 3.1 вказані співвідношення між базовими поняттями, що використовуються у об'єктно-орієнтованих та реляційних базах даних.

Переваги використання об'єктно-орієнтованих баз даних перед реляційними БД:

1. Об'єктно-орієнтовані бази даних дозволяють представляти складні об'єкти більш безпосереднім чином, ніж реляційні системи.
2. Визначення власних абстракцій. Об'єктно-орієнтовані бази даних надають можливість визначати нові абстракції і управляти реалізацією таких абстракцій. Сучасні пакети ООБД дають користувачеві можливість створення нового класу з атрибутами і методами, мати класи, успадковати атрибути та методи від суперкласів, створювати екземпляри класу, кожен з яких володіє унікальним об'єктним ідентифікатором, витягувати ці екземпляри по одному або групами, а також завантажувати і виконувати методи, визначати об'єкти як сукупності інших об'єктів, визначати властивості які теж можуть мати складну структуру і визначатися за допомогою конструктора колекцій.
3. Полегшене проектування деяких зв'язків. В об'єктно-орієнтованих базах даних підтримується засіб інверсних зв'язків для вираження взаємних посилань між двома об'єктами (бінарний зв'язок). Така система забезпечує довідкову цілісність шляхом встановлення відповідного зворотного посилання відразу ж після створення прямого посилання.
4. Відсутність потреби в обумовлених користувачами ключах. У моделі ООБД є поняття ідентифікаторів об'єктів, вони автоматично генеруються системою і гарантовано є унікальними для кожного об'єкта. Ця обставина наряду з тим, що в моделі ООБД усувається потреба в обумовлених користувачами ключах, дає об'єктно-орієнтованим базам даних інші переваги. По-перше, ідентифікатор об'єкта не може бути модифікований додатком. По-друге, поняття ідентифікований об'єкт тягне окреме і узгоджене поняття ідентичності, незалежне від того, яким чином проводиться доступ до об'єкта або як об'єкт моделюється за допомогою описових даних.
5. Наявність предикатів порівняння. У РБД порівняння завжди базується лише на значеннях. У цій моделі два записи є однією і тою ж сутністю,

якщо всі їх ключові атрибути мають однакові значення. Однак в моделі ООБД були розроблені і визначені інші типи порівняння.

6. Менша потреба в з'єднаннях. Реляционні з'єднання - це механізм, що зіставляє відносини між таблицями на основі значень відповідних пар атрибутів в цих відносинах. В ООБД потреба в таких поєднаннях є значно меншою. Але оскільки в ООБД два класи можуть мати відповідні пари атрибутів, в цій моделі все ще може зберігатися можливість проведення реляционного з'єднання.
7. Виграш в продуктивності. У більшості ООБД при завантаженні об'єкта в пам'ять збережені в цьому об'єкті ідентифікатори перетворюються у вказівники в пам'яті.
8. Об'єктна алгебра. Об'єктна алгебра не настільки детально розроблена і не є настільки ж розвинутою, як реляційна алгебра. Але як би там не було, така алгебра існує.

Недоліки використання об'єктно-орієнтованих баз даних перед реляційними БД:

1. Недостатність можливостей для оптимізації запитів. Однією з найбільш значних проблем в ООБД є оптимізація декларативних запитів. Оптимізацію запитів до ООБД ускладнює додаткова складність самої об'єктно-орієнтованої моделі даних.
2. Відсутність стандартної алгебри запитів. Ця обставина теж ускладнює оптимізацію запитів.
3. Проблеми з безпекою. У РБД підтримується авторизація, тоді як в більшості ООБД вона відсутня. РБД надають користувачам передавати і вилучати права на читання або зміни. ООБД зможуть отримати більш широке поширення в області бізнесу лише в тому випадку, якщо ця функція в них буде вдосконалена.
4. Обмежені можливості настройки продуктивності. У більшості ООБД є лише обмежені можливості налаштування продуктивності. У РБД

інсталяторам надається можливість налаштовувати продуктивність системи шляхом завдання великої кількості параметрів, що встановлюються системним адміністратором.

5. Недостатня підтримка складних об'єктів. Повна функціональність складних об'єктів все ще не підтримується. Можна здійснювати навігацію по посиланнях і кодувати операції із застосуванням цих посилань, але відсутні визначені родові операції, в яких використовуються різні види семантики посилань.
6. Обмежена інтеграція з об'єктно-орієнтованими системами програмування. Конфлікти по іменах; необхідність переробляти ієрархії класів; схильність ООБД до перевантаження системних операцій.

1.4 Приклад використання ООБД

Припустимо, перед нами стоїть завдання розробки інформаційної системи для деякого підприємства. Підприємство працює з клієнтами, тому база даних нашої інформаційної системи зберігає дані про клієнтів. Також з клієнтами працюють співробітники, кожен співробітник даної інформаційної системи може робити маніпуляції з інформацією про клієнтів від свого імені. Тому в програмі створена наступна ієрархія класів. У базовому класі Person зберігається загальна інформація про персону: ПІБ, список адрес електронної пошти. Його потомки - класи Client та Employee. У першому зберігається якась клієнтська інформація, у другому - дані для входу. На рисунку 1.1 зображена архітектура класів цієї інформаційної системи.

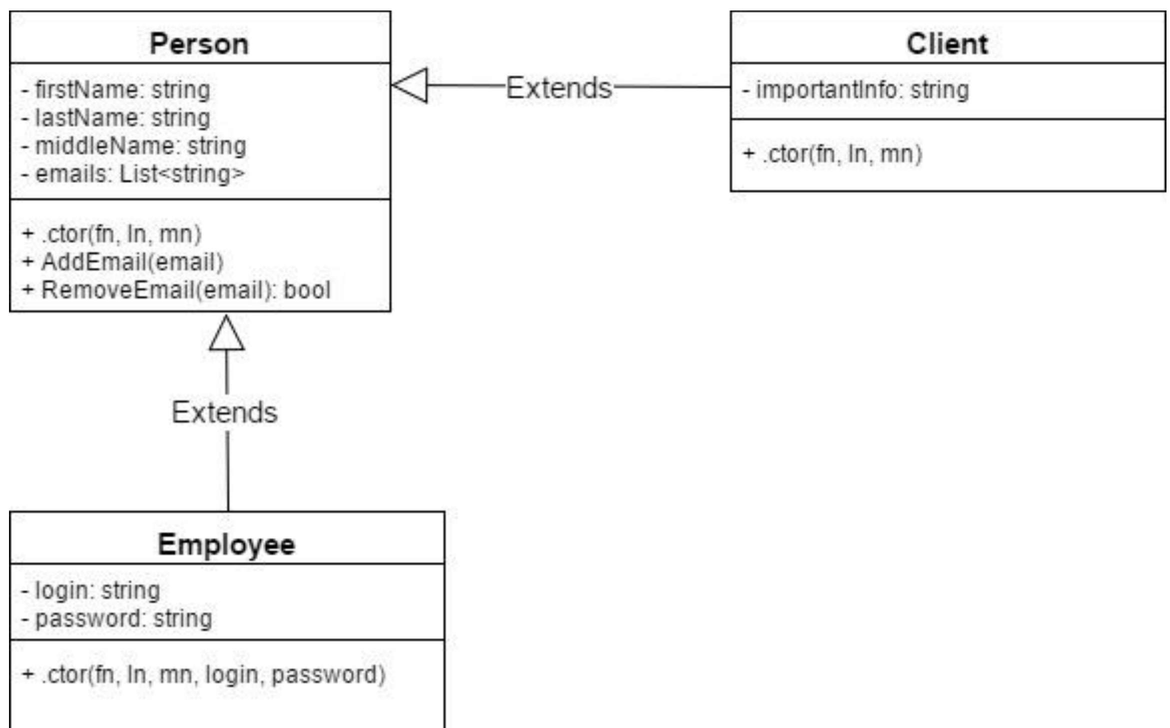

Рисунок 1.1 – UML-діаграма класів інформаційної системи

У випадку з використанням реляційних баз даних в нашій інформаційній системі схема бази даних виглядала б так, як на рисунку 1.2. Наслідування відповідає відносинам один-до-одного між таблицями. Також виникла необхідність додати ще одну таблицю Email з складовим основним ключем, першою частиною якого є зовнішній ключ на таблицю Person, а другий – сама електронна адреса.

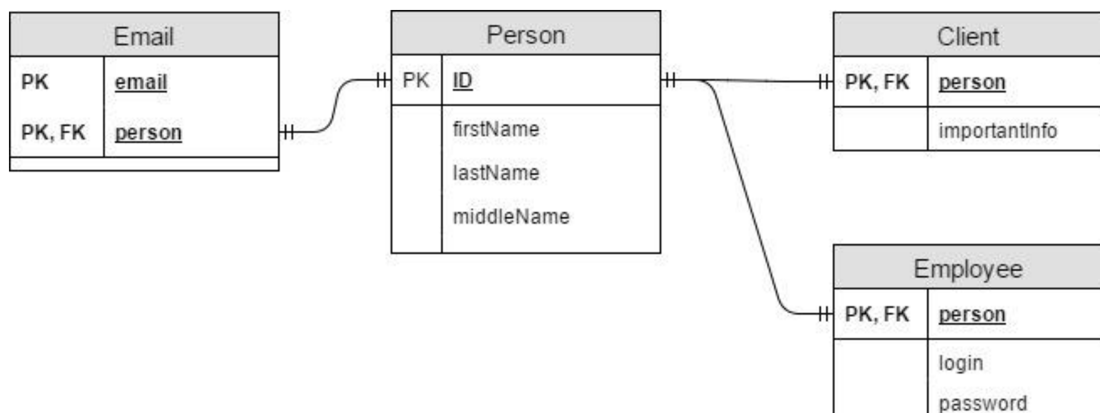

Рисунок 1.2 – Реляційне представлення даних інформаційної системи

Тепер порівняємо, яким чином відбувається відправка запитів зі сторони користувача.

Для вивантаження інформації про клієнта потрібно пов'язати таблиці Client і Person із допомогою ключового слова JOIN, і з допомогою конструкції WHERE визначити умови, які повинні виконуватись для клієнта, інформацію про якого ми шукаємо. У нашому випадку ми шукаємо клієнта по його ідентифікатору.

```
1. SELECT Person.firstName,
2.        Person.lastName,
3.        Person.middleName,
4.        Client.importantInfo
5.   FROM Person
6.        JOIN Client
7.          ON Client.person = Person.id
8.  WHERE Person.id = 25;
```

Рисунок 1.3 – Вивантаження даних з допомогою SQL-запиту

Додавання нових даних здійснюється із допомогою INSERT-запитів. Для того, щоб додати нового клієнта, необхідно спочатку додати нову запис в таблицю Person. Потім в таблицю Client додаємо нову запис, ідентифікатором якої є ідентифікатор щойно доданої записи в таблиці Person. Виконання цих запитів повинне бути нерозривним, тому вони обернуті в обну транзакцію.

Недолік даного підходу у збереженні інформації полягає у тому, що дані в таблицях Person і Client є не нерозривними. Можливо, наприклад, видалити запис із таблиці Client і не видалити запис із цим ідентифікатором із таблиці Person.

```
1. BEGIN TRANSACTION;
2. INSERT INTO Person (firstName, lastName, middleName)
3.      VALUES ('Andrew', 'Tokarskiy', NULL);
4. INSERT INTO Client (person, importantInfo)
5.      VALUES ((SELECT MAX(Person.id) FROM Person), '!!Some info!!');
6. COMMIT;
```

Рисунок 1.4 – Додавання даних з допомогою SQL-запитів

У випадку із необхідністю видалення запису із таблиці Person потрібно видалити або змінити всі записи із інших таблиць, що на ідентифікатор цієї

людини посилаються, на рисунку 1.5 продемонстрована операція видалення клієнта. Всі операції видалення повинні бути обернуті в транзакцію.

```
1. BEGIN TRANSACTION;
2. DECLARE @id INT;
3. SET @id = 25;
4. DELETE FROM Email WHERE person = @id;
5. DELETE FROM Client WHERE person = @id;
6. DELETE FROM Person WHERE id = @id;
7. COMMIT;
```

Рисунок 1.5 – Видалення даних з допомогою SQL-запитів

Як бачимо, зберігання таких даних у реляційних базах даних є достатньо складним для розробника. Розглянемо, яким чином ці ж дані зберігаються у об’єктних базах даних.

Операція вивантаження клієнтів в ООБД відрізняється від операції вивантаження клієнтів в реляційних БД тим, що немає необхідності використовувати JOIN для зв’язування сутностей. Той факт, що Client являється потомком Person, базі даних вже відомий. На рисунку 1.6 продемонстроване вивантаження даних із ООБД.

```
1. Client GetClientById(Database db, int id)
2. {
3.     return db.AsQueryable<Client>()
4.               .First(person => person.Id == id);
5. }
```

Рисунок 1.6 – Вивантаження даних з допомогою ООБД

Це ж стосується і додавання нової інформації, як видно із рисунка 1.7. Також немає необхідності створювати в базі даних нову сутність для зберігання у ній електронних адресів. Вони вже зберігаються у полі emails, що має тип даних List<string>.

```
1. var client = new Client
2. {
3.     FirstName = "John",
4.     LastName = "Smith",
5.     MiddleName = null,
6.     ImportantInfo = "!!Some info!!"
7. };
8. client.AddEmail("smith@email.com");
9.
   db.Persist(client);
```

Рисунок 1.7 – Додавання даних з допомогою ООБД

Видалення об’єкта із об’єктно-орієнтованої бази даних є дуже простим. Достатньо лише вказати об’єкт, що потрібно видалити.

1.5 Висновок

Як бачимо, ООБД мають дуже цікаву концепцію. Маніпулювання із даними у вигляді об’єктів дозволяє зберігати дані із складною структурою і не затрачати зусилля на вирішення такої проблеми, як impedance mismatch. Це дозволяє отримути достатньо великий виграш у продуктивності інформаційної системи. Приклади, які розглядались, демонструють простоту виконання запитів, порівняно із реляційними базами даних. Проте варто відмітити, що ООБД ще не є ідеальними, а на ринку мають серйозних конкурентів, у які вкладаються великі гроші.

2. ТЕХНІЧНІ АСПЕКТИ ООБД

2.1 Ідентифікатори об’єктів

Потужність використання бази даних для управління інформацією безпосередньо залежить від використання індексів, які дозволяють виконувати ефективний пошук, щоб отримувати тільки необхідну інформацію, коли це дійсно необхідно. Існує кілька різних структур індексів, кожен з яких безпосередньо впливає на ефективність пошуку.

2.1.1 Зв’язний список

Зв'язний список - базова динамічна структура даних, що складається з вузлів, кожен з яких містить як власне дані, так і одну або дві посилання на наступний і / або попередній вузол списку. У чистому вигляді, він складається з вказавника на головний перший елемент і лінійної структури, кожен вузол якої містить ключове значення індексу і вказівник, який вказує на наступний вузол в списку, як це зображено на малюнку 2.1. Ця структура, будучи дуже простою, має дуже погану оцінку ефективності додавання $O(n^2)$, так як кожен знову доданий ключ починає пошук з першого індексу і порівнює себе з кожним послідовним ключем до тих пір, поки не знайдено правильне положення.

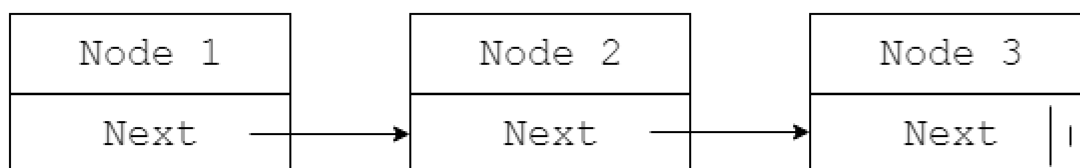

Рисунок 2.1 – Структура зв’язного списку

2.1.2 Бінарне дерево

Крім зв'язаного списку існує більш складне бінарне дерево. Ця структура даних отримала таку назву, так як в вузлах крім ключового значення індексу, є

два вказівника, які вказують на наступний вузол в структурі, в той час як у зв'язаному списку був тільки один.

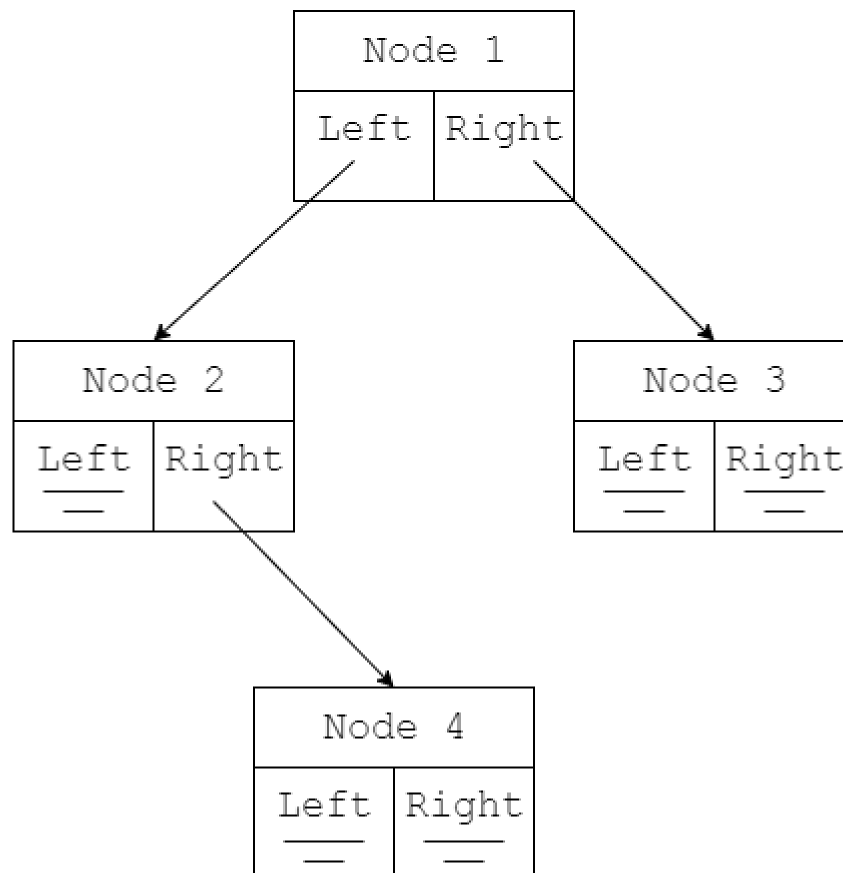

Рисунок 2.2 – Структура бінарного дерева

Ця структура має кращу оцінку складності додавання $O(n \log n)$, так як кожен знову доданий ключ починає пошук з вершини дерева, порівнюючи своє значення з кожним вузлом, і в разі, якщо значення ключа менше поточного вузла, то пошук переходить в праве поддерево, в іншому випадку - в ліве. При цьому виконується в середньому $\log n$ порівнянь, перш ніж знайдеться правильне положення.

Бінарне дерево, однак, має проблеми, коли кожен наступний ключ перевершує попередній, наприклад, в разі послідовного додавання ключів 1,2,3,4 ... По-дерево перетворюється в зв'язаний список зі складністю $O(n^2)$.

2.1.3 AVL-дерево

AVL-дерево - покращення бінарного дерева, запропоноване математиками Адельсоном-Вельським Георгієм Максимовичеи і Ландісом, що, по суті, є збалансованим бінарним деревом.

При додаванні нового ключа структура перебудовується таким чином, щоб зберегти ефективне додавання зі складністю $O(n \log n)$. Це працює навіть в тому випадку, коли кожен раз вставляється ключ, що перевершує попередній.

У чому ж проблема цих стандартних дерев пошуку? Розглянемо величезну базу даних, представлену у вигляді одного зі згаданих дерев. Очевидно, що ми не можемо зберігати все це дерево в оперативній пам'яті, в ній зберігається лише частина інформації, решта ж зберігається на сторонньому носії (наприклад, на жорсткому диску, швидкість доступу до якого є набагато повільнішою). Такі дерева будуть вимагати забагато ($\log n$) звернень до носія. Якраз цю проблема і вирішують B-дерева.

2.1.4 B-дерево

Найпотужнішим з усіх індексів баз даних є індекс на основі B-дерева (Рисунок 2.3). Він може використовувати фіксовану або змінну довжину ключа і управляє ними таким чином, щоб дозволити виконувати кілька типів пошуку, таких як, наприклад, пошук першого, останнього, більше ніж, менше ніж, а також рівного елементів. Він орієнтований на сторінки, що дозволяє підвищити ефективність операцій введення/виводу. Цей індекс може бути використаний для більшості форматів даних: цілі числа, числа з плаваючою комою, дати, текстові рядки і т.д.

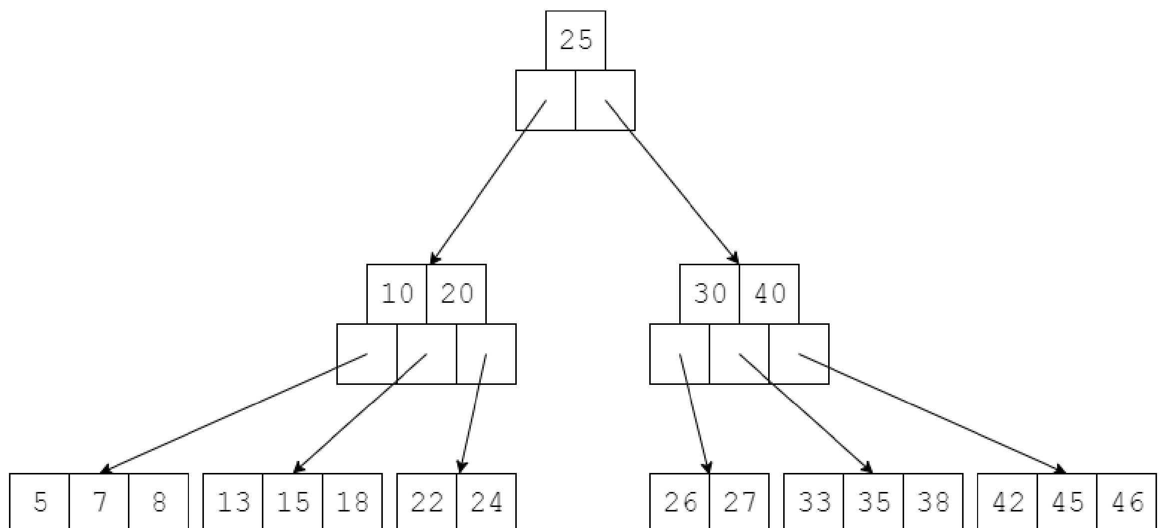

Рисунок 2.3 – Структура B-дерева

B-дерево аналогічно за конструкцією бінарному дереву, але кожен вузол має більше одного значення ключа і двох вказівників, насправді кожна сторінка B-дерева зазвичай містить близько 50 значень ключів, які сортуються від найменшого до найбільшого. У межах кожного ключового значення в межах вузлів індексу є вказівник, який містить посилання на наступну сторінку нижче по вузлу. При пошуку спочатку отримують корінь дерева, переглядають список ключів, поки не виявлять перший ключ, який більше від необхідного ключа, потім переходять до наступного вузла, якщо жоден з ключів не перевищує шуканого ключа, то переходять до останньої сторінки. Після того, як знайдений лист дерева, відповідний ObjectID може бути отриманий і пошук завершується.

Особливості:

1. Вузол зберігає впорядкований набір індексів. Є деяка мінімальна і максимальна можлива кількість ключів у вузлі. Кожна із цих величин може бути виражена із допомогою числа $t \geq 2$, що називається мінімальною степінню B-дерева.
2. B-дерево завжди є повним, тобто відстань від кореня дерева до його листків завжди однакова.

3. Корінь B-дерева повинен мати як мінімум 1 ключ, а інші вузли – як мінімум $t-1$ ключів
4. Кожен вузол містить не більше $2t-1$ ключів. Таким чином, кожен некореневий вузол має не більше, ніж $2t$ вузлів-потомків.

Розглянемо одну із головних властивостей B-дерева, що робить його одним із головних претендентів на роль зберігання індексів у більшості баз даних. Ця властивість показана у наступній теоремі.

Теорема: Висота B-дерева із $n \geq 1$ вузлами і мінімальною степінню $t \geq 2$ не перевищує $\log_t \frac{n+1}{2}$

Доведення: Нехай B-дерево має висоту h. Оскільки перший вузол має як мінімум 1 індекс, то він має як мінімум 2 вузла-потомка. Тоді кожен вузол на глибині 2 має як мінімум t потомків, а, отже, на глибині 3 ми маємо 2t вузлів, на глибині 4 - $2t^2$ і так далі. Отже, кількість ключів n задовольняє нерівності

$$n \geq 1 + (t-1)\sum_{i=1}^{h} 2t^{i-1} = 1 + 2(t-1)\sum_{i=1}^{h} t^{i-1} = 1 + 2(t-1)\left(t^{h-1} + \cdots + 1\right)$$

Для спрощення наступного виразу скористаємось формулою скороченого множення.

$$a^n - b^n = (a-b)(a^{n-1} + a^{n-2}b^1 + a^{n-3}b^2 + \cdots + a^1b^{n-2} + b^{n-1})$$

Отримуємо:

$$n \geq 1 + 2(t-1)\left(t^{h-1} + \cdots + 1\right) = 1 + 2\left(t^h - 1\right) = 2t^h - 1$$

Просте перетворення дає нам нерівність

$$t^h \leq \frac{n+1}{2}$$

Звідси:

$$h = \log_t t^h \leq \log_t \frac{n+1}{2}$$

Теорему доведено.

У чому ми можемо бути впевнені, виходячи із цієї теореми? Розглянемо приклад. Нехай t=2, n=1000000. Тоді $h \leq \log_2 500000.5 \approx 18.93$

Для одного мільйона індексів висота дерева не перевищує 18.93. Це означає, що кількість операцій порівняння для пошуку необхідного індекса навіть не перевищить сотню, оскільки кількість порівнянь в одному вузлі – 3. Ріст B-дерева відбувається не у висоту, а у ширину.

Складність пошуку індекса у B-дереві - $O(th) = O(t \log_t n)$

На відміну від пошуку індексів, операція їх додавання є значно складнішою, ніж в бінарному дереві, так як просто створити новий лист і вставити туди ключ не можна, оскільки це буде порушувати вищевказані властивості B-дерева. Також не можна вставляти ключ в уже заповнений лист. Для цього необхідна операція розбиття вузла на два вузла. Якщо лист був заповнений, то в ньому знаходилося 2t-1 ключів, а, отже, у результаті розбиття вузла отримуємо два вузла, в якому зняходяться по t-1 ключів, а середній елемент (для якого t-1 перших ключі менше його, а останні t-1 останні більше) утворює бітьківський вузол для двох вузлів по t-1 ключів. Відповідно, якщо батьківський вузол також був заповнений - то нам знову доводиться розбивати. І так далі до кореня (якщо розбивається корінь - то з'являється новий корінь і глибина дерева збільшується). Як і у випадку звичайних бінарних дерев, вставка здійснюється за один прохід від кореня до листа. На кожній ітерації (в пошуках позиції для нового ключа - від кореня до листа) ми розбиваємо всі заповнені вузли, через які проходять (в тому число листа). Таким чином, якщо в результаті для вставки потрібно розбити якийсь вузол - ми впевнені в тому, що його батько не заповнений.

2.1.5 Хеш-індекси

Хеш індекси, іноді використовуються в СУБД, діляться на дві групи: динамічне і статичне хешування. Статичне хешування є дуже потужним інструментом, але може застосовуватися в дуже обмеженому ряді випадків, маючи при цьому приголомшливу складність пошуку O(1). Статичне хешування використовується для пошуку по ObjectID. По суті, такий індекс є індексом масиву, де ObjectID прямо вказує на позицію в масиві, тобто адреса, де знаходиться об'єкт. Проте, статичне хеширования дуже обмежене в своєму застосуванні: по-перше, тільки унікальні значення можуть бути використані для побудови індексу і, по-друге, алгоритм хешування повинен бути простим.

Динамічне хешування, з іншого боку, є більш гнучким, але коли створюється загальний алгоритм для будь-яких даних, складність пошуку падає до $O(n \log n)$, що є тим самим значенням, що і для B-дерева, але без можливості шукати перший, останній, наступний, попередній, більше і менше. Саме тому багато СУБД не використовують динамічні індекси хешування.

2.2 Вирішення проблеми наслідування

Невід'ємною частиною об'єктно-орієнтованої архітектури є наслідування. Наслідування (inheritance) – це концепція ООП, згідно із якою тип даних може наслідувати дані і функціонал деякого існуючого типу.

Для об'єктно-орієнтованих баз даних наслідування є «природнім ефектом», оскільки, як було розглянуто раніше, ООБД повторюють всі властивості об'єктно-орієнтованої архітектури. Також існує і множинне наслідування (multiple inheritance) – це вид наслідування, при якому клас-потомок може мати більше одного батьківського класу. Наслідування можливим є не тільки у ООБД, а і у інших базах даних. Так, у базі даних PostgreSQL є можливість для таблиць створювати таблиць-потомків. Розглянемо, яким чином це реалізовано у PostgreSQL.

Для цього створимо схему бази даних, заповнимо її таблиці даними та виконаємо ряд SELECT-запитів, як це зображено на рисунку 2.4.

```
1.
2.      --- Створення таблиць
3.      CREATE TABLE Human (
4.          firstName TEXT,
5.          lastName  TEXT
6.      );
7.
8.      CREATE TABLE Programmer (
9.          progLanguage TEXT
10.     ) INHERITS (Human); --- Таблиця Programmer наслідує Human
11.
12.
13.     – Заповнення даними
14.     INSERT INTO Programmer (firstName, lastName, progLanguage)
15.          VALUES ('Jon', 'Skeet', 'C#');
16.
17.     INSERT INTO Human(firstName, lastName)
18.          VALUES ('John', 'Smith');
19.
20.     INSERT INTO Programmer(firstName, lastName, progLanguage)
21.          VALUES ('Bill', 'Gates', 'Visual Basic');
22.
23.     INSERT INTO Human(firstName, lastName)
24.          VALUES ('Roman', 'Ivanov');
25.
26.     -- Виконання SELECT-запитів
27.
28.     SELECT * FROM Human; ---- #1
29.     SELECT * FROM Programmer; ---- #2
30.     SELECT * FROM ONLY Human; ---- #3
31.     SELECT * FROM ONLY Programmer; ---- #4
32.     SELECT Human.tableoid, ---- #5
33.            Human.*
34.       FROM Human;
35.
36.     ---- #6
37.     SELECT pg_class.oid,
38.            pg_class.relname,
39.            Human.*
40.       FROM Human
41.            JOIN pg_class
42.              ON pg_class.oid = Human.tableoid;
43.
44.     ---- #7
45.     SELECT * FROM pg_inherits;
```

Рисунок 2.4 – Набір запитів, що демонструє роботу наслідування в PostgreSQL

Результат виконання цього ряду запитів зображений на рисунку 2.5.

	firstname	lastname
1	John	Smith
2	Roman	Ivanov
3	Jon	Skeet
4	Bill	Gates

	firstname	lastname	proglanguage
1	Jon	Skeet	C#
2	Bill	Gates	Visual Basic

	firstname	lastname
1	John	Smith
2	Roman	Ivanov

	firstname	lastname	proglanguage
1	Jon	Skeet	C#
2	Bill	Gates	Visual Basic

	tableoid	firstname	lastname
1	1196801	John	Smith
2	1196801	Roman	Ivanov
3	1196807	Jon	Skeet
4	1196807	Bill	Gates

	oid	relname	firstname	lastname
1	1196801	human	John	Smith
2	1196801	human	Roman	Ivanov
3	1196807	programmer	Jon	Skeet
4	1196807	programmer	Bill	Gates

	inhrelid	inhparent	inhseqno
1	1196807	1196801	1

Рисунок 2.5 – Результат виконання тестових запитів

Із результатів, отриманих із виконання першого запиту, видно, що всі записи із таблиці Programmer одночасно знаходяться і в таблицю Human. Як видно із запиту 3, із допомогою ключового слова ONLY можна вибрати ті записи, які є у даній таблиці, але яких немає у жодній із потомків.

Запити 5 і 6 дозволяють визначити кінцеву таблицю, у якій знаходяться записи. Це здійснено із допомогою колонки tableoid, яку мають всі таблиці в PostgreSQL.

Таблиця pg_inherits в СУБД PostgreSQL зберігає в собі список усіх наслідувань. У нашому прикладі (результат виконання запиту №7) видно, що таблиця із ідентифікатором 1196807 – потомок таблиці із ідентифікатором 1196801.

Детальніше роботу таблиці pg_inherits можна розглянути на іншому прикладі. Розглянемо наступний SQL-код, зображений на рисунку 2.6.

```
1.
2.     CREATE TABLE Humanoid (
3.         humanoid_data TEXT
4.     );
5.
6.     CREATE TABLE AI (
7.         ai_data TEXT
8.     );
9.
10.    CREATE TABLE Human () INHERITS (Humanoid);
11.    CREATE TABLE Droid () INHERITS (Humanoid, AI);
12.    CREATE TABLE C3PO() INHERITS (Droid);
13.
14.    SELECT * FROM pg_inherits;
```

	inhrelid	inhparent	inhseqno
1	1196825	1196813	1
2	1196831	1196813	1
3	1196831	1196819	2
4	1196837	1196831	1

Рисунок 2.6 – Демонстрація роботи таблиці pg_inherits в PostgreSQL

У цьому прикладі також продемонстровано multiple inheritance. Є дві базових таблиці – Humanoid та AI. Human наслідується із Humanoid, а Droid – із Humanoid та AI (це приклад множинного наслідування). Також, в свою чергу, Droid є батьківською таблицею для C3PO.

Розглянемо колонки таблиці pg_inherits. inhrelid та inhparent вказують, яка таблиця наслідується із якої відповідно. Якщо у таблиці-потомка є кілька батьківських таблиць, то число inhseqno визначає порядок, у якому будуть розташовуватись стовпці наслідуваних таблиць.

Таким чином, користуючись даними із таблиці pg_inherits, можемо отримати граф, на якому зображено схему наслідування у базі. Цей граф зображений на рисунку 2.7.

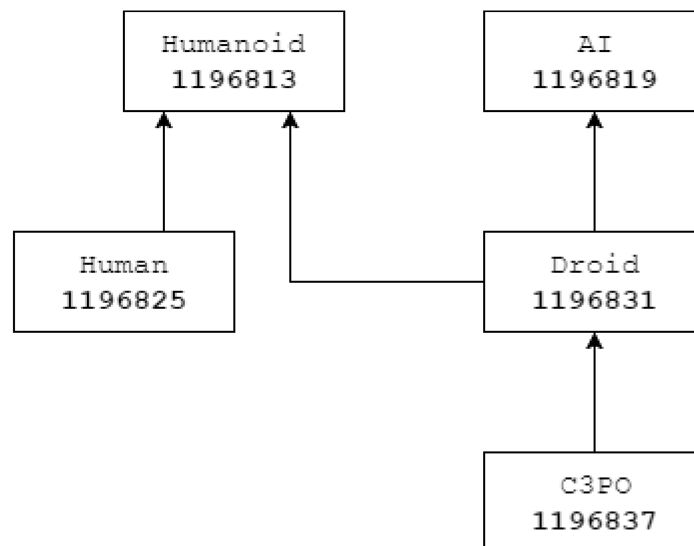

Рисунок 2.7 – Граф наслідування

Отже, можна зробити висновок, що для забезпечення базі даних можливістю наслідування потрібно в тому чи іншому вигляді зберігати граф наслідування. У СУБД PostgreSQL дані про наслідування зберігаються у окремій службовій таблиці pg_inherits.

У випадку, якщо multiple inheritance забороняється, граф наслідування буде деревом або набором дерев. У такому випадку швидкий доступ до інформації про наслідування може забезпечити її зберігання у службових хеш-таблицях, ключ якої – ідентифікатор таблиці/класу, а значення – ідентифікатор батьківської таблиці/класу.

Слід звернути увагу, що не всі команди SQL можуть працювати з ієрархією наслідування. Команди, які використовуються для запитів отримання даних, зміни даних або зміни схем (наприклад, SELECT, UPDATE, DELETE, більшість варіантів ALTER TABLE, але не INSERT і ALTER TABLE ... RENAME) зазвичай за умовчанням включають таблиці-нащадки і підтримують нотацію ONLY для виключення цих таблиць-нащадків. Команди, які обслуговують базу даних і виконують тонкі настройки (наприклад, REINDEX, VACUUM) зазвичай працюють тільки з окремими, фізичними таблицями і не підтримують рекурсивную обробку ієрархії наслідування.

Серйозне обмеження можливості наслідування полягає в тому, що індекси (включаючи обмеження унікальності) і зовнішні ключі застосовуються тільки до одиноких таблиць, а не до нащадків. Це обмеження справедливо як для посилаються так і для посилальних сторін зовнішнього ключа.

Наприклад, якщо ви визначаєте cities.name як UNIQUE або PRIMARY KEY, то ці обмеження не зупинять наявність в таблиці capitals рядків, в яких значення name дублюють значення в рядках таблиці cities. І ці дубльовані рядки будуть за замовчуванням показуватися в запитах до таблиці cities. Фактично, за замовчуванням таблиця capitals не матиме жодних обмежень унікальності і таким чином може містити безліч рядків з одним і тим же значенням name. Ви можете додати обмеження унікальності для таблиці capitals.

Схожим чином, якщо ми задали для cities.name обмеження REFERENCES з посиланням на іншу таблицю, це обмеження не буде автоматично поширюватися на таблицю capitals. У цьому випадку, ви можете вручну додати обмеження REFERENCES для capitals.

2.3 Висновок

У розділі були розглянуті такі технічні аспекти баз даних, як збереження ключів та наслідування. Збереження ключів у B-деревах або їх модифікаціях дозволяє виконувати пошук, додання, видалення даних за дуже малу кількість дискових операцій. B-дерева являються структурою для зберігання ключів у більшості відомих ООБД.

Також була розглянута проблема наслідування. Продемонстрована робота наслідування на прикладі PostgreSQL. Користуватись ним у цій базі дуже незручно, але принципи зберігання інформації про наслідування класів схожі і в ООБД.

3. ІСНУЮЧІ РЕАЛІЗАЦІЇ

3.1 Intersystems Caché

Caché – пропрієтарна об’єктно-орієнтована база даних, розроблена компанією Intersystems. Дані в цій базі даних зберігаються у виді багатомірних масивів. Caché пропонує розробникам три режими доступу до даних: об'єктний, SQL або шляхом прямого доступу до багатомірних структур даних.

Дуже великою проблемою об’єктно-орієнтованих баз даних є значна перевага проектів, що використовують реляційні бази даних, оскільки при переході на об’єктну технологію потрібно буде знову починати все «з нуля». Крім того, об’єктна технологія не має розвинутої і стандартизованої мови запитів. В СУБД Caché не лише реалізовані всі особливості об’єктно-орієнтованої технології, вона ще і може полегшити перехід із реляційної технології завдяки важливій особливості Caché - незалежності зберігання даних від способу їх представлення.

Дані в Caché зберігаються у виді розріджених масивів, що називаються глобалами (рисунок 3.1). Кількість індексів такого масива може бути будь-яким. Крім того, індекси глобалів не є типізованими.

```
1. set ^PersonInfo(1) = "Ivan Ivanov"
2. set ^PersonInfo(1, "Lord of the Rings") = 1
3. set ^PersonInfo(1, "Harry Potter") = 0
4. set ^PersonInfo(2) = "Ivan Ivanenko"
5. set ^PersonInfo(3) = "John Smith"
6. set ^PersonInfo(3, "Witcher") = 1
```

Рисунок 3.1 – Приклад створення глобала на мові Caché Object Script

Розглянемо приклад створення глобала (Рисунок 3.1). Створимо просту базу даних бабліотеки. У базі даних зберігаємо людей і їх ідентифікатори. У статусі людей буде вказано, які книги вони брали і які із них було повернуто.

Наприклад, глобал PersonInfo зберігає у собі інформацію про трьох людей. Перший з них – Ivan Ivanov. Він брав дві книги: «Володар перстнів» і «Гаррі Поттер», причому останню із них він не повернув. Ivan Ivanenko не брав жодних книг, а John Smith прочитав і повернув «Відьмака». Як бачимо, структура глобалів дуже проста – це звичайні багатомірні асоціативні масиви, характер вихідної інформації яких залежить від того, скільки параметрів подано на вхід. В нашому випадку, якщо на вхід поданий лише один параметр (ідентифікатор), то функція поверне ім’я людини. Якщо подані два параметри (ідентифікатор та назва книжки), то функція поверне 1 або 0 в залежності від того, була повернута книжка чи ні. Ім’я глобала і сенс кожного із індексів придумані розробником.

На рівні глобалів не існує звичного для реляційних баз даних декларативної мови запитів. Запити визначаються алгоритмічним способом - виконання запиту зводиться до виконання коду, написаного на мові Caché Object Script, який надає достатній набір простих, ефективних операцій для роботи з даними, збереженими в глобальний. Унікальність Caché Object Script як мови програмування в тому, що це мабуть єдина мова, в синтаксисі якої явно введена конструкція для вказівки того, де зберігається змінна - в пам'яті або грубо кажучи на диску. При цьому вам не треба заздалегідь визначати структури в базі даних - ви просто працюєте з ними так само як зі змінними в мовах з нестрогою типізацією.

Концепція Unified Data Model заснована на принципі «дані одні – моделей представлення даних багато»

Дані в Caché повністю складаються із таких асоціативних структур даних, але організовані вони так, що ці дані легко перетворюються в реляційні та об’єктні.

Слідом за InterSystems, яка вже реалізувала на основі глобалів об'єктно-орієнтований і реляційний (SQL) доступи ви здатні реалізувати свою власну, унікальну модель даних і так само використовувати принцип Unified Data Model - підхід до управління персистентного, який передбачає роботу з одними і тими ж даними в різних моделях залежно від зручності їх використання в контексті конкретного завдання. Наприклад, для швидкої вставки і читання можливе використання key-value моделі, а для запитів використовуються можливості реляційної моделі. При побудові своєї мови запитів для NoSQL-рішення можна використовувати Caché Object Script, який надає набір простих операцій для роботи з даними.

Caché можна порівняти із NoSQL завдяки розподіленості і масштабування. Якщо порівнювати Caché в такій важливій для NoSQL категорії, як забезпечення горизонтального масштабування і розподіленого зберігання з використанням таких механізмів як шардинг і партиціювання, то з одного боку Caché не має готових out of the box варіантів з такими назвами. З іншої точки зору це не зовсім так, тому що в Caché можлива реалізація ефективної розподіленої обробки даних завдяки вищесказаному.

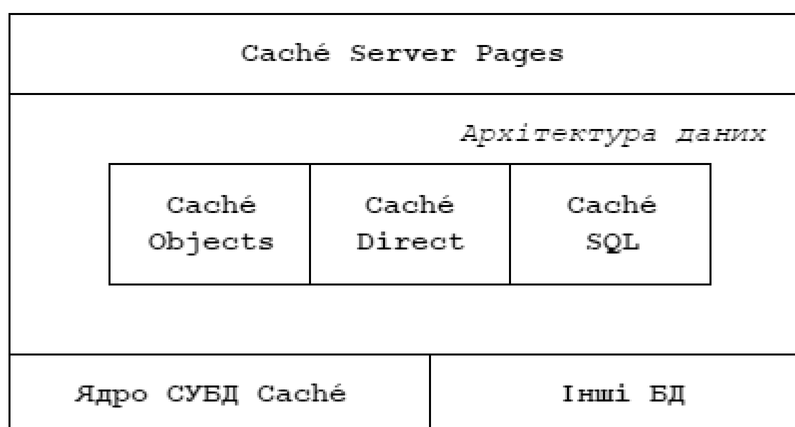

Рисунок 3.2 - Архітектура системи Caché

Як бачимо із рисунка 3.2, архітектура СУБД Caché складається із таких основних частин.

1. Caché SQL – представлення багатомірних структур даних у вигляді реляційних таблиць.
2. Caché Direct – забезпечення прямого доступу до реляційних таблиць.
3. Caché Objects – представлення багатомірних структур даних у вигляді об’єктів, у яких інкапсульовані як дані, так і методи їх обробки.

Відповідно до стандарту в Caché, в об’єктній моделі реалізовано два типи класів:

1. класи типів даних (літерали);
2. класи об'єктів (об'єкти).

Класи типів даних визначають допустимі значення констант (літералів) і дозволяють їх контролювати. Літерал не може існувати незалежно від свого значення, в той час як об'єкти мають унікальну ідентифікацію.

Класи типів даних поділяється на два підкласу типів:

1. атомарні;
2. структуровані.

Атомарними літеральнимі типами в Caché є традиційні скалярні типи даних (%String, %Integer, %Float, %Date і ін.). В Cache 'реалізовані дві структури класів типів даних - список і масив. Кожен літерал унікально ідентифікується індексом в масиві і порядковим номером в списку.

Розрізняють два підтипи класів об'єктів - зареєстровані та незареєстровані. Зареєстровані класи мають визначену поведінку, тобто набір методів, успадкованих з системного класу «%RegisteredObject» і що відповідають за створення нових об'єктів і за управління розміщенням об'єктів

в пам'яті. Незареєстровані класи не мають наперед визначену поведінку, за розробку методів таких класів відповідає розробник.

Зареєстровані класи можуть бути двох типів - вбудовані і збережені. Вбудовувані класи успадковують свою поведінку від системного класу «%SerialObject». Основною особливістю зберігання вбудованого класу є те, що об'єкти вбудованих класів існують в пам'яті як незалежні екземпляри, однак можуть бути збережені в базі даних, тільки будучи вбудованими в інший клас.

Основною перевагою використання вбудованих класів є мінімум витрат, пов'язаних з можливим в майбутньому зміною набору однакових властивостей класів, представлених у вигляді вбудованого об'єкта.

Збережені класи успадковують свою поведінку від системного класу «%Persistent». «%Persistent» надає широкий набір функцій своїм потомкам, що включає: створення об'єкта, підкачування об'єкта з БД в пам'ять, видалення об'єкта і т.п. Кожен екземпляр зберігається класу має 2 унікальних ідентифікатори - OID і OREF. OID (object ID) характеризує об'єкт, записаний в БД, тобто на фізичному носії, а OREF (object reference) характеризує об'єкт, який був підкачаний з БД і знаходиться в пам'яті.

3.1.1 Використання Caché у космічному напрямку

У 2013 році Європейське космічне агентство (European Space Agency, ESA) почало здійснення амбіційного проекту зі створення тривимірної карти Чумацького Шляху. Для забезпечення обробки наукової інформації під час польоту автоматичного космічного апарату GAIA (Global Astrometric Interferometer for Astrophysics), зображеного на рисунку 3.3, ESA вибрало технологію InterSystems Caché, оскільки в цьому проекті необхідно швидко зберігати і аналізувати величезні масиви даних.

Рисунок 3.3 - Макет Gaia на салоні Ле Бурже 2013 року

Передбачається, що політ космічного апарату GAIA буде тривати п'ять років, протягом яких буде проводитися спостереження мільярда зірок в нашій Галактиці і моніторинг їх стану. Для кожної зірки буде виконано близько 100 різних вимірів, визначені положення в просторі, відносні відстані, характеристики руху, світність. Очікується, що в результаті цієї космічної місії будуть виявлені сотні тисяч нових астрономічних об'єктів, зокрема планети біля інших зірок і так звані коричневі карлики. Крім того, в межах Сонячної системи GAIA буде вести спостереження сотень тисяч астероїдів.

Все це тільки збільшує і без того колосальний обсяг даних, які потрібно зібрати і проаналізувати. Ключовим елементом комплексу GAIA є астрометричної рішення Astrometric Global Iterative Solution (AGIS), яке дозволить в ітераційне режимі підвищувати точність всіх просторових вимірів, виконуваних супутником.

В ході роботи система AGIS повинна буде вносити до 50 млрд Java-об'єктів в базу даних за кожні сім днів.

Найважливіша обставина, що вплинула на вибір Caché , це унікальна технологія, що дає можливість вставки Java-об'єктів безпосередньо в багатовимірні структури, використовувані механізмом бази даних Caché.

ЄКА планує, що загальна вартість проекта, до якої входить вартість космічного апарату, засобів виведення і наземного контролю складе приблизно 577 мільйонів євро. Контракт на розробку і побудову самого телескопу Gaia, вартістю 317 мільйонів євро, отримала європейська компанія EADS Astrium. Вартість наукової обробки даних (буде розділена між країнами-учасницями ЄКА) оцінюється в 120 мільйонів євро.

3.1.2 Використання Caché у медицині

Caché також відома як база даних, що часто використовується у медичних цілях.

Міська клінічна лікарня № 12 (МКЛ №12) Департаменту охорони здоров'я м Москви була створена розпорядженням Уряду Москви після придбання у власність міста Москви лікарняного комплексу.

МКЛ №12 - багатопрофільний лікувально-профілактичний заклад на 1180 ліжок, що надає цілодобову стаціонарну, консультативну та амбулаторну медичну допомогу. У лікарні - 25 клінічних і 15 параклінічних відділень.

У клініці працюють понад півтори тисячі осіб. За рік близько 40 тис. пацієнтів отримують висококваліфіковану медичну допомогу стаціонарно або амбулаторно. На базі лікарні працюють кафедри Російського університету Дружби народів і Російського Державного медичного інституту.

В кінці 90-х рр. адміністрація медсанчастини прийняла рішення про створення інтегрованої системи, в якій всі дані про пацієнта - і реєстраційні, і фінансові, і медичні - перебували б в одному місці, а введення цих даних проводилося б самими співробітниками в процесі виконання ними своїх посадових обов'язків. Крім того, система повинна була функціонувати в

режимі: 24 години на добу, 7 днів на тиждень, 365 днів на рік (з урахуванням проведення регламентного технічного обслуговування). Роботи по її впровадженню необхідно було проводити таким чином, щоб не переривалася основна діяльність медустанови.

У 2002 р фахівці компанії «Е-Куб» представили макет інтегрованої системи МІС «Е-Куб», в якій дані з усіх програм (включаючи автономні) об'єднувалися в електронній історії хвороби пацієнта. Остаточно переконало керівництво в реальності створення інтегрованої системи та обставина, що МІС «Е-Куб» була розроблена на платформі MSM-Caché і, отже, не було необхідності повної заміни комп'ютерного обладнання та перенавчання співробітників медичної установи.

Впровадження МІС« Е-Куб »в приймальному відділенні дозволило заздалегідь отримувати інформацію про те, скільки пацієнтів і з яким діагнозом везуть в лікарню, і акумулювати всю інформацію про наявність вільних місць у відділеннях. У підсумку, виграш у часі при госпіталізації тяжкохворих може становити від 20 до 40 хвилин. Також система дозволяє відслідковувати в режимі реального часу хід обстеження пацієнтів, використовуючи різні колірні індикатори. Якщо пацієнт знаходиться в приймальному відділенні довше максимально допустимого інтервалу часу, то інформація про нього висвітиться червоним кольором - це знак для керівництва відділенням, що необхідно проконтролювати ситуацію.

Особливості системи МІС «Е-Куб»:

1. три варіанти інтерфейсу користувача (текст, графіка, веб), що полегшують процес впровадження;
2. зручний ієрархічний гіпертекстовий (drill down / up) підхід до представлення даних;
3. багаторівнева система розмежування доступу;
4. низькі вимоги до обладнання;

5. розвинена система оповіщення (e-mail, SMS, внутрішня пошта);
6. використання для клієнтської частини ОС Windows і Linux.

Вибір ООБД можна було пояснити складністю самої природи інформації, яка зберігається у медичних інформаційних системах. Так, у пацієнтів, наприклад, повинна зберігатись історія хвороб. Але кожна із хвороб є специфічною, і тому кожна із них має свій набір параметрів, що її характеризують. Тому для опису цієї інформації необхідний складний механізм наслідування.

3.2 База даних VelocityDb

VelocityDb – об’єктно-орієнтована база даних для роботи із платформою .NET. Функціонал для відправки запитів для читання чи обробки інформації реалізовані у виді методів абстрактного класу SessionBase, наслідники якого і визначають режим підключення до бази даних.

Класи, що зберігаються, повинні бути потомками класу OptimizedPersistable, що містить у собі ідентифікатор об’єкта і методи для додавання об’єкта у базу та для його видалення. Класи, що не є потомками OptimizedPersistable, зберігаються у базі лише той час, поки є на них хоча б одне посилання від об’єктів OptimizedPersistable. Також у бібліотеці реалізовані класи VelocityDbList<T> та VelocityDbDictionary<T>, що являються аналогами List<T> та Dictionary<T>, але водночас є і потомками OptimizedPersistable.

Для відправки запитів зручно користуватись LINQ, як це зображено на рисунку 3.4. LINQ – вбудований у мови програмування платформи .NET Framework (C#, VB.NET, F# тощо) синтаксис мови запитів, що нагадує SQL.

```
1.  private void SendQuery()
2.  {
3.      using (var conn = new SessionNoServer("FolderName"))
4.      {
5.          conn.BeginRead();
6.          var query =
7.              from person in conn.AllObjects<Person>()
8.              where person.BirthDate != null
9.              let day = person.BirthDate.Value.Day
10.             let month = person.BirthDate.Value.Month
11.             where day == DateTime.Today.Day
12.                && month == DateTime.Today.Month
13.             orderby person.FirstName, person.LastName
14.             group person by person.GetType();
15.
16.         foreach (var personGroup in query)
17.         {
18.             Console.WriteLine(personGroup.Key == typeof(Employee)
19.                 ? "Employees" : "Clients");
20.             foreach (var person in personGroup)
21.             {
22.                 Console.WriteLine("\t{0}", person);
23.             }
24.         }
25.
26.         conn.Commit();
27.     }
28. }
```

Рисунок 3.4 – Приклад запиту до бази даних VelocityDb із використанням LINQ

Розглянемо приклад запиту. Нехай у базі даних компанії люди можуть буди як співробітниками, так і клієнтами. Є база даних людей. Потрібно вивести всіх людей у яких сьогодні день народження, але згрупувати клієнтів і співробітників окремо. На рисунку 3.4 зображене вирішення даної задачі.

Функції BeginRead() та Commit() визначають межі транзакції. Функція AllObjects<T>() вилучає об’єкти типу T. СУБД VelocityDb має два види транзакцій – Read і Update, які об’являються викликом методів BeginRead() і BeginUpdate() відповідно і завершені викликом методу Commit().

Read-транзакції дозволяють лише читати інформацію і не допускають внесення ніяких змін у дані. Це зроблено, щоб кілька Read-транзакцій могли безболісно використовуватись одночасно, що значно прискорює вилучення даних при одночасній роботі багатьох клієнтів.

3.3 Тестування баз даних на продуктивність

Порівняємо продуктивність реляційних та об’єктно-орієнтованих баз даних на практиці. Для цього ми розглянемо швидкість та об’єм пам’яті, що використовується при додаванні, зчитуванні та повторному зчитуванні великої кількості даних. Також критерієм для порівняння буде розмір бази даних, заповненої даними.

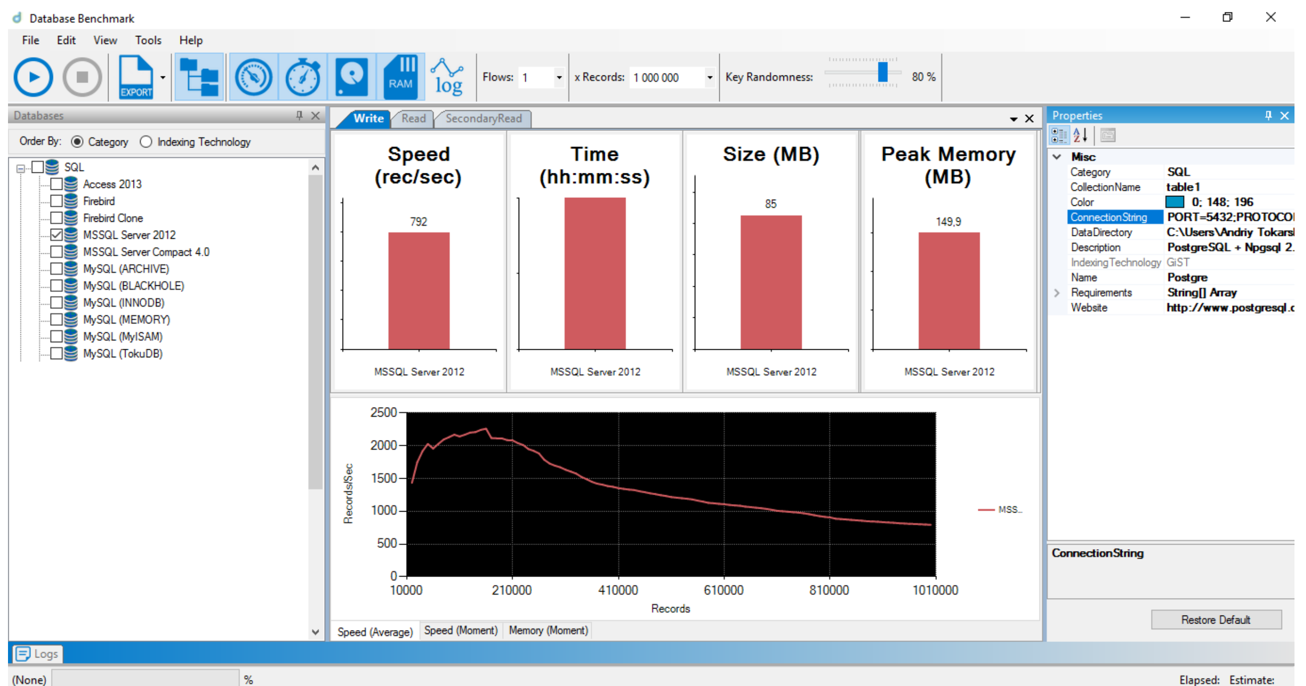

Рисунок 3.5: Інтерфейс Database Benchmark

Для порівняння будемо використовувати програму Database Benchmark v3.0.0. Ця програма може вимірювати продуктивність дуже великої кількості баз даних, серед яких є як реляційні, так і об’єктні. Database Benchmark дозволяє спостерігати за процесом тестування у реальному часі. Розроблена із використанням .NET Framework + WPF.

Порівнювати будемо такі бази даних, як Microsoft SQL Server 2016, Db4Objects, VelocityDb, Volante, Perst. Всі ці бази даних можливо тестувати із використанням Database Benchmark.

Тестувати будемо швидкість і затрати пам'яті на додавання та читання одного мільйона записів, а також розмір відповідних баз даних.

Таблиця 3.1 - Швидкість операцій, виконуваних базами даних (у записах/сек)

	Додавання	Читання	Повторне читання
Db4Objects	9898	4839	4868
Perst	1894	1175	1194
VelocityDb	145603	30452	426333
Volante	15447	136968	119346
MS SQL Server 2016	792	412031	435002

Таблиця 3.2: Максимальні затрати оперативної пам'яті (у Мб)

	Додавання	Читання	Повторне читання
Db4Objects	1134.1	1134.2	738.3
Perst	130	122.3	112.1
VelocityDb	256.2	272.5	272.6
Volante	2423.5	2505.7	2498
MS SQL Server 2016	149.9	130.5	130.5

Таблиця 3.3: Розмір заповнених баз даних (у Мб)

Db4Objects	Perst	VelocityDb	Volante	MS SQL Server 2016
229.9	188.6	53.1	123	85

У таблиці 3.2-3.4 занесені результати вимірювання продуктивності баз даних. Таблиця 3.2 показує швидкість виконання операцій, 3.3 показує затрати пам'яті на виконання цих операцій, а 3.3 – розміри баз даних.

3.3.1 Результати тестування реляційних баз даних

Для порівняння із об'єктно-орієнтованими базами даних мною було протестовано реляційну базу даних Microsoft SQL Server. Database Benchmark згенерував таблицю і заповнив її такими даними: число типу BIGINT, що виступає основним ключом таблиці, дві колонки типу VARCHAR(255), дві колонки типу INT, дві колонки типу REAL, і колонка типу DATETIME. Таблицю було заповнено одним мільйоном записів, що видно із рисунка 3.6.

```
/****** Script for SelectTopNRows command from SSMS  ******/
SELECT COUNT(ID) FROM dbo.table1;

SELECT TOP (1000) [ID]
      ,[Symbol]
      ,[Time]
      ,[Bid]
      ,[Ask]
      ,[BidSize]
      ,[AskSize]
      ,[Provider]
  FROM [TestDatabase].[dbo].[table1]
```

	(No column name)
1	1000000

	ID	Symbol	Time	Bid	Ask	BidSize	AskSize	Provider
1	45789490150975	XAGUSD	2019-05-03 22:56:04.597	5,99	6,21	417	8464	SSE
2	57691047792169	USDJPY	2019-10-12 06:21:02.597	55,75	55,81	2569	6752	Gain
3	59560107520268	USDJPY	2019-06-04 00:33:44.597	68,78	68,96	6253	5701	SSE
4	83242539069798	$DAX	2019-05-02 04:25:59.597	9091,13	9091,26	7043	8076	NSEI
5	84681375865187	$DAX	2019-09-11 03:40:01.597	9092,82	9093,04	6903	9450	NYSE
6	92189023103234	AUDBGN	2019-08-11 14:12:19.597	1,4589	1,4604	3398	7732	NSEI
7	109415657494...	GBPCHF	2019-09-12 04:18:01.597	1,538	1,5409	2578	2614	SSE
8	109415657494...	GBPCAD	2019-09-12 04:18:08.597	1,9373	1,9399	679	3113	TSE

Рисунок 3.6 - Тестова таблиця SQL Server, заповнена даними

Із результатів виконання програми Database Benchmark отримуємо, що Microsoft SQL Server є найповільнішою серед протестованих баз даних по швидкості додання нових записів.

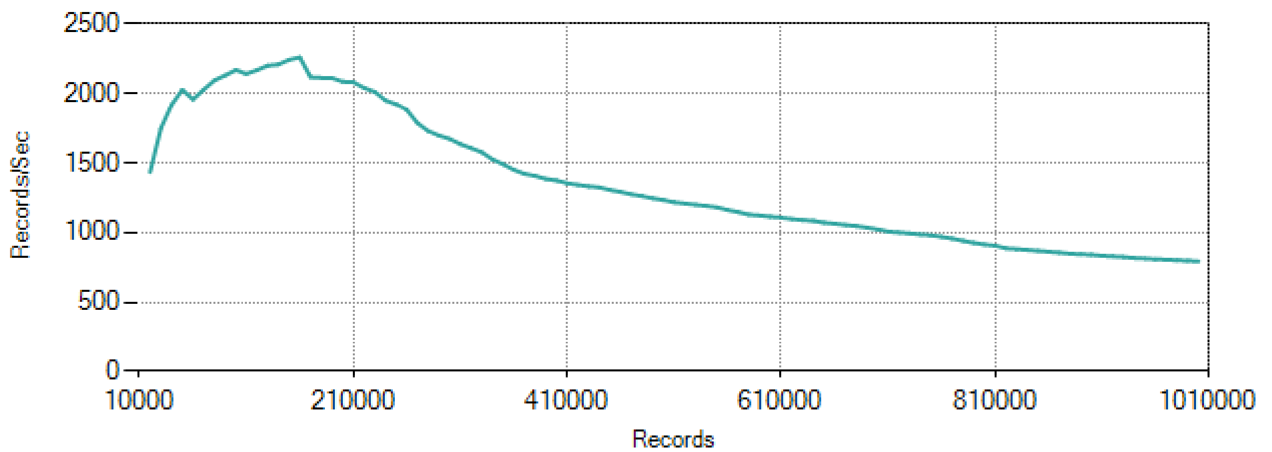

Рисунок 3.7 - Графік залежності середньої швидкості додання нових записів від кількості записів в SQL Server

Рисунок 3.7 демонструє, що швидкість додання нових записів починає стрімко спадати після того, як кількість записів досягне приблизно 200000, але навіть в кращому випадку SQL Server не може сперечатись із більш швидкими у цьому плані базами даних VelocityDb, db4objects, Volante.

Проте розмір бази даних дорівнює всього 85Mbyte, що є другим результатом після VelocityDb.

Але жодна із протестованих об’єктних баз даних не перегнала SQL Server у плані швидкості читання даних. Крім того, SQL Server не вимагала великих затрат оперативної пам’яті. Менше оперативної пам’яті вимагає лише база даних Perst.

MS SQL Server є ідеальною для збереження даних, якщо в інформаційній системі виникає необхідність швидкого вилучення даних, а нові дані заносяться не дуже часто. Також важливою перевагою SQL Server над іншими базами даних є велика кількість гарно структурованої документації, чого не можна сказати про інші бази даних.

3.3.2 Результат тестування об’єктно-орієнтованих баз даних

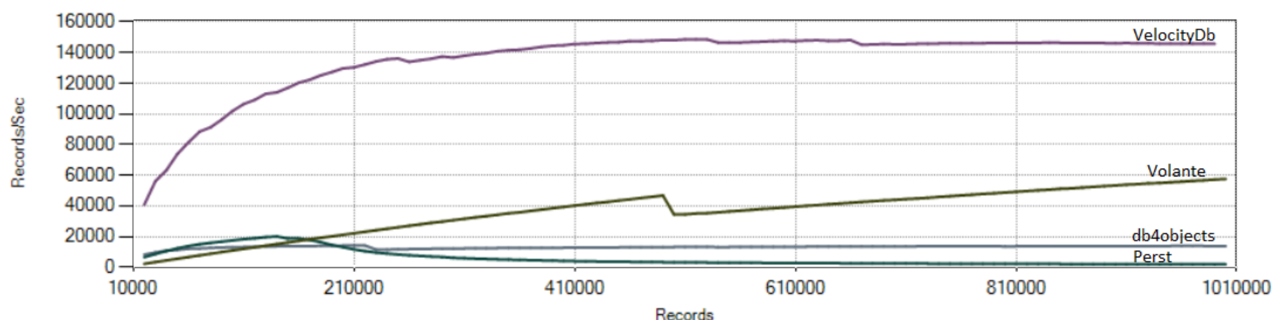

Рисунок 3.8: Аналіз швидкості додавання записів ООБД

Серед об’єктних баз даних, по-перше слід виділити базу даних, що явно програла іншим у плані продуктивності. Цією базою даних є Perst. Хоча вона показує найкращі результати в плані затрат оперативної пам’яті, вона явно програє всім у плані швидкості вилучення інформації, і виграє лише SQL Server у плані швидкості додавання інформації.

Єдиною причиною використовувати Perst є необхідність мінімізації затрат оперативної пам’яті, але MS SQL Server програє Perst у цьому плані не дуже сильно. Крім цього, Perst не має багатьох очевидних сильних сторін, що є у SQL Server.

База даних VelocityDb виграє відразу у трьох характеристиках: розмір заповненої бази даних та швидкість додання нових записів. Крім того, швидкість повторного читання даних близька до значення цієї швидкості в MS SQL Server.

БД Volante виграє VelocityDb по значенню швидкості першого читання записів, але всі операції додавання, читання і повторного читання записів дуже дорогі. Додавання і читання одного мільйону записів із використанням Volante коштувало близько 2.5 Gbyte RAM.

БД Volante і VelocityDb являються in-memory database. Вони вирішують проблему дуже повільного доступу до пам’яті на жорсткому диску. In-memory

databases спираються на збереження даних на оперативному запам'ятовуючому пристрої. Це робить доступ до даних дуже швидким.

Серед протестованих об'єктних баз даних, очевидно, найкращою можна назвати VelocityDb. Її можна використовувати при необхідності швидкої обробки великої кількості даних, а також для швидкого зчитування даних, особливо при необхідності їх повторного зчитування. У випадку необхідності використання безкоштовної об'єктної бази даних хорошим варіантом буде Db Volante.

3.4 Висновок

У розділі розглянуті бази даних Caché та VelocityDb, а також була проведена порівняльна характеристика продуктивності баз даних db4object, VelocityDb, Perst, Volante та MS SQL Server. Порівняння здійснювалось із допомогою програми Database Benchmark. Результат виконання показав, що серед баз даних, що розглядались, найкращі показники показали MS SQL Server та VelocityDb. Якщо SQL Server показав найкращі результати по швидкості читання інформації, VelocityDb – по швидкості її додання. При повторному читанні цих же даних швидкість бази даних VelocityDb наближається до швидкості MS SQL Server.

Також у розділі показано, що ООБД можна застосовувати у багатьох сферах, де є необхідність зберігання складно структурованих даних. Так, пропрієтарна база даних Caché застосовується у медицині, космічній промисловості тощо.

4. РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

4.1 Опис інформаційної системи

У дипломній роботі реалізована інформаційна система бібліотеки, що використовує у якості сховища даних об’єктно-орієнтовану базу даних VelocityDb. Вибір бази даних обгрунтований результатом порівняльного аналізу баз даних, результати якого показані у попередньому розділі.

Інформаційна система зберігає у базі інформацію про працівників і клієнтів, а також про книги і історію їх користування клієнтами. Працівник може додавати нові книги, а також відмічати, що конкретний клієнт взяв чи повернув книгу. В кожній із книг зберігається їх історія користування клієнтами, а саме зберігається клієнт, дата початку користування, дата кінця користування (якщо вона відмічена, як NULL, то книга не повернута), і працівник, який віддав книгу у користування користувачу.

Також інформаційна система здатна повідомляти користувача про те, що у якихось його колег сьогодні день народження, а також є можливість перегляду книг, які затримують клієнти.

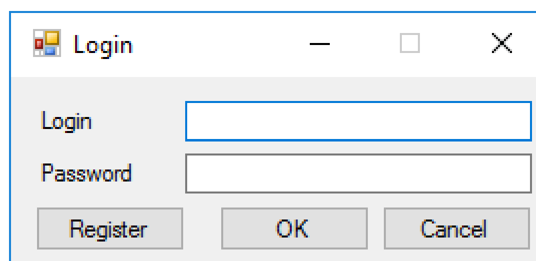

Рисунок 4.1 – Форма входу користувача

Для початку користування програмою користувачу потрібно буде авторизуватись (форма авторизації зображена на рисунку 4.1). Якщо у працівника існує обліковий запис, можна ввести логін і пароль, у іншому випадку – потрібно натиснути кнопку Register.

Для реєстрації користувача потрібно обов’язково вказати свої прізвище та ім’я, а також логін та пароль (рисунок 4.2).

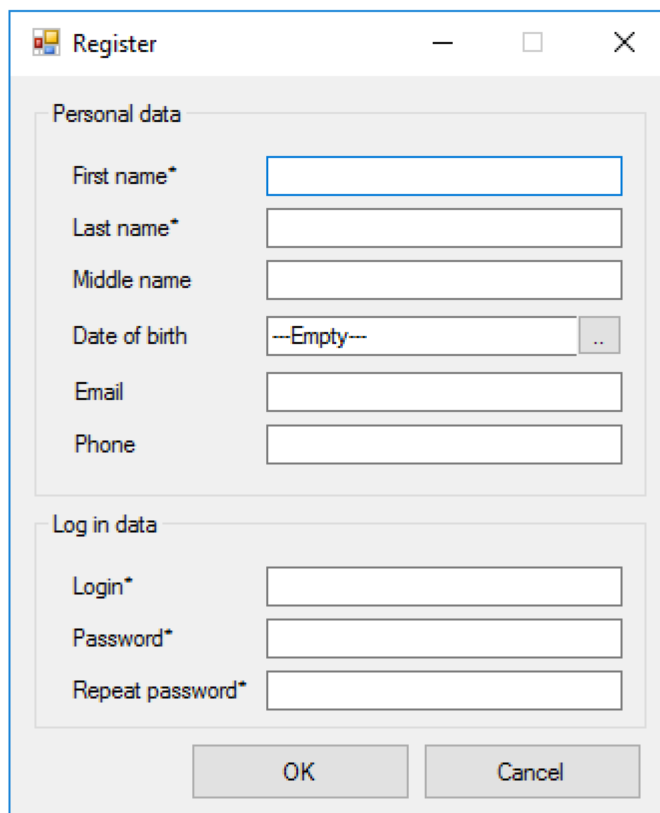

Рисунок 4.2 – Форма реєстрації

Пістя авторизації програма перенаправляє користувача на вікно із головним меню програми, зображеним на рисунку 4.3.

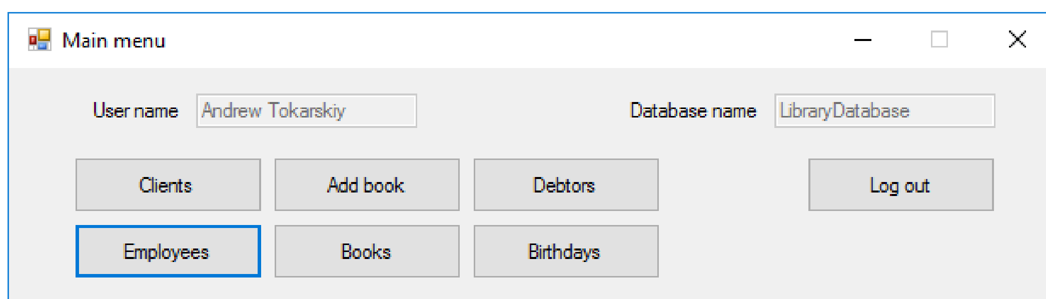

Рисунок 4.3 – Головне меню програми

Із меню можна перейти на вікна із клієнтами, працівниками. Також можна перейти на вікно додання книги, списку книг, списку клієнтів-боржників та списку людей, у яких сьогодні день народження.

Для перегляду списку книг потрібно натиснути на клавішу Books. Відкриється вікно, зображене на рисунку 4.4. Подвійний клік на рядок із книгою відкриє детальну інформацію про книгу. Тоді відкриється вікно, зображене на рисунку 4.5.

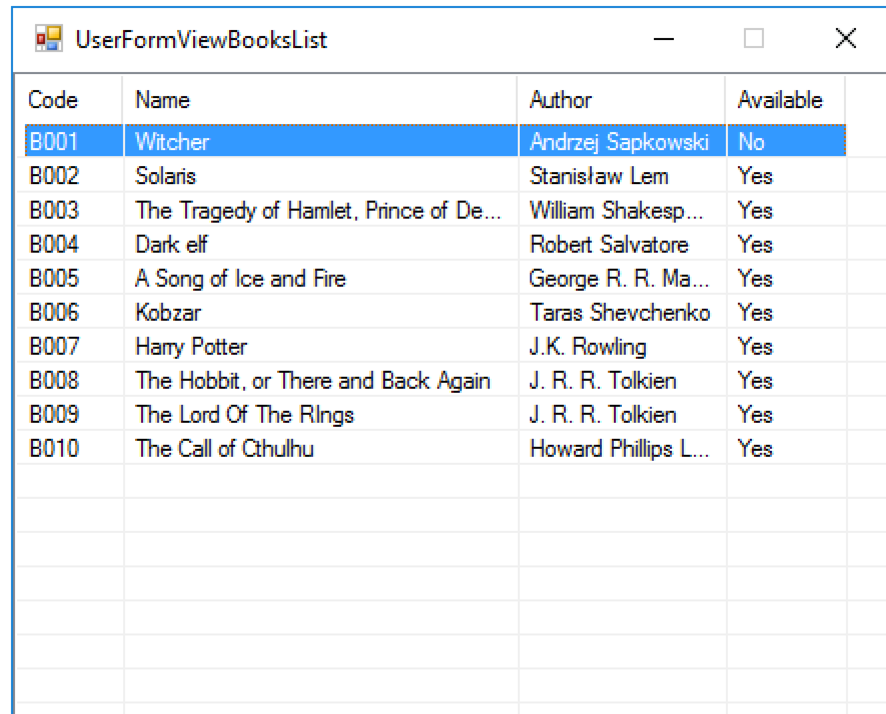

Рисунок 4.4 – Вікно із списком книг

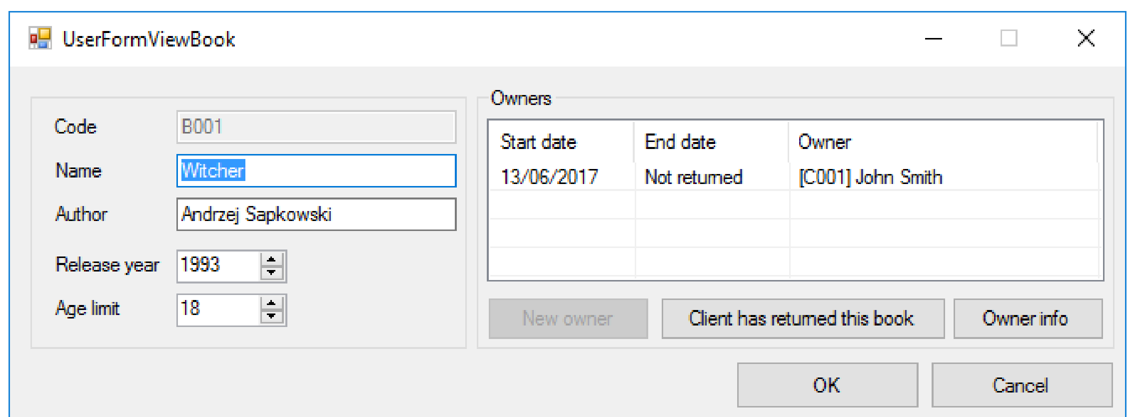

Рисунок 4.5 – Вікно із детальною інформацією про книгу

Серед інформації, що зберігається у книзі, можна виділити її код, назву, ім'я автора, рік випуску, вікові обмеження та історію користування. Із інформації із вікна, зображеного на рисунку 4.5, можна сказати, що користувач із кодом C001 та іменем John Smith взяв книгу Witcher 13 червня 2017 року та

ще не повернув її. Натиснення кнопки Client has returned this book підтверджує, що клієнт повернув книгу. Із допомогою кнопки New owner можна вибрати нового користувача книгою, а дата початку користування для нього відразу ж встановиться на сьогоднішню. До речі, для книг із віковим обмеженням не можна вказати клієнта із невказаним віком або віком молодше значення вікового обмеження у якості користувача.

Натиснення кнопки Owner info відкриє інформацію про людину, що зараз володіє даною книгою.

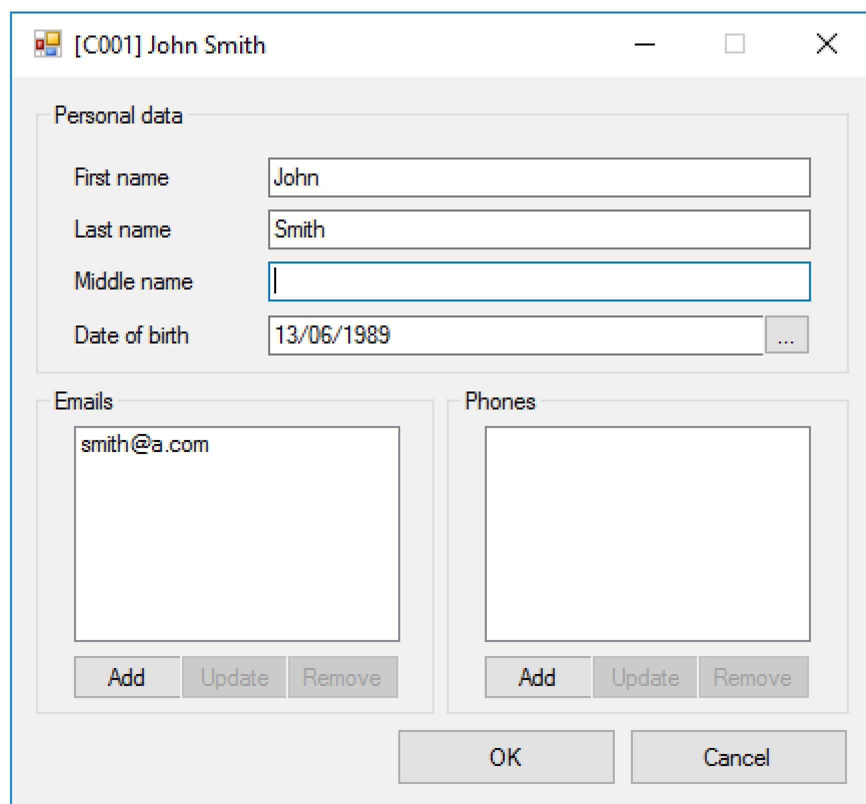

Рисунок 4.6 – Вікно із інформацією про людину

На рисунку 4.6 зображене це вікно. Як бачимо, серед персональної інформації людини є її ПІБ, дата народження, що може бути і невказаною, а також списки її телефонів та електронних адрес. Натиснення кнопки ОК підтверджує зміни інформації.

Вищевказані вікна показують структуру даних, що зберігаються у розробленій інформаційній системі.

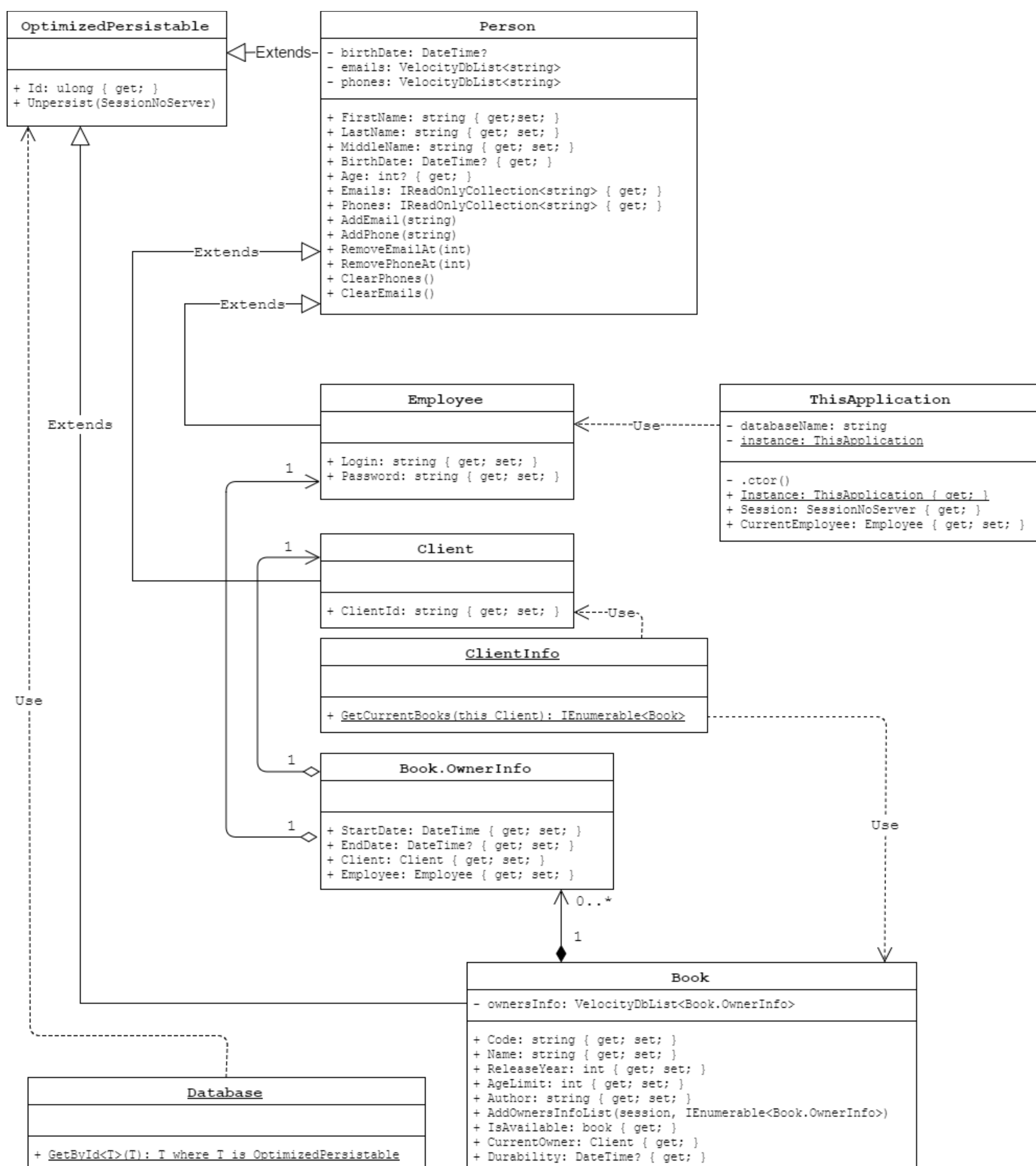

Рисунок 4.7 – UML-діаграма класів інформаційної системи

На рисунку 4.7 зображена UML-діаграма класів, які використовує дана інформаційна система. Клас OptimizedPersistable – клас, що знаходиться у бібліотеці вбудованої бази даних VelocityDb, тобто створювався не мною. У ньому є дуже велика кількість методів та властивостей, на діаграмі сною вказані лише найнеобхідніші для мене – властивість Id та метод Unpersist.

Id використовується для зберігання ідентифікатора об’єкта, що зберігається у базі.

Метод Unpersist створений для видалення об’єкта із бази даних. Варто відмітити, що цей метод є віртуальним, і його при необхідності можна перевантажити і написати власну реалізацію очистки, якщо це потребується. Наприклад, у класі Person цей метод перевантажений таким чином, що він видаляє спочатку списки телефонів та електронних адрес, а потім і сам об’єкт.

OptimizedPersistable у даній інформаційній системі є базовим класом для класів Person та Book. У свою чергу, класи Employee та Client походять від Person.

Клас Database є статичним, у ньому зберігається лише один статичний метод, реалізований для спрощення отримання об’єктів із бази даних.

Клас ThisApplication є класом, що зберігає у собі налаштування програми, а саме ім’я бази даних та авторизований працівник.

4.2 Висновок

Була реалізована інформаційна система, що використовує базу даних VelocityDb у якості сховища даних. Розроблена із використанням .NET Framework + WinForms. База даних із допомогою інформаційної системи була заповнена тестовими даними про книги, клієнтів і працівників бібліотеки. Запити виконуються достатньо швидко.

Слід відмітити, що архітектура ООБД, хоч і повністю повторює архітектуру класів, але для використання ООБД необхідні зміни у архітектуру класів. Так, наприклад, є необхідність класи даних, що зберігаються, робити потомками класу OptimizedPersistable.

Також об’єктні бази даних незручно використовувати для переносу даних на іншу об’єктну базу даних через необхідність знову ж міняти архітектуру.

5. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту бібліотеки. Програмний продукт буде створюватись із використанням платформи .NET та використовуватиме об’єктно-орієнтовану базу даних у якості сховища даних.

Програма створена для збереження інформації про клієнтів бібліотеки та для того, щоб вести облік книг.

Програмний продукт повинен добре працювати навіть на комп’ютерах із низькими характеристиками.

Нижче наведено аналіз різних варіантів реалізації програмного продукту з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

1. Визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.
2. Для кожної функції визначаються повні річні витрати й кількість робочих часів.
3. Для кожної функції наоснові оцінок попереднього пункту визначається кількісна характеристика джерел витрат.
4. Після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

5.1 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічний аналізу розробки.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

1. Програмний продукт повинен функціонувати на персональних комп’ютерах із стандартним набором компонент;
2. Забезпечувати цілодобовий доступ співробітників бібліотеки до інформації
3. Забезпечувати зручність і простоту взаємодії з користувачем;
4. Передбачати мінімальні витрати на впровадження програмного продукту.

5.1.1 Обґрунтування функцій програмного продукту

Головна функція F_0 – обрання програмного продукту для впровадження у бібліотеку. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

F_1 – вибір реалізації платформи .NET

F_2 – вибір об'єктно-орієнтованої бази даних

F_3 – вибір графічної оболонки

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

а) .NET Framework

б) Mono

Функція F_2:

а) Volante

б) VelocityDb

Функція F_3:

а) Windows Forms

б) GTK#

5.1.2 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 5.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 5.1).

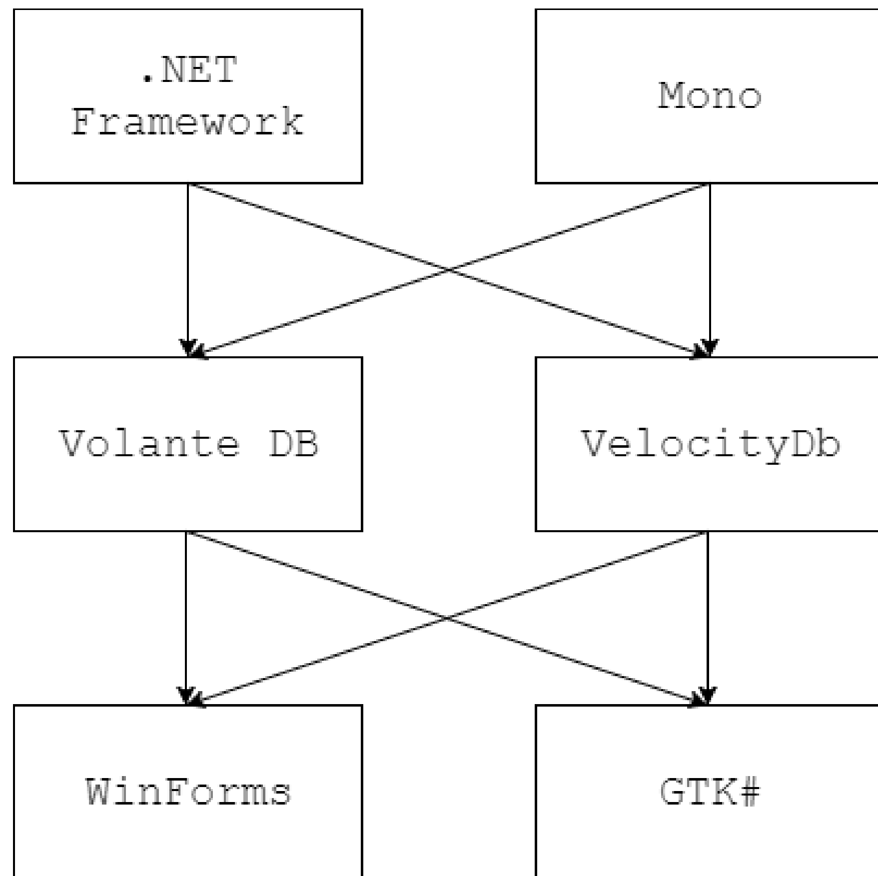

Рисунок 5.1 – Морфологічна карта

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

Для того, щоб перечислити переваги і недоліки кожного із підходів, можна побудувати позитивно-негативну матрицю. Це матриця, у якій окремо вказуються переваги і недоліки для кожного із підходів.

На основі аналізу позитивно-негативної матриці (таблиця 5.1) робимо висновок, що при обрані програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам.

Таблиця 5.1 – Позитивно-негативна матриця

Основні функції	**Варіанти реалізації**	**Переваги**	**Недоліки**
F1	*А*	Хороша документація, надійність, висока швидкість, повна сумісність із свіжими версіями C# і VB.NET.	Не є кросплатформенним
	Б	Open-source, кросплатформенність.	В окремих місцях працює повільніше, ніж .NET Framework. Сумісність із останніми версіями C# і VB.NET не гарантується.
F2	А	БД із дуже швидким доступом до даних, безкоштовна для будь-якого використання, open-source.	Мало документації, великий розхід оперативної пам'яті.
	Б	Швидка, легка для розуміння. Безкоштовна для некомерційного використання	Мало документації.
F3	*А*	Легко використовувати, гарний інтерфейс, велика спільнота активних користувачів	Код, що викликається із Win32 API не можна портувати
	Б	Open-source, крос-платформенність	Неготова документація,

Функція *F1:*

Будемо використовувати .NET Framework, так як він є більш стабільним, а програмі не потрібно бути крос-платформенною, бо більшість користувачів не знайомі із Unix-системами, а на комп’ютери більшості навчальних закладів встановлений Windows.

Функція *F2:*

Для точної оцінки потрібно розглянути альтернативи Volante DB та VelocityDb детальніше із залученням експертів.

Функція *F3:*

Оскільки, як було розглянуто раніше, крос-платформеність не є основною задачею, для розробника необхідно швидко та зручно розробити гарний інтерфейс, достатнім вибором буде Windows Forms.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

1. F1a – F2a – F3a
2. F1a – F2б – F3a

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

5.2 Обґрунтування системи параметрів ПП

5.2.1 Опис параметрів

На підставі даних про основні функції, що повинен мати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

– *X1* – навантаження процесора при виконанні коду;

– *X2* – час відклику вікон;

– *X3* – затрати оперативної пам’яті;

– *X4* – розмір бази даних.

X1: Відображає кількість записів, що може заповнити база даних за 1 секунду.

X2: Відображає кількість записів, що може прочитати база даних за 1 секунду

X3: Відображає затрати оперативної пам’яті в мегабайтах під час роботи із базою даних.

X4: Показує розмір заповненої бази даних.

5.2.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 4.2.

Таблиця 5.2 – Основні параметри ПП

Назва Параметра	**Умовні позначення**	**Одиниці виміру**	**Значення параметра**		
			гірші	**середні**	**кращі**
Навантаження процесора при виконанні коду	X1	%	100	60	30
Час відклику вікон	X2	мс	3000	500	40
Затрати RAM	X3	Мб	2000	600	200
Розмір БД	X4	Мб	300	150	50

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 5.2 – рис. 5.5.

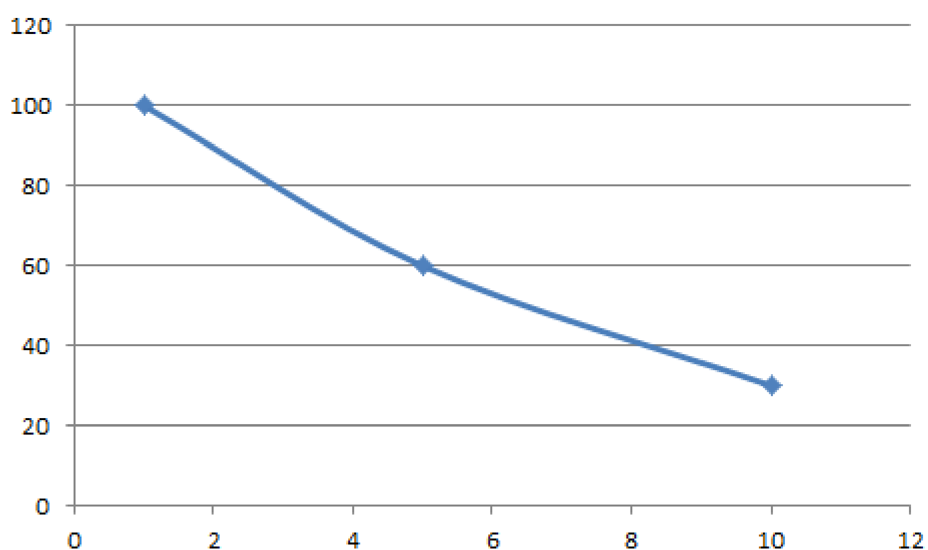

Рисунок 5.2 – X1, Навантаження процесора під час виконання коду, %

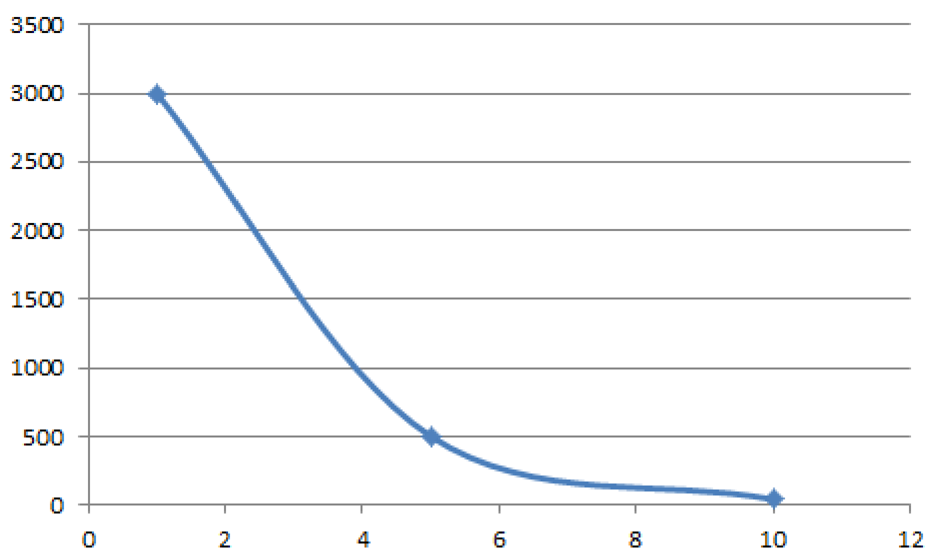

Рисунок 5.3 – X2, Час відклику вікон, мс

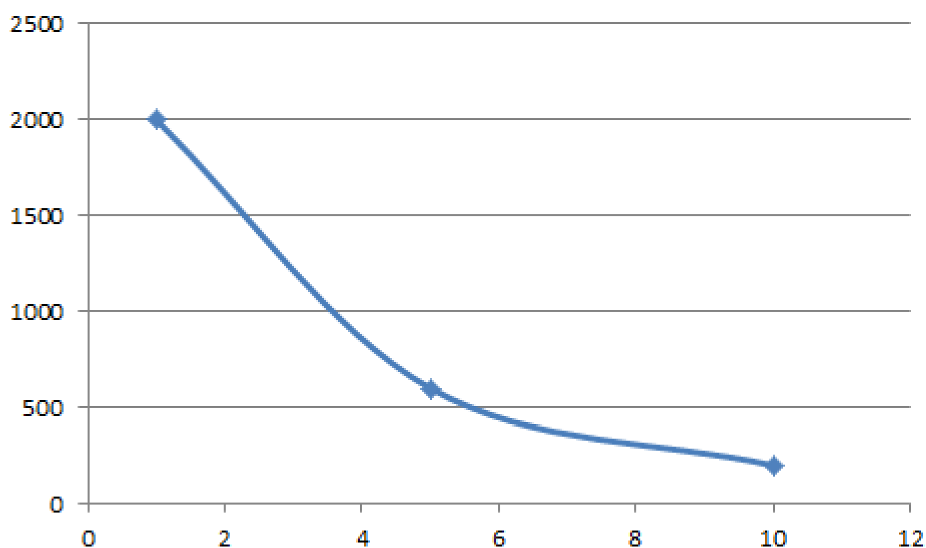

Рисунок 5.4 – X3, Затрати RAM, Мб

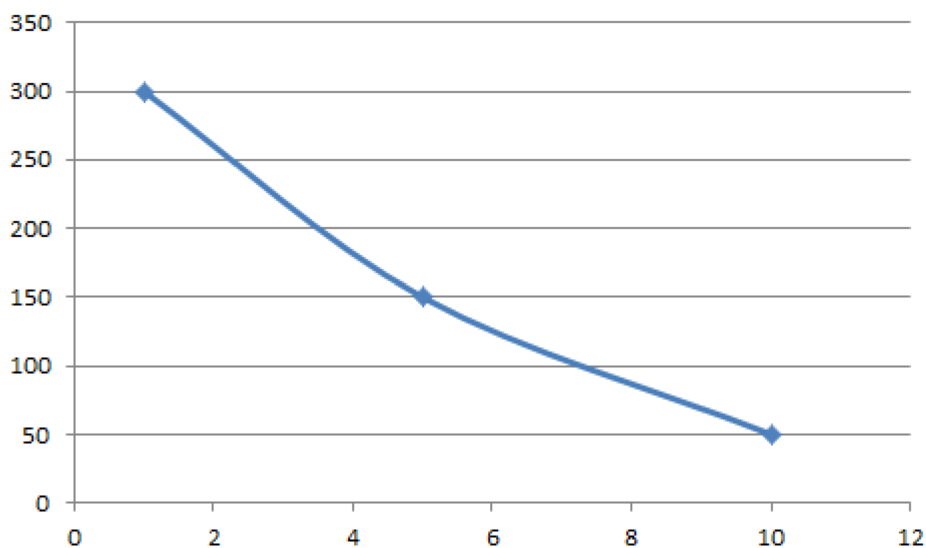

Рисунок 5.5 – X4, Розмір БД, Мб

5.2.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – вибір програмного продукту, який має найбільш зручний та функціональний набір можливостей при низький необхідності у ресурсах.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

– визначення рівня значимості параметра шляхом присвоєння різних рангів;

– перевірку придатності експертних оцінок для подальшого використання;

– визначення оцінки попарного пріоритету параметрів;

– обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 5.3.

Таблиця 5.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Навантаження процесора при виконанні коду	зап/сек	1	2	1	2	1	1	2	10	-7,5	56,25
X2	Час відклику вікон	зап/сек	2	1	2	1	2	3	1	12	-5,5	30,25
X3	Затрати RAM	Мб	4	3	3	4	4	4	3	26	8,5	56,25
X4	Розмір БД	Мб	3	4	4	3	3	2	4	22	4,5	30,25
	Разом		10	10	10	10	10	10	10	70	0	173

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70,$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 173.$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 179}{7^2(4^3 - 4)} = 0{,}706 > W_k = 0{,}67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 5.4.

Таблиця 5.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	>	<	>	<	<	>	<	0.5
X1 і X3	<	<	<	<	<	<	<	<	0,5
X1 і X4	<	<	<	<	<	<	<	<	0,5
X2 і X3	<	<	<	<	<	<	<	<	0,5
X2 і X4	<	<	<	<	<	>	<	<	0.5
X3 і X4	>	<	<	>	>	>	<	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1{,}5 \text{ при } x_i > x_j \\ 1{,}0 \text{ при } x_i = x_j \\ 0{,}5 \text{ при } x_i < x_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю A=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$$K_{вi} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \text{де } b_i = \sum_{i=1}^{N} a_{ij}.$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{вi} = \frac{b'_i}{\sum_{i=1}^{n} b'_i}, \text{де } b'_i = \sum_{i=1}^{N} a_{ij} b_j.$$

Як видно з таблиці 5.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 5.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1	0,5	0,5	0,5	2,5	0,15625	9,25	0,156	34,125	0,157
X2	1,5	1	0,5	0,5	3,5	0,21875	12,25	0,207	44,875	0,207
X3	1,5	1,5	1	1,5	5,5	0,34375	21,25	0,360	77,875	0,360
X4	1,5	1,5	0,5	1	4,5	0,28125	16,25	0,275	59,125	0,273
Всього:					16	1	59	1	216	1

5.3 Аналіз рівня якості варіантів реалізації функцій

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 5.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j} ,$$

де n – кількість параметрів;$K_{вi}$– коефіцієнт вагомості i–го параметра;B_i – оцінка i–го параметра в балах.

Таблиця 5.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	50	9	0,157	1,413
F2	А	X3	2423	1	0,360	0,36
		X4	123	7	0,273	1,911
	Б	X3	272,5	8	0,360	2,88
		X4	53.1	9	0,273	2,457
F3	А	X2	30	9	0,273	2,457

За даними з таблиці 5.6 за формулою

$$К_К = К_{ТУ}[F_{1k}] + К_{ТУ}[F_{2k}] + \dots + К_{ТУ}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

$К_{К1} = 1,413 + 0,36 + 1,911 + 2,457 = 6,141$

$К_{К2} = 1,413 + 2,88 + 2,457 + 2,457 = 9,207$

Як видно з розрахунків, кращим є другий варіант, для якого коефіцієнт технічного рівня має найбільше значення.

5.4 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка архітектури класів та їх логіки;

2. Реалізація графічного інтерфейсу

3. Тестування програмного функціоналу;

Варіант I також має завдання

4. Вивчення БД Volante

Варіант II також має завдання

5. Вивчення БД VelocityDb

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється як

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М}, \qquad (5.1)$$

де T_P – трудомісткість розробки ПП;$K_П$ – поправочний коефіцієнт;$K_{СК}$ – коефіцієнт на складність вхідної інформації; $K_М$ – коефіцієнт рівня мови програмування;$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм; $K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни Б та групи складності алгоритму 2, трудомісткість дорівнює: T_P =27 людино-днів. Поправочний коефіцієнт, який враховує вид

нормативно-довідкової інформації для першого завдання: $К_П$ = 1.08. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді, за формулою 5.1, загальна трудомісткість програмування першого завдання дорівнює:

$Т_1 = 27 \cdot 1.08 \cdot 0.8 = 23.33$ людино-днів.

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм другої групи складності, степінь новизни В), тобто $Т_Р$ =19 людино-днів, $К_П$ =0.72,$К_{СК}$ = 1,$К_{СТ}$ =0.8:

$Т_2 = 19 \cdot 0.72 \cdot 0.8 = 10.94$ людино-днів.

Для третього завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ =19 людино-днів, $К_П$ =0.9,$К_{СК}$ = 1,$К_{СТ}$ =0.8:

$Т_3 = 19 \cdot 0.8 \cdot 0.9 = 13.68$ людино-днів.

Для четвертого завдання (використовується алгоритм другої групи складності, степінь новизни А), тобто $Т_Р$ =36 людино-днів, $К_П$ =1.51,$К_{СК}$ = 1,$К_{СТ}$ =1:

$Т_4 = 36 \cdot 1.51 = 54.36$ людино-днів.

Для п’ятого завдання (використовується алгоритм другої групи складності, степінь новизни А), тобто $Т_Р$ =27 людино-днів, $К_П$ =1.26,$К_{СК}$ = 1,$К_{СТ}$ =1:

$Т_5 = 27 \cdot 1.51 = 34.02$ людино-днів.

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

T_I = (23.33 + 10.94 + 13.68 + 54.36) · 8 = 818.48 людино-годин;

T_{II} = (23.33 + 10.94 + 13.68 + 34.02) · 8 = 655.76 людино-годин;

Найбільш високу трудомісткість має варіант I.

В розробці беруть участь два програмісти з окладом 5000 грн., один фінансовий аналітик з окладом 9000грн. Визначимо зарплату за годину за формулою:

$$С_ч = \frac{М}{T_m \cdot t} \text{ грн.},$$

де М – місячний оклад працівників;T_m – кількість робочих днів тиждень;t – кількість робочих годин в день.

$$С_ч = \frac{5000 + 9000 + 5000}{3 \cdot 21 \cdot 8} = 37.7 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою

$С_{ЗП} = С_ч \cdot Т_i \cdot К_Д$,

де $С_Ч$– величина погодинної оплати праці програміста;$Т_i$ – трудомісткість відповідного завдання; $К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП}$ = 37.7· 818.48 · 1.2 = 37028.03 грн.

II. $С_{ЗП}$ = 37.7· 655.76 · 1.2 = 28942.04 грн.

Відрахування на соціальне страхування становить 22%:

I. $С_{ВІД}$ = $С_{ЗП}$ · 0.22 = 37028.03 * 0.22 = 8146.16 грн.

II. $С_{ВІД}$ = $С_{ЗП}$ · 0.22 = 28942.04 * 0.22 = 6367.25 грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 5000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 5000 \cdot 0{,}2 = 12000$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 12000 \cdot (1 + 0.2) = 14400$ грн.

Відрахування на соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0.22 = 14400 * 0{,}22 = 3168$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 10000 грн.

$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 10000 = 2875$ грн.,

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$– річна норма амортизації; $Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 10000 \cdot 0.05 = 575$ грн.,

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 16) \cdot 8 \cdot 0.9 = 1706.4$ годин,

де $Д_К$ – календарна кількість днів у році; $Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів; $Д_Р$ – кількість днів планових ремонтів устаткування; t –кількість робочих годин в день; $К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$C_{ЕЛ} = T_{ЕФ} \cdot N_{C} \cdot K_{З} \cdot Ц_{ЕН} = 1706,4 \cdot 0,75 \cdot 1.93819 = 2480.49$ грн.,

де N_{C} – середньо-споживча потужність приладу; $K_{З}$– коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$C_{Н} = Ц_{ПР} \cdot 0.67 = 12000 \cdot 0,67 = 8040$ грн.

Тоді, річні експлуатаційні витрати будуть:

$C_{ЕКС} = C_{ЗП} + C_{ВІД} + C_{А} + C_{Р} + C_{ЕЛ} + C_{Н}$

$C_{ЕКС} = 14400 + 3168 + 2875 + 575 + 2480.49 + 8040 = 31538.49$ грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$C_{М-Г} = C_{ЕКС} / T_{ЕФ} = 31538.49 / 1706,4 = 18.48$ грн/год.

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$C_{М} = C_{М-Г} \cdot T$

I. $C_{М} = 18.48 * 818.48 = 15135.51$ грн.;

II. $C_{М} = 18.48 * 655.76 = 12118.44$ грн.;

Накладні витрати складають 67% від заробітної плати:

$C_{Н} = C_{ЗП} \cdot 0,67$

I. $C_{Н} = 37028.03 * 0,67 = 24808.78$ грн.;

II. $C_{Н} = 28942.04 * 0,67 = 19391.17$ грн.;

Отже, вартість розробки ПП за варіантами становить:

$C_{ПП} = C_{ЗП} + C_{ВІД} + C_{М} + C_{Н}$

I. $С_{ПП}$ = 37028.03 + 8146.16 + 15135.51 + 24808.78 = 85118.48 грн.;

II. $С_{ПП}$ = 28942.04 + 6367.25 + 12118.44 + 19391.17 = 66818.90 грн.;

5.5 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$К_{ТЕРj} = К_{Кj} / С_{Фj}$,

$К_{ТЕР1} = 6.141 / 85118.48 = 0,7 \cdot 10^{-4}$;

$К_{ТЕР2} = 9.207 / 66818.90 = 1,3 \cdot 10^{-4}$;

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 1,3 \cdot 10^{-4}$.

5.6 Висновок

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом

ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 0{,}14 \cdot 10^{-4}$.

Цей варіант реалізації програмного продукту має такі параметри:

– мова програмування – C#;

– платформа .NET Framework

– VelocityDb у якості сховища даних;

– інтерфейс користувача, створений за технологією Windows Forms.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, непоганий функціонал і швидкодію.

ВИСНОВКИ

Об'єктно-орієнтовані бази даних мають дуже цікаву концепцію в порівнянні з класичними реляційними базами даних: дані, що зберігаються у вигляді об'єктів, інкапсуляція, успадкування, поліморфізм - всі ці особливості забезпечують зручну роботу і велику швидкодію.

У першому розділі була розглянута загальна концепція об’єктно-орієнтованих баз даних. Концепція ООБД була порівняна із концепцією реляційних БД, а також розглянуті приклади, що демонструють особливості роботи в ООБД.

ООБД є достатньо зручними для роботи із даними, що мають складну структуру. Із їх допомогою нескладно зберігати, наприклад, набір об’єктів, атрибутами якого являються масиви, списки, хеш-таблиці або, взагалі, інші об’єкти.

Другий розділ був присвячений деяким технічним аспектам об’єктно-орієнтованих баз даних: були розглянуті особливості індексації та проблеми наслідування.

Для швидкої індексації більшість об’єктно-орієнтованих баз даних використовують B-дерева або їх модифікації. Перевага їх у тому, що вузли зберігають цілий набір ключів, а кількість дискових операції для доступу до даних є дуже малою завдяки дуже невисокій висоті дерева навіть при величезній кількості ключів.

У третьому розділі були розглянуті існуючі об’єктно-орієнтовані бази даних, а також було виконаний тест продуктивності деяких баз даних. Результати тесту показали, що реляційні бази даних дуже добре підходять для інформаційних систем, у яких необхідно часто вилучати дані, і в яких робота проходить із вже сформованим набором даних. Серед об’єктних баз даних є БД

із дуже великою швидкістю додавання і оновлення великої кількості об’єктів із складною структурою.

Також були розглянуті деякі приклади використання ООБД. Так, їх можна використовувати у галузях, де є найрізноманітніші види об’єктів, кожен із яких має свій набір атрибутів. Такими галузями є, наприклад, астрономія, медицина, географія тощо.

В процесі виконання дипломної роботи також була реалізована інформаційна система бібліотеки, що у якості сховища даних використовує об’єктно-орієнтовану базу даних VelocityDb. Інформаційна система описана у четвертому розділі.

Але, на жаль, вони мають дуже слабку теоретичну базу, мізерну документацію і дуже малу популярність для того, щоб домогтися успіху на ринку. У реляційні БД вкладені великі гроші, і дуже багато проектів успішно використовують їх для зберігання даних. Дуже велика кількість IT-гігантів (наприклад, Microsoft, Oracle) реалізували свої реляційні бази даних, що у якості мови запиту використовують SQL. Перехід від одної БД до іншої майже не представляє проблем. Реляційні бази даних мають потужну теоретичну базу, велику кількість документації, широку спільноту і хороший рівень технічної підтримки.

ПЕРЕЛІК ПОСИЛАНЬ

1. David Maier. Object-Orienter Database Theory. An Introduction / David Maier – Muenchen : TU Muenchen. – 2001
2. Объектно-ориентированные базы данных - основные концепции, организация и управление: краткий обзор. – Режим доступа: http://citforum.ru/database/articles/art_24.shtml/. – Дата доступа: 13.04.2017
3. Thomas A. Mueck. Index Data Structures in Object-Oriented Databases / Thomas A. Mueck – Luxembourg : Springer Science & Business Media. – 1997
4. Официальный сайт компании Intersystems. – Режим доступа: http://www.intersystems.com/. – Дата доступа: 25.05.2017
5. PostgreSQL. Tutorial inheritance. – Режим доступа: https://www.postgresql.org/docs/8.4/static/tutorial-inheritance.html/. – Дата доступа: 27.05.2017
6. S. K. Singh. Database Systems: Concepts, Design and Applications / S. K. Singh – London : Dorling Kinderslay. – 2011
7. Architecture In Object Oriented databases. – Режим доступа: http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.116.434/. – Дата доступа: 29.05.2017
8. Thomas H. Cormen. Introduction to algorithms. Second Edition / Thomas H. – Cambridge : The MIT Press. – 2005, pp. 521-525

ДОДАТОК А

ТЕКСТ ПРОГРАМИ

```
// --------------------------------Database.cs-------------------------------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using VelocityDb;

namespace LibraryHelper
{
    public static class Database
    {
        public static T GetById<T>(ulong id) where T : OptimizedPersistable
        {
            using (var conn = ThisApplication.Instance.Session)
            {
                conn.BeginRead();
                var ret = conn
                    .AllObjects<T>()
                    .FirstOrDefault(x => x.Id == id);
                conn.Commit();
                return ret;
            }
        }
    }
}


// ------------------------------Person.cs------------------------------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using VelocityDb;
using VelocityDb.Collection;
using VelocityDb.Session;

namespace LibraryHelper
{
    public class Person : OptimizedPersistable
    {
        private DateTime? birthDate;
        private VelocityDbList<string> emails;
        private VelocityDbList<string> phones;

        public Person() : base()
        {
            emails = new VelocityDbList<string>();
            phones = new VelocityDbList<string>();
        }

        /// <summary>
        ///     Имя
        /// </summary>
```

```
public string FirstName { get; set; }

/// <summary>
///     Фамилия
/// </summary>
public string LastName { get; set; }

/// <summary>
///     Отчество
/// </summary>
public string MiddleName { get; set; }

/// <summary>
///     Дата рождения человека
/// </summary>
public DateTime? BirthDate
{
    get { return birthDate; }
    set
    {
        if (value.HasValue)
        {
            birthDate = value.Value.Date;
        }
        else
        {
            birthDate = null;
        }
    }
}

/// <summary>
///     Возраст человека. Явно установить нельзя, зависит от дня
///     рождения
///     и от текущей даты
/// </summary>
public int? Age
{
    get
    {
        if (birthDate == null)
        {
            return null;
        }

        var today = DateTime.Today;
        int age = today.Year - birthDate.Value.Year;
        if (birthDate > today.AddYears(-age))
        {
            age--;
        }
        return age;
    }
}

/// <summary>
///     Список электронных адресов человека
/// </summary>
public IReadOnlyCollection<string> Emails
{
    get
    {
        return emails.ToList().AsReadOnly();
```

```
    }
}

/// <summary>
///     Список телефонных номеров человека
/// </summary>
public IReadOnlyCollection<string> Phones
{
    get
    {
        return phones.ToList().AsReadOnly();
    }
}

public void AddEmail(string email)
{
    emails.Add(email);
}

public void AddPhone(string phone)
{
    phones.Add(phone);
}

public void RemoveEmail(string email)
{
    emails.Remove(email);
}

public void RemovePhone(string phone)
{
    phones.Remove(phone);
}

public void RemoveEmailAt(int index)
{
    emails.RemoveAt(index);
}

public void RemovePhoneAt(int index)
{
    phones.RemoveAt(index);
}

public void ClearEmails()
{
    emails.Clear();
}

public void ClearPhones()
{
    phones.Clear();
}

public override string ToString()
{
    return $"{FirstName} {LastName}";
}

public override void Unpersist(SessionBase session)
{
    var emails = this.emails;
    this.emails = null;
```

```
            emails.Unpersist(session);

            var phones = this.phones;
            this.phones = null;
            phones.Unpersist(session);

            base.Unpersist(session);
        }
    }
}

// -------------------------Employee.cs-------------------------------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LibraryHelper
{
    public class Employee : Person
    {
        public string Login { get; set; }
        public string Password { get; set; }
    }
}

// -------------------------Client.cs---------------------------------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using VelocityDb.Collection;

namespace LibraryHelper
{
    public class Client : Person
    {
        public string Code { get; set; }

        public override string ToString()
        {
            return $"[{Code}] {base.ToString()}";
        }
    }

    public static class ClientInformation
    {
        public static IEnumerable<Book> GetCurrentBooks(this Client client)
        {
            IEnumerable<Book> result;
            using (var conn = ThisApplication.Instance.Session)
            {
                conn.BeginRead();
                result =
                    from x in conn.AllObjects<Book>()
                    where x.CurrentOwner.Id == client.Id
                    select x;

                conn.Commit();
            }
            return result.ToList();
```

```
        }
    }
}

// -------------------------Book.cs---------------------------------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using VelocityDb;
using VelocityDb.Collection;
using VelocityDb.Session;

namespace LibraryHelper
{
    public class Book : OptimizedPersistable
    {
        public class OwnerInfo
        {
            public DateTime StartDate { get; set; }
            public DateTime? EndDate { get; set; }
            public Client Client { get; set; }
            public Employee Employee { get; set; }

            public TimeSpan Durability
            {
                get
                {
                    return !EndDate.HasValue ?
                           DateTime.Today - StartDate :
                           EndDate.Value - StartDate;
                }
            }

            public bool IsReturned
            {
                get
                {
                    return EndDate != null;
                }
            }
        }

        public string Code { get; set; }
        public string Name { get; set; }
        public int ReleaseYear { get; set; }
        public string Author { get; set; }
        public int AgeLimit { get; set; }

        private VelocityDbList<OwnerInfo> ownersInfo;

        public Book()
        {
            ownersInfo = new VelocityDbList<OwnerInfo>();
        }

        public IReadOnlyList<OwnerInfo> Owners
        {
            get
            {
                return ownersInfo.ToList().AsReadOnly();
            }
```

```
}

public void AddOwnerInfo(OwnerInfo info)
{
    ownersInfo.Add(info);
}

public void RemoveAt(int index)
{
    ownersInfo.RemoveAt(index);
}

public override void Unpersist(SessionBase session)
{
    this.ownersInfo = null;
    ownersInfo.Unpersist(session);

    base.Unpersist(session);
}

public bool IsAvailable
{
    get
    {
        return ownersInfo.Count != 0 ?
               ownersInfo.Last().IsReturned :
               true;
    }
}

public Client CurrentOwner
{
    get
    {
        OwnerInfo last = ownersInfo.LastOrDefault();
        if (last == null)
        {
            return null;
        }

        if (last.EndDate.HasValue)
        {
            return null;
        }

        return last.Client;
    }
}

public void AddOwnerInfoList(SessionNoServer session,
                             IEnumerable<OwnerInfo> info)
{
    if (session == null || info == null)
    {
        throw new ArgumentNullException();
    }

    var oinfo = ownersInfo;
    ownersInfo = null;
    oinfo.Unpersist(session);

    ownersInfo = new VelocityDbList<OwnerInfo>();
    foreach (var elem in info)
```

```
            {
                ownersInfo.Add(elem);
                session.UpdateObject(elem);
            }
        }

        public TimeSpan? Durability
        {
            get
            {
                var last = ownersInfo.LastOrDefault();
                if (last == null)
                {
                    return null;
                }

                return last.EndDate.HasValue ? null : new TimeSpan?
                                  (DateTime.Today - last.StartDate);
            }
        }

        public class BookNotReturnedException : Exception { }
    }
}

// ----------------------ThisApplication.cs----------------------------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using VelocityDb.Session;

namespace LibraryHelper
{
    public class ThisApplication
    {
        private static ThisApplication instance;
        private string databaseName;

        private ThisApplication()
        {
            databaseName = "TestDatabase7";
            using (var conn = new SessionNoServer(databaseName))
            {
                conn.BeginUpdate();
                conn.Commit();
            }
        }

        /// <summary>
        ///     Instance of program settings
        /// </summary>
        public static ThisApplication Instance
        {
            get
            {
                return instance ?? (instance = new ThisApplication());
            }
        }

        public Employee CurrentEmployee { get; set; }
        public string DatabaseName => databaseName ?? String.Empty;
```

```
        public SessionNoServer Session => new SessionNoServer(databaseName);
    }
}

// -------------------------UserFormBirthdayReminder.cs-----------------------
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace LibraryHelper
{
    public partial class UserFormBirthdayReminder : Form
    {
        private class BirthdayInfo
        {
            public ulong PersonId { get; set; }
            public string Name { get; set; }
            public int Age { get; set; }
        }

        private Dictionary<Type, List<BirthdayInfo>> birthdaysInfo;
        public UserFormBirthdayReminder()
        {
            InitializeComponent();
            SendQuery();
            FillBoxes();
        }

        private void SendQuery()
        {
            using (var conn = ThisApplication.Instance.Session)
            {
                conn.BeginRead();
                var query =
                    from person in conn.AllObjects<Person>()
                    where person.BirthDate != null
                    let day = person.BirthDate.Value.Day
                    let month = person.BirthDate.Value.Month
                    where day == DateTime.Today.Day &&
                          month == DateTime.Today.Month
                    orderby person.FirstName, person.LastName
                    group new BirthdayInfo
                    {
                        PersonId = person.Id,
                        Name = person.FirstName,
                        Age = person.Age.Value
                    }
                    by person.GetType();

                birthdaysInfo = new Dictionary<Type, List<BirthdayInfo>>();
                foreach (var group in query)
                {
                    birthdaysInfo.Add(group.Key, group.ToList());
                }

                conn.Commit();
```

```
            }
        }

        private void FillBoxes()
        {
            if (birthdaysInfo.ContainsKey(typeof(Employee)))
            {
                foreach (var employee in birthdaysInfo[typeof(Employee)])
                {
                    EmployeesBox.Items.Add(
                        $"{employee.Name}: {employee.Age} years old");
                }
            }
        }

        public void ShowIfBirthdaysExists(bool errorBox = false)
        {
            if (birthdaysInfo.Count != 0)
            {
                ShowDialog();
            }
            else if (errorBox)
            {
                MessageBox.Show("Did not find people who have birthday today. ",
                                "",
                                MessageBoxButtons.OK,
                                MessageBoxIcon.Information);
                return;
            }
        }
    }
}

//-----------------------UserFormAddBook.cs------------------------------
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace LibraryHelper
{
    public partial class UserFormAddBook : Form
    {
        private Book currentBook;
        public UserFormAddBook()
        {
            InitializeComponent();
            ageLimitBox.Maximum = DateTime.Today.Year;
        }

        private void OKButton_Click(object sender, EventArgs e)
        {
            using (var conn = ThisApplication.Instance.Session)
            {
                conn.BeginUpdate();
                currentBook = new Book
                {
                    Code = codeBox.Text,
```

```
                Name = nameBox.Text,
                Author = authorBox.Text,
                AgeLimit = (int)yearBox.Value,
                ReleaseYear = (int)ageLimitBox.Value
            };
            conn.Persist(currentBook);
            conn.Commit();
        }
        Close();
    }

    public Book ShowAndReturnBook()
    {
        ShowDialog();
        return currentBook;
    }

    private void CancelButton_Click(object sender, EventArgs e)
    {
        currentBook = null;
        Close();
    }
  }
}

// ---------------------UserFormViewPerson.cs---------------------------
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Microsoft.VisualBasic;
using System.Globalization;

namespace LibraryHelper
{
    public partial class UserFormViewPerson : Form
    {
        private Person currentPerson;
        private DateTime? birthDate;
        public UserFormViewPerson(Person person)
        {
            currentPerson = person;
            InitializeComponent();
            InitializeUserFormData();
            Text = person.ToString(); //Change the caption of the window

            UpdateEmailButton.Enabled = false;
            RemoveEmailButton.Enabled = false;
            UpdatePhoneButton.Enabled = false;
            RemovePhoneButton.Enabled = false;
        }

        private void InitializeUserFormData()
        {
            firstNameBox.Text = currentPerson.FirstName;
            lastNameBox.Text = currentPerson.LastName;
            middleNameBox.Text = currentPerson.MiddleName ?? String.Empty;
            birthDate = currentPerson.BirthDate;
```

```
    birthDateBox.Text = birthDate.HasValue ?
        birthDate.Value.ToString("dd/MM/yyyy",
                  CultureInfo.InvariantCulture) :
        "---Empty---";

    foreach (var email in currentPerson.Emails)
    {
        emailsBox.Items.Add(email);
    }

    foreach (var phone in currentPerson.Phones)
    {
        phonesBox.Items.Add(phone);
    }
}

private bool CheckEmpty()
{
    return firstNameBox.Text == String.Empty ||
           lastNameBox.Text == String.Empty;
}

private void OKButton_Click(object sender, EventArgs e)
{
    if (CheckEmpty())
    {
        MessageBox.Show("First name and last name must be filled. ",
                        "Error",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
        return;
    }

    using (var session = ThisApplication.Instance.Session)
    {
        session.BeginUpdate();

        currentPerson = session
            .AllObjects<Person>()
            .FirstOrDefault(x => x.Id == currentPerson.Id);

        currentPerson.FirstName = firstNameBox.Text;
        currentPerson.LastName = lastNameBox.Text;
        currentPerson.MiddleName =
            middleNameBox.Text != String.Empty ?
            middleNameBox.Text : null;
        currentPerson.BirthDate = birthDate;

        currentPerson.ClearPhones();
        foreach (string phone in phonesBox.Items)
        {
            currentPerson.AddPhone(phone);
        }

        currentPerson.ClearEmails();
        foreach (string email in emailsBox.Items)
        {
            currentPerson.AddEmail(email);
        }

        currentPerson.Update();
        session.Commit();
    }
```

```
    Close();
}

private void AddEmailButton_Click(object sender, EventArgs e)
{
    var email = Interaction.InputBox("Enter an email");
    if (email == String.Empty)
    {
        return;
    }

    emailsBox.Items.Add(email);
}

private void UpdateEmailButton_Click(object sender, EventArgs e)
{
    int index = emailsBox.SelectedIndex;
    if (index < 0)
    {
        return;
    }

    var email = Interaction.InputBox(
        Prompt: "Enter an email",
        DefaultResponse: (string)emailsBox.Items[index]);

    emailsBox.Items[index] = email;
}

private void RemoveEmailButton_Click(object sender, EventArgs e)
{
    int index = emailsBox.SelectedIndex;
    if (index < 0)
    {
        return;
    }

    emailsBox.Items.RemoveAt(index);
}

private void CancelButton_Click(object sender, EventArgs e)
{
    Close();
}

private void ChangeBirthDateButton_Click(object sender, EventArgs e)
{
    var window = new UserFormDatePicker(currentPerson.BirthDate);
    birthDate = window.ShowAndChooseDate();
    birthDateBox.Text = birthDate.HasValue ?
        birthDate.Value.ToString("dd/MM/yyyy",
                  CultureInfo.InvariantCulture) :
        "---Empty---";
}

private void AddPhoneButton_Click(object sender, EventArgs e)
{
    var phone = Interaction.InputBox("Enter a phone");
    if (phone == String.Empty)
    {
        return;
    }
```

```
            phonesBox.Items.Add(phone);
        }

        private void UpdatePhoneButton_Click(object sender, EventArgs e)
        {
            int index = phonesBox.SelectedIndex;
            if (index < 0)
            {
                return;
            }

            var phone = Interaction.InputBox(
                Prompt: "Enter a phone ",
                DefaultResponse: (string)phonesBox.Items[index]);

            phonesBox.Items[index] = phone;
        }

        private void RemovePhoneButton_Click(object sender, EventArgs e)
        {
            int index = phonesBox.SelectedIndex;
            if (index < 0)
            {
                return;
            }

            phonesBox.Items.RemoveAt(index);
        }

        private void emailsBox_SelectedIndexChanged(object sender, EventArgs e)
        {
            UpdateEmailButton.Enabled = emailsBox.SelectedIndex >= 0;
            RemoveEmailButton.Enabled = emailsBox.SelectedIndex >= 0;
        }

        private void phonesBox_SelectedIndexChanged(object sender, EventArgs e)
        {
            UpdatePhoneButton.Enabled = phonesBox.SelectedIndex >= 0;
            RemovePhoneButton.Enabled = phonesBox.SelectedIndex >= 0;
        }
    }
}

// -------------------UserFormViewPersonList.cs------------------------------
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace LibraryHelper
{
    public partial class UserFormViewPersonsList : Form
    {
        public enum ViewMode
        {
            All, Employees, Clients
        }
```

```
private List<Tuple<string, ulong>> personsData;
private ViewMode mode;

private bool Suits(Person person)
{
    switch (mode)
    {
        case ViewMode.Employees:
            return person is Employee;
        case ViewMode.Clients:
            return person is Client;
        default:
            return true;
    }
}

public UserFormViewPersonsList(ViewMode mode = ViewMode.All)
{
    this.mode = mode;
    InitializeComponent();
    ReinitializePersonsList();
}

private void ReinitializePersonsList()
{
    personsBox.Items.Clear();
    using (var conn = ThisApplication.Instance.Session)
    {
        conn.BeginRead();
        var query =
            from person in conn.AllObjects<Person>()
            where Suits(person)
            orderby person.FirstName, person.LastName
            select Tuple.Create(person.ToString(), person.Id);

        personsData = query.ToList();

        foreach (var personData in personsData)
        {
            personsBox.Items.Add(personData.Item1);
        }
        conn.Commit();
    }
}

private void ShowEmployeeButton_Click(object sender, EventArgs e)
{
    var index = personsBox.SelectedIndex;
    if (index < 0)
    {
        return;
    }

    var person = Database.GetById<Person>(personsData[index].Item2);
    if (person != null)
    {
        var window = new UserFormViewPerson(person);
        window.ShowDialog();
    }

    ReinitializePersonsList();
}
```

```
        private void DeleteEmployeeButton_Click(object sender, EventArgs e)
        {
            int index = personsBox.SelectedIndex;
            if (index < 0)
            {
                return;
            }

            using (var conn = ThisApplication.Instance.Session)
            {
                conn.BeginUpdate();
                conn.AllObjects<Person>()
                    .FirstOrDefault(x => x.Id == personsData[index].Item2)?
                    .Unpersist(conn);
                conn.Commit();
            }

            ReinitializePersonsList();
        }

        private void personsBox_DoubleClick(object sender, EventArgs e)
        {
            var index = personsBox.SelectedIndex;
            if (index < 0)
            {
                return;
            }

            var person = Database.GetById<Person>(personsData[index].Item2);
            if (person != null)
            {
                var window = new UserFormViewPerson(person);
                window.ShowDialog();
            }

            ReinitializePersonsList();
        }
    }
}

// ---------------------UserFormRegisterEmployee.cs--------------------------
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using VelocityDb.Session;

namespace LibraryHelper
{
    public partial class UserFormRegisterEmployee : Form
    {
        private DateTime? birthDate;
        public UserFormRegisterEmployee()
        {
            InitializeComponent();
            birthDateBox.Text = "---Empty---";
            birthDate = null;
```

```
}

private void OKButton_Click(object sender, EventArgs e)
{
    if (CheckEmpty())
    {
        MessageBox.Show("Error * ",
                        "Error",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
        return;
    }

    if (!PasswordMatch())
    {
        MessageBox.Show("Passwords must match. ",
                        "Error",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
        return;
    }

    if (CheckDuplicateLogin())
    {
        MessageBox.Show("This login already exists. ",
                        "Error",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
        return;
    }


    StoreEmployeeObject();


    Close();
}

private Employee MakeEmployeeObject()
{
    var employee = new Employee
    {
        FirstName = firstNameBox.Text,
        LastName = lastNameBox.Text,
        MiddleName = middleNameBox.Text != String.Empty ?
                     middleNameBox.Text : null,
        BirthDate = birthDate,
        Login = loginBox.Text,
        Password = passwordBox.Text
    };

    if (emailBox.Text != String.Empty)
    {
        employee.AddEmail(emailBox.Text);
    }

    if (phoneBox.Text != String.Empty)
    {
        employee.AddPhone(phoneBox.Text);
    }

    return employee;
}
```

```
        private void StoreEmployeeObject()
        {
            using (var conn = ThisApplication.Instance.Session)
            {
                conn.BeginUpdate();
                var employee = MakeEmployeeObject();
                conn.Persist(employee);
                ThisApplication.Instance.CurrentEmployee = employee;
                conn.Commit();
            }
        }

        private bool CheckDuplicateLogin()
        {
            bool result;
            using (var conn = ThisApplication.Instance.Session)
            {
                conn.BeginRead();
                result = conn
                    .AllObjects<Employee>()
                    .Any(x => x.Login == loginBox.Text);
                conn.Commit();
            }
            return result;
        }

        private bool CheckEmpty()
        {
            return firstNameBox.Text == String.Empty ||
                   lastNameBox.Text == String.Empty ||
                   loginBox.Text == String.Empty ||
                   passwordBox.Text == String.Empty;
        }

        private bool PasswordMatch()
        {
            return passwordBox.Text == password1Box.Text;
        }

        private void CancelButton_Click(object sender, EventArgs e)
        {
            Close();
        }

        private void ChangeBirthDateButton_Click(object sender, EventArgs e)
        {
            var window = new UserFormDatePicker();
            birthDate = window.ShowAndChooseDate();

            birthDateBox.Text = birthDate.HasValue ?
                birthDate.Value.ToString("dd/MM/yyyy",
                            CultureInfo.InvariantCulture) :
                "---Empty---";
        }
    }
}

// -------------------------UserFormLogin.cs-------------------------------
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
```

```
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace LibraryHelper
{
    public partial class UserFormLogin : Form
    {
        private Employee logined;

        public UserFormLogin()
        {
            InitializeComponent();

        }

        private void OKButton_Click(object sender, EventArgs e)
        {
            using (var conn = ThisApplication.Instance.Session)
            {
                conn.BeginRead();
                logined = conn.AllObjects<Employee>().FirstOrDefault(x =>
                {
                    return x.Login == loginBox.Text &&
                           x.Password == passwordBox.Text;
                });
                conn.Commit();
            }

            if (logined == null)
            {
                MessageBox.Show("Access denied. ",
                                "Error",
                                MessageBoxButtons.OK,
                                MessageBoxIcon.Error);
                return;
            }

            Close();
        }

        public Employee ShowAndLogin()
        {
            ShowDialog();
            var screenSize = Screen.PrimaryScreen.WorkingArea.Size;
            Location = new Point
            {
                X = screenSize.Width / 2 - Width / 2,
                Y = screenSize.Height / 2 - Height / 2
            };

            return logined;
        }

        private void CancelButton_Click(object sender, EventArgs e)
        {
            logined = ThisApplication.Instance.CurrentEmployee;
            Close();
        }

        private void RegisterButton_Click(object sender, EventArgs e)
```

```
        {
            Hide();
            var window = new UserFormRegisterEmployee();
            window.ShowDialog();

            if (ThisApplication.Instance.CurrentEmployee == null)
            {
                Show();
                return;
            }

            logined = ThisApplication.Instance.CurrentEmployee;
            Close();
        }
    }
}

// ----------------------UserFormViewDebtors.cs--------------------------
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace LibraryHelper
{
    public partial class UserFormViewDebtors : Form
    {
        private List<DebtorInfo> info;
        private class DebtorInfo
        {
            public ulong DebtorId { get; set; }
            public ulong BookId { get; set; }
            public DateTime StartDate { get; set; }
            public string BookName { get; set; }
            public string DebtorName { get; set; }
            public string EmployeeName { get; set; }
        };

        public UserFormViewDebtors()
        {
            InitializeComponent();
            using (var conn = ThisApplication.Instance.Session)
            {
                conn.BeginRead();
                IEnumerable<DebtorInfo> query =
                    from book in conn.AllObjects<Book>()
                    where book.CurrentOwner != null
                    //where book.Durability.Value > new TimeSpan(90, 0, 0, 0, 0)
                    select new DebtorInfo
                    {
                        DebtorId = book.CurrentOwner.Id,
                        BookId = book.Id,
                        StartDate = book.Owners.Last().StartDate,
                        BookName = book.Name,
                        DebtorName = book.CurrentOwner.ToString(),
                        EmployeeName = book.Owners.Last().Employee.ToString() ??
                                       "---Empty---"
                    };
```

```
                query = query.ToList();

                info = (List<DebtorInfo>)query;
                conn.Commit();

                foreach (var elem in info)
                {
                    infoBox.Items.Add(new ListViewItem(new string[]
                    {
                        elem.BookName,
                        elem.DebtorName,
                        elem.StartDate.ToString("dd/MM/yyyy"),
                        elem.EmployeeName
                    }));
                }
            }
        }

        private void ViewClientButton_Click(object sender, EventArgs e)
        {
            try
            {
                var index = infoBox.SelectedIndices[0];
                if (index < 0)
                {
                    return;
                }

                var person = Database.GetById<Person>(info[index].DebtorId);
                var window = new UserFormViewPerson(person);
                window.ShowDialog();
            }
            catch(ArgumentOutOfRangeException)
            {
                return;
            }
        }

        private void BookInfoButton_Click(object sender, EventArgs e)
        {
            try
            {
                var index = infoBox.SelectedIndices[0];
                if (index < 0)
                {
                    return;
                }

                var book = Database.GetById<Book>(info[index].BookId);
                var window = new UserFormViewBook(book);
                window.ShowDialog();
            }
            catch(ArgumentOutOfRangeException)
            {
                return;
            }
        }
    }
}
```