Довідка
про відсутність плагіату у випускній кваліфікаційній роботі

Міністерство освіти і науки України
Український державний університет науки та технологій

Кафедра «Комп'ютерні інформаційні технології»

ДОВІДКА

За результатами перевірки випускної кваліфікаційної роботи здобувача вищої освіти Рижкової Анастасії Андріївни
(прізвище, ім’я, по батькові)
на тему: Аналіз можливостей та пристосування методів рефакторингу до запитів на мові SQL в роботі не виявлено порушень академічної доброчесності.

Керівник ВКР Іванов О.П. _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Український державний університет науки і технологій

Кафедра Комп'ютерні інформаційні технології

«ДО ЗАХИСТУ»

Завідувач кафедри
_____ /Вадим ГОРЯЧКІН/
«_17_» _____12_____ 20_21_ р.

ДИПЛОМНА РОБОТА
на здобуття освітнього ступеня «магістр»

Галузь знань **12 Інформаційні технології**

Спеціальність **121 Інженерія програмного забезпечення**

Тема **Аналіз можливостей та пристосування методів рефакторингу до запитів на мові SQL**

Theme **Analysis of capabilities and adaptation of refactoring methods to queries in SQL language**

Керівник дипломної роботи доц. _____ Олександр ІВАНОВ

Нормоконтролер доц. _____ Олена КУРОП'ЯТНИК

Студентка групи ПЗ2021 _____ Анастасія РИЖКОВА

Student Anastasiia RYZHKOVA

Дніпро – 2021

Дніпровський національний університет залізничного транспорту
імені академіка В. Лазаряна

Факультет Технічна кібернетика кафедра Комп'ютерні інформаційні технології
Спеціальність Інженерія програмного забезпечення

«ЗАТВЕРДЖУЮ»
Завідувач кафедри
_____ проф. Шинкаренко В. І.
(підпис)
«___» _____ 2021 р.

ЗАВДАННЯ

до дипломної роботи на здобуття ОС **магістр**

студентки групи **ПЗ2021 Рижкова Анастасія Андріївна**

1 Тема дипломної роботи: **Аналіз можливостей та пристосування методів рефакторингу до запитів на мові SQL**

затверджена наказом по університету від «**18**» **листопада 2020** р. № **690**.

2 Термін подання студентом закінченого проекту _____

3 Вихідні дані до дипломного проекту _____
_____
_____
_____
_____

4 Зміст пояснювальної записки (перелік питань до розробки) складається з: вступу, аналізу сучасного стану рефакторингу, результатів створених методів рефакторингу, розробки інструменту розрахунку за ARI, оцінки методів рефакторингу SQL запитів, охорона праці та безпека в надзвичайних ситуаціях, загальних висновків та бібліографічного списку _____
_____
_____

5 Перелік демонстраційного матеріалу: пояснювальна записка, технічне завдання, робочий проект, тези конференцій, стаття підготовлена до друку, презентація по дипломний роботі_____
_____
_____

6. Консультанти (з назвами розділів):

Розділ	Консультант	Підпис, дата	
		завдання видав	завдання прийняв
Техніко-економічні розрахунки	доцент Гненний М. В.	13.09.21	19.09.21
Охорона праці	доцент Саблін О. І.	12.10.21	13.10.21

КАЛЕНДАРНИЙ ПЛАН

№ пор.	Назва розділів дипломної роботи	Термін виконання розділів проекту (роботи)	Примітка
1	Вступ	02.08.2021 - 15.08.2021	
2	Аналіз сучасного стану дослідження проблеми за науковими літературними джерелами	16.08.2021 - 29.08.2021	
3	Аналіз сучасного стану програмно-апаратного забезпечення, яке потребує вдосконалення для вирішення проблем дослідження	30.08.2021 - 05.09.2021	
4	Постановка задачі, технічне завдання	06.09.2021 - 12.09.2021	
5	Техніко-економічні показники	13.09.2021 - 19.09.2021	
6	Розробка інструментальних засобів дослідження	20.09.2021 - 17.10.2021	30%
7	Виконання досліджень	18.10.2021 - 31.10.2021	
8	Оформлення тез доповідей	01.11.2021 - 07.12.2021	
9	Оформлення статті у фаховий журнал	08.11.2021 - 14.11.2021	60%
10	Оформлення пояснювальної записки	15.11.2021 - 28.11.2021	
11	Розробка демонстраційних матеріалів	29.11.2021 - 05.12.2021	100%

Дата видачі завдання «18» листопада 2020 р.

Керівник дипломної роботи _____ О.П. Іванов

(підпис) (ПІБ)

Завдання прийняв до виконання _____ А.А Рижкова

(підпис) (ПІБ)

РЕФЕРАТ

Об’єкт дослідження: адаптація методів рефакторингу для мови SQL, перевірка доцільності використання методів за метрикою читабельності ARI.

Предметом дослідження є порівняння отриманих результатів під час розрахунків метрики читабельності ARI для запитів до та після рефакторингу.

Метою дослідження є визначення доцільності використання методів рефакторингу для мови SQL.

Методи дослідження: адаптація методів рефакторингу для ООП мов до декларативної мови SQL. Порівняння результатів розрахунків для метрики читабельності ARI.

Результати дослідження дозволяють розширити методи рефакторингу для мови SQL та бути впевненим щодо доцільності їх використання.

Пояснювальна записка складається зі вступу, п’яти розділів, висновків, бібліографічного списку та 4 додатків:

- вступ описує актуальність та новизну обраної теми, складається із 2 сторінок;
- перший розділ складається з 7 сторінок і описує сучасний стан технологій;
- другий розділ складається з 20 сторінок і містить обґрунтування експериментального методу дослідження;
- третій розділ складається з 16 сторінок, описує архітектуру системи;
- четвертий розділ складається з 19 сторінок де описані експерименти;
- п’ятий розділ розкриває питання охорони праці, складається з 11 сторінок;
- додатки містять технічне завдання, робочий проект, статтю підготовлену до друку, тези на: 81 Всеукраїнську науково-технічну конференцію молодих учених, магістрантів та студентів «Наука і сталий розвиток транспорту», Всеукраїнську науково-технічну конференцію молодих учених, магістрантів та студентів «Науково-технічний прогрес на транспорті».

Усього робота містить: таблиць – 23, рисунків – 9, бібліографія – 52 джерела.

Ключові слова: Рефакторинг, SQL, ARI, читабельність.

ЗМІСТ

Вступ 10
1 Аналіз сучасного стану рефакторингу 13
1.1 Задачі рефакторингу 13
1.2 Аналіз метрик читабельності 14
1.2.1 Coleman-Liau Index 14
1.2.2 SMOG 15
1.2.3 Flesch-Kincaid Readability Index 15
1.2.4 The Gunning’s Fog Index 16
1.2.5 The Automated Readability Index 16
1.3 Огляд сучасного стану літературних джерел 17
1.4 Огляд сучасного стану програмних продуктів 17
Висновки до першого розділу 18
2 Результати створених методів рефакторингу 20
2.1 Структура опису методів рефакторингу запитів 20
2.2 Розроблені методи рефакторингу 22
2.2.1 Складання запитів 22
2.2.1.1 Перейменування псевдонімів та змінних 22
2.2.1.2 Підстановка розширенням 23
2.2.1.3 Перелічення параметрів для вставки 24
2.2.1.4 Уникнення сортування даних 25
2.2.2 Уточнення даних 25
2.2.2.1 Уточнення імені об’єкту 25
2.2.2.2 Додавання або видалення квадратних дужок для ідентифікаторів 27
2.2.2.3 Додавання точки з комою 27
2.2.2.4 Додавання або видалення псевдоніму (AS) 29
2.2.2.5 Надання оператору повного ім’я 30
2.2.3 Спрощення складних запитів 31
2.2.3.1 Виділення частин запиту у common table expression (CTE) 31
2.2.3.2 Заміна алгоритму об’єднання таблиць 32

2.2.4 Спрощення умовних виразів 33
2.2.4.1 Перетворення умови на in вираз 33
2.2.4.2 Використання WHERE до/замість HAVING 34
2.2.4.3 Заміна LIKE на оператор рівності 35
2.2.4.4 Використання EXIST або JOIN замість IN 36
2.2.5 Робота з коментарями 37
2.2.5.1 Додавання або видалення коментарів 37
Висновки до другого розділу 39
3 Розробка інструменту розрахунку за ARI 40
3.1 Формалізація задачі 40
3.2 Архітектура програмної системи 40
3.3 Внутрішнє проектування 41
3.3.1 Вибір мови програмування 41
3.3.2 Технологічна платформа 42
3.3.3 Ієрархія та взаємодія класів системи 43
3.3.4 Використані принципи проектування 44
3.4 Розробка інтерфейсу користувача 45
3.5 Тестування та налагодження програми 48
3.5.1 Аналіз методів тестування та відлагодження 48
3.5.2 Проектування тестів за допомогою методу еквівалентних розбитів 49
3.5.3 Тестування системи за допомогою модульного тестування 51
3.5.4 Тестування системи методом припущення про похибку 52
Висновки до третього розділу 54
4 Оцінка методів рефакторингу SQL запитів 56
4.1 Складання запитів 56
4.1.1 Перейменування псевдонімів та змінних 56
4.1.2 Підстановка розширенням 58
4.1.3 Перелічення параметрів для вставки 59
4.1.4 Уникайте сортування даних 59
4.2 Уточнення даних 60

4.2.1 Уточнення імені об’єкту 60
4.2.2 Додавання або видалення квадратних дужок для ідентифікаторів 61
4.2.3 Додавання точки з комою 62
4.2.4 Додавання псевдоніму 63
4.2.5 Надання оператору повного імені 65
4.3 Спрощення складних запитів 66
4.3.1 Виділити як common table expression (CTE) 66
4.3.2 Заміна алгоритму об’єднання таблиць 67
4.4 Спрощення умовних виразів 68
4.4.1 Використання WHERE замість HAVING 68
4.4.2 Заміна LIKE на оператор рівності 69
4.4.3 Перетворення умови на in вираз 70
4.4.4 Використання EXIST або JOIN замість IN 71
4.5 Робота з коментарями 72
4.5.1 Додавання або видалення коментарів 72
Висновки до четвертого розділу 74
5 Охорона праці та безпека в надзвичайних ситуаціях 75
5.1 Вимоги безпеки при виконанні розрахункових робіт на робочому місці за персональним комп`ютером 75
5.1.1 Шкідливі фактори на робочому місці 76
5.1.2 Характеристики робочого місця 77
5.1.3 Освітлення 78
5.1.4 Мікроклімат 80
5.1.5 Шум та вібрація 81
5.2 Дії працівників у надзвичайних ситуаціях 81
Висновки до п’ятого розділу 83
Загальні висновки 84
Бібліографічний список 86
Технічне завдання та робочий проект
Тези конференцій

Стаття підготовлена до друку

ПЕРЕЛІК УМОВНИХ ПОЗНАК, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ARI – (англ. automated readability index) автоматичний індекс читабельності.

CTE – (англ. common table expression) загальний табличний вираз.

SQL – (англ. structured query language) мова структурованих запитів.

БД – база даних.

Рефакторинг - це процес зміни програмного проекту, у ході якого зовнішня поведінка коду залишається незмінною під час удосконалення його внутрішньої структури.

СКБД – системи керування базами даних.

ВСТУП

Інформаційні технології – це важлива складова повсякденного життя, тому наразі створюється та вдосконалюються все більше і більше програмних продуктів. Велика кількість спеціалістів з інформаційних технологій залучається до проектів, які тільки почали розроблюватися або вже існують багато років та потребують нарощування функціоналу або підтримки. Для ефективної роботи спеціалістів необхідно підтримувати тексти програм у зрозумілому людині вигляді, адже здебільшого текст програми не пишеться, а читається. Для покращення читабельності та зрозумілості тексту програми необхідно застосовувати рефакторинг коду. Набагато легше читати текст програми на якому був проведений рефакторинг, тому розробники проводять даний етап регулярно по завершенню завдання або по гострій необхідності, як окрему задачу на проекті. Сам процес рефакторингу призваний для удосконалення тексту програми без зміни функціональності продукту. Методи рефакторингу використовуються для очищення тексту програми та мінімізації можливостей появи нових помилок.

Актуальність роботи. Наразі в багатьох проектах використовуються реляційні бази даних, де за допомогою СКБД та мови запитів SQL можна маніпулювати даними (вставляти, оновлювати, видаляти та обирати). Так як запити є частиною загального тексту програми є потреба у методах рефакторингу для мови запитів, але наразі даний підхід мало розкритий у спеціалістів інформаційних технологій.

Мета. Метою дослідження є проведення аналізу доцільності пристосування методів рефакторингу для мови запитів SQL.

Об’єкт і предмет дослідження. Об’єктом та предметом дослідження є адаптація методів рефакторингу ООП мов для мови SQL, що вирішить проблему читабельності та зрозумілості тексту запитів. Досліджується різниця результатів метрики читабельності тексту отриманих під час обробки запитів методами рефакторингу.

Завдання. В процесі розробки програми і написання документації необхідно було вирішити наступне:

- адаптувати та навести власні методи рефакторингу для мови запитів SQL, навести приклади використання;
- розробити програмний продукт, що розраховує метрику читабельності тексту для оцінки запропонованих методів рефакторингу;
- проаналізувати отримані результати порівняння запитів «до» та «після» рефакторингу та на їх основі розробити рекомендації щодо доцільності їх використання.

Методи. Можна виділити наступні методи дослідження:

1. Адаптація методів для ООП мов програмування, які наразі мають широке застосування для мови запитів SQL.
2. Порівняння запитів «до» та «після» використання методів рефакторингу та аналіз за метрикою читабельності ARI.

Наукова новизна. Результати дослідження дозволяють зробити висновки щодо доцільності впровадження методів рефакторингу для мови SQL. Наразі велика кількість робіт вже присвячена вдосконаленню архітектури баз даних та форматуванню запитів. Також вже існують продукти, які починають розвивати тему рефакторингу мови запитів SQL, але дана робота направлена на додатковий внесок у розробку методів рефакторингу та їх аналіз, які можуть бути застосовані для будь-яких СКБД, які використовують SQL.

Практична значимість. Практичне значення одержаних результатів полягає у використанні методів рефакторингу, які розроблені таким чином, щоб не змінювати схему баз даних та користуватися загальним стандартом мови SQL. Даний підхід надасть змогу користуватися методами з різними СКБД, що робить їх універсальними.

Апробація результатів дослідження та публікації. Основні положення магістерської роботи доповідалися та були схвалені на наукових семінарах кафедри КІТ (01.03.2021 р., 09.12.2021 р.).

Публікації за темою роботи. За результатами роботи опубліковано наукові праці – тези доповідей на наукових конференціях:

— Всеукраїнська науково-технічна конференція молодих учених, магістрантів та студентів «Науково-технічний прогрес на транспорті» від 29 березня 2021 року.

— 81 Всеукраїнська науково-технічна конференція молодих учених, магістрантів та студентів «Наука і сталий розвиток транспорту» від 28 жовтня 2021 року.

Підготовлено наукову статтю «Рефакторинг SQL запитів».

1 АНАЛІЗ СУЧАСНОГО СТАНУ РЕФАКТОРИНГУ

1.1 Задачі рефакторингу

Рефакторинг вже давно є одним із складових розробки програмного продукту, який зазвичай пристосовується до задачі перед додаванням нового функціоналу до проекту. У разі, якщо довго не робити рефакторинг тексту програми, з часом продукт стає все більш заплутаним. Тому чим довше відтягується етап рефакторингу на проекті, тим довше та дорожче буде в подальшому проводити цей процес.

Методи рефакторингу розроблювалися спеціалістами інформаційних технологій, що здебільшого при розробці методів спиралися на свій досвід програмування. Для ООП мов було розроблено більше 70 методів рефакторингу, які користуються популярністю серед інженерів-програмістів [27].

Рефакторинг треба проводити, якщо автору або його колегам не подобається, як саме вирішена поставлена задача та точно таке рішення ніхто повторно б не відтворював. Існує «правило трьох», яке говорить наступне [39]:

- роблячи задачу в перший раз – робите, як вважаєте за потрібне;
- робите таку ж задачу вдруге, тим способом, що робили в перший раз. В результаті рішення не подобається, але залишаєте зроблене;
- роблячи втретє – починайте рефакторинг.

Виділяються три випадки, коли краще провести рефакторинг:

- роблячи задачу, яка є новою для проекту. Одразу після виконання проаналізуйте написаний код та пристосуйте необхідні методи рефакторингу до нього;
- виправляючи помилки програми. При проведенні рефакторингу можна знайти місце у якому була помилка та виправити її, поліпшити код;
- аналізуючи текст програми самостійно або з колегами під час промотору тексту програми перед його публікацією. Таку процедуру краще проводити з автором коду, щоб дати йому поради для покращення його праці.

Щоб правильно провести рефакторинг коду, треба потроху змінювати текст програми не додаючи новий та не видаляючи існуючий функціонал, дотримуючись проходження тестів. Вдало проведений рефакторинг складається з наступних пунктів:

– текст програми став чистішим;
– функціонал залишився таким, який він був до проведення рефакторингу;
– всі авто-тести успішно проходяться;

Рано чи пізно доведеться проводити рефакторинг коду, так як прогрес у сучасному світі не стоїть на місці. З'являються нові мови програмування, бібліотеки, фреймворки, тощо, які приходять на заміну старим технологіям. Щоб підтримувати довге життя продукту треба оновлювати його відповідно до нових стандартів.

1.2 Аналіз метрик читабельності

Для аналізу чи буде метод рефакторингу для мови SQL доцільним чи ні пропонується оцінювати даний експеримент за допомогою метрик читаємості тексту. Наразі немає метрики, щоб могла оцінити читаємість тексту мови SQL, але так як мова SQL є наближеною до людської природньої мови, то можна скористатися метриками винайденими для природніх мов [49].

За результатами пошуку були знайдені наступні метрики читабельності текстів для природніх мов:

— Coleman-Liau Index;
— SMOG;
— Flesch-Kincaid Readability Index;
— The Gunning's Fog Index;
— The Automated Readability Index.

1.2.1 Coleman-Liau Index

Coleman-Liau Index – метрика читабельності тексту, яка схожа на The Automated Readability Index (ARI). Дана метрика апроксимує складність тексту до номеру класу у американській системі навчання. Формула для розрахунків є натупною:

$$CLI = 0.0588L - 0.296S - 15.8,$$

де L – середне число кількості літер на 100 слів;

S – середня кількість речень на 100 слів.

Індекс рівний одиниці може добре сприйматися дітьми 6-7 років, що відповідно є першокласниками. Така метрика може легко розраховуватися за допомогою комп'ютеру, бо не потребує складних вхідних даних [29, 40].

1.2.2 SMOG

Метрика SMOG – це акронім для «Simple Measure of Gobbledygook», що в перекладі – «Проста міра Гобледигука». Метрика являє собою міру читабельності, що оцінює роки освіти, необхідні для розуміння тексту. Метрика показує точніші результати за інші. Формула для даної метрики виглядає наступним чином [29, 51]:

$$SMOG\ grade = 1.0430\sqrt{PS * \frac{30}{N}} + 3.1291,$$

де PS – кількість слів, у яких більше трьох складів;

N – кількість речень.

Розглядаєма метрика є складною для автоматизації, бо у формулі існує залежність від кількості складів.

1.2.3 Flesch-Kincaid Readability Index

Flesch-Kincaid Readability Index – метрика, що перевіряє легкість прочитання тексту для англійської мови. Формула наведена нижче [43]:

$$FKRI = 206.835 - 1.015 * ASL - 84.6\ AWS,$$

де ASL – середня довжина речення у словах;

AWS – середня довжина слова по складах.

Така метрика має шкалу значень наведену у таблиці 1.2.3:

Таблиця 1.2.3 – Шкала результатів для Flesch-Kincaid Readability Index

Значення	Рівень тексту
1	2
90-100	Дуже простий
80-89	Простий
70-79	Не дуже простий

Продовження таблиці 1.2.3

1	2
60-69	Звичайний
50-59	Не дуже складний
30-49	Складний
0-29	Дуже складний

Метрика є не дуже легкою у автоматизації, тому що присутній зв’язок зі складами слів.

1.2.4 The Gunning’s Fog Index

The Gunning’s Fog Index – метрика для перевірки читабельності для англійського тексту. Метрика апроксимована під роки навчання у навчальних закладах для людини, що бачить текст перший раз. Тексти для читання широкій аудиторії потребують значення менше 12 балів. Для розрахунку використовується наступна формула [29, 42]:

$$FOG = 0.4\ (ASL + PHW),$$

де ASL – середня довжина речення у словах;

PHW – процент складних слів у тексті. Складними словами вважаються такі, що мають три або більше склади.

Метрика є не дуже легкою у автоматизації, тому що присутній зв’язок зі складами слів.

1.2.5 The Automated Readability Index

The Automated Readability Index – метрика для оцінки читабельності тексту, що апроксимована до номеру класу у американській системі навчання. Формула метрики наведена нижче [29, 41]:

$$ARI = 4.71 * \frac{C}{W} + 0.5 * \frac{W}{S} - 21.43,$$

де C – кількість літер та цифр у тексті,

W – кількість слів у тексті,

S – кількість речень у тексті.

У порівнянні з іншими метриками, дана основана на кількості літер, а не складів слів. Дана метрика була розроблена для автоматизації на електроно-обчислювальних машинах, тому є легкою для реалізації.

Метрика Automated Readability Index є найкращою для програмної реалізації, тому була обрана для оцінки доцільності впровадження методів рефакторингу для мови SQL.

1.3 Огляд сучасного стану літературних джерел

Методи рефакторингу були описані в серії книг, призначених для застосування у імперативних мовах програмування. Так як мова запитів має іншу парадигму, то ці методи рефакторингу не можуть бути застосовані безпосередньо, а потребують адаптацію та переробку під декларативне програмування.

Для адаптації методів рефакторингу функціональної мови на мову запитів SQL була обрана книга, яка є однією з найпопулярніших на сьогоднішній день – "Рефакторинг. Улучшение проекта существующего кода", автора Мартіна Фаулера [27]. Автор пропонує понад 20 прикладів поганої практики програмування для ООП мов та понад 70 методів рефакторингу.

Дана книга розглянута у якості прикладу для даної роботи. Завдяки цьому джерелу була проведена адаптація методів рефакторингу для ООП мов до мови SQL. Також була наведена схожа структура подання методів до огляду іншими спеціалістами.

Наразі літературних джерел за тематикою рефакторинга SQL запитів виявлено не було.

1.4 Огляд сучасного стану програмних продуктів

На ринку вже існують програмні застосункі, які автоматизують методи рефакторингу насписані для мови T-SQL. Методи, що пропонуються здебільшого впливають на схему бази даних здебільшого через засоби пов’язані з СКБД та її архітектуру, яка покращується за рахунок проведення нормалізації.

Було знайдено наступні програмні продукти, що є популярними:

— ApexSQL [45];

— SQL Promt [44].

Обидва продукти позиціонують себе як інструменти для форматування запитів, але також представляють можливість робити рефакторинг. Так як методи продуктів повторюються, то список методів рефакторингу буде наведено для обох продуктів разом. Рефакторинг складається з наступних методів:

— інкапсуляція: збережена процедура;

— уточнююча назва об’єкту;

— безпечне перейменування;

— пошук невикористаних параметрів та змінних;

— розділення таблиці;

— підстановка розширенням;

— перейменування змінних та псевдонімів;

— додавання сурогатного ключа;

— зміна параметрів;

— інкапсуляція: скалярна вбудована функція;

— інкапсуляція: таблична вбудована функція;

— інкапсуляція: представлення;

— зміна зв’язку на «один до багатьох»;

— додавання або видалення додаткового ключового слова AS;

— додавання або видалення квадратних дужок ідентифікаторів;

— вставка крапки з комою.

До методів рефакторингу, які не змінюють схему бази даних та можуть бути застосовані для всіх СКБД можна віднести: уточнююча назва об’єкту, підстановка розширенням, перейменування змінних та псевдонімів, додавання або видалення додаткового ключового слова AS, додавання або видалення квадратних дужок ідентифікаторів, вставка крапки з комою. Шість методів з шістнадцяти можуть бути застосовані у будь-якій СКБД.

Висновки до першого розділу

У результаті дослідження сучасного стану проблеми були описані задачі, які ставить перед собою процес рефакторингу, обрана метрика, яка допоможе оцінити

доцільність розроблених методів рефакторинга для мови SQL, розглянуті літературні джерела та програмні продукти.

Метрика, яка була обрана є швидкою для розрахування та легка у реалізації програмними засобами, тому що у формулі метрики містяться лише кількісні показники.

За літературними джерелами був проведений аналіз подачі матеріалів пов'язаних з методами рефакторингу. Загальна структура взята до уваги та запропоновані методи рефакторингу будуть описуватися за цим прикладом, що добре структурує інформацію для прочитання спеціалістами.

Оглянуті програмні застосунки наразі більше займаються рефакторингом архітектури бази даних, ніж запитів, але вже є методи, які можна уніфікувати для всіх СКБД. Так як методи, що можуть використовуватися з будь якою СКБД не мають доброї документації та не проаналізовані метрикою читабельності, також беруться до уваги у досліджені та розглядаються, як потенційні для використання всіма спеціалістами, хто користується SQL.

2 РЕЗУЛЬТАТИ СТВОРЕНИХ МЕТОДІВ РЕФАКТОРИНГУ

Для розробки нових методів використана книга «Рефакторинг. Улучшение проекта существующего кода»[27], щоб адаптувати вже існуючі методи рефакторингу для об’єктно орієнтовних мов програмування на методи, які стануть до нагоди у мові SQL. Мартін Фаулер пропонує понад 20 прикладів поганої практики програмування для ООП мов і понад 70 методів рефакторинга. Робота побудована наступним чином: розглядається метод для функціональної мови програмування, передбачається метод для мови запитів, описується, описується його мотивація, техніка виконання та пишуться приклади "до" та "після" застосованого методу.

Для прикладу методу адаптації можна розглянути метод «Перетворення умови на IN вираз», який буде детальніше описано нижче. Даний метод був адаптований для SQL мови з методу рефакторингу «Консолідація умовного виразу» для ООП мови. Останній метод пропонує замінити ряд умовних виразів одним складним або ж винести його, як окремий метод. Для нового методу рефакторингу SQL запиту пропонується об’єднати складну умову, яка перевіряє вміст ряду значень в одному з рядків обраної таблиці, IN виразом.

Також дослідити саму мову та знайти такі мовні конструкції, які зможуть замінювати один одного задля спрощення читання та розуміння тексту запиту.

Для прикладу, методом, який пропонує заміну на більш вдалу мовну конструкцію, є «Уточнення імені об’єкту». Цей метод пропонує доповнити імена об’єктів їх джерелом, наприклад, для стовпця уточнити ім’я таблиці з якої він взятий.

2.1 Структура опису методів рефакторингу запитів

Створені методи, щодо покращення тексту запитів, будуть мати наступну структуру:

1. Назва метода.

2. Адаптація з методу – назва методу рефакторингу для ООП мови, який був адаптований під розглядаємий метод (необов’язково).

3. Опис методу, що розглядається – уточнюється, що конкретно буде змінюватися та за яких можливих умов.

4. Мотивація до застосування – описується чому спеціаліст повинен застосувати даний метод для свого запиту.

5. Приклади рефакторинга до застосування методу та після – уточнюються мовні конструкції, які будуть замінені на більш вдалі та наводиться приклад виправлення.

Методи рефакторингу умовно можна розділити на наступні групи:

1. Складання запитів:
    a. перейменування псевдонімів та змінних;
    b. підстановка розширенням;
    c. перелічення параметрів для вставки;
    d. уникнення сортування даних.
2. Уточнення даних:
    a. уточнення імені об’єкту;
    b. додавання або видалення квадратних дужок для ідентифікаторів;
    c. додавання точки з комою;
    d. додавання або видалення псевдоніму (AS);
    e. надання оператору повного імені.
3. Спрощення складних запитів:
    a. виділення частин запиту у Common Table Expression (CTE);
    b. заміна алгоритму об’єднання таблиць.
4. Спрощення умовних виразів:
    a. використання WHERE до/замість HAVING;
    b. заміна LIKE на оператор рівності;
    c. перетворення умови на IN вираз;
    d. використання EXISTS або JOIN замість IN.
5. Робота з коментарями:
    a. додавання або видалення коментарів.

2.2 Розроблені методи рефакторингу

2.2.1 Складання запитів

2.2.1.1 Перейменування псевдонімів та змінних

Адаптовано з методу: «Перейменування методу»

Опис**:** метод дозволяє перейменовувати змінні або псевдоніми без порушення залежностей.

Мотивація**:** щоб чітко розуміти з якою сутністю або даними розробнику доведеться працювати або отримати на виході, об’єкт повинен бути названий відповідно до того що він означає в даному контексті. Для покращення розуміння перейменуйте об’єкт на більш зрозумілу назву.

Приклад «до застосування методу»: не інформативні псевдоніми.

```
SELECT Table.name, AVG(Table.Count)
FROM(
    SELECT project.name, project_employee.employee_position, COUNT(task.id) AS Count
    FROM project_employee
    JOIN project ON project.id = project_employee.project_id
    JOIN task ON task.project_employee_id = project_employee.id
    GROUP BY project_employee.employee_position, project.name) AS Table
GROUP BY Table.name;
```

Приклад «після застосування методу»: псевдоніми, які описують відповідні дані, відображають сутність об’єкту.

```
SELECT tasks_number_per_employee.name, AVG(tasks_number_per_employee, tasks_number)
FROM(
```

SELECT project.name, project_employee.employee_position, COUNT(task.id) as **tasks_number**

FROM project_employee

JOIN project ON project.id = project_employee.project_id

JOIN task ON task.project_employee_id = project_employee.id

GROUP BY project_employee.employee_position, project.name) AS **tasks_number_per_employee**

GROUP BY **tasks_number_per_employee**.name;

2.2.1.2 Підстановка розширенням

Адаптовано з методу: «Додавання параметру», «Видалення параметру»

Опис**:** не використовуйте знак «*» (підстановку розширенням) в операторі SELECT.

Мотивація: не повертайте з запиту непотрібні стовбці, це погіршує розуміння запиту, його специфікацію. Не завжди очевидно, що повертає запит. Повернення непотрібних рядків і стовпців таблиці витрачає зайвий час процесора, пам'ять, диск і пропускну здатність мережі.

Приклад «до застосування методу»: не замінена підстановка розширенням.

SELECT *****

FROM project_employee

JOIN task ON task.project_employee_id = project_employee.id

JOIN project ON project_employee.project_id = project.id

JOIN employee ON employee_id = employee.id

Приклад «після застосування методу»: псевдоніми, які описують відповідні дані, відображають сутність об’єкту.

SELECT **employee.lastname, task.title, project.name**

FROM project_employee

JOIN task ON task.project_employee_id = project_employee.id
JOIN project ON project_employee.project_id = project.id
JOIN employee ON employee_id = employee.id

2.2.1.3 Перелічення параметрів для вставки

Адаптовано з методу: «Додавання параметру», «Видалення параметру»

Опис**:** при використанні оператору INSERT перелічуйте параметри для вставки.

Мотивація: перелічите параметри для уточнення стовбців для вставки даних та їх порядку. Це не тільки зробить запит більш зрозумілим для вставки нових даних, але й при зміні об’єктів таблиці буде легше виправляти дані для вставки у випадку, якщо компілятор не зможе виконати запит.

Приклад «до застосування методу»: не уточнені параметри для вставки.

INSERT task
VALUES
(1, 'Create project architecture','ASP.NET project + Angular', 4, 6),
(2, 'Create calendar matrix','ASP.NET project + Angular', 1, 1),
(3, 'Test plan','Write test plan', 4, 4);

Приклад «після застосування методу»: зрозумілий порядок для вставки даних та уточнені стовбці.

INSERT task**(id, title, description, task_status_id, project_employee_id)**
VALUES
(1, 'Create project architecture','ASP.NET project + Angular', 4, 6),
(2, 'Create calendar matrix','ASP.NET project + Angular', 1, 1),
(3, 'Test plan','Write test plan', 4, 4);

2.2.1.4 Уникнення сортування даних

Опис: не використовуйте сортування даних ORDER BY, якщо це не потрібно в даному контексті.

Мотивація: зайві операції роблять текст запиту лише більш заплутаним, тому без потреби не рекомендується вживати непотрібні заходи. Сортування коштує дорого. Якщо вам потрібно сортувати, переконайтеся, що запит не сортує зайві дані.

Приклад «до застосування методу»: сортування даних в даному випадку не потрібне, можна замінити іншим оператором.

```
SELECT TOP(1) task_status.id
FROM task_status
JOIN task ON task.task_status_id = task_status.id
JOIN project_employee ON project_employee.id = task.project_employee_id
WHERE project.id = project_employee.project_id
ORDER BY task.id
```

Приклад «після застосування методу»: сортування даних замінено іншим оператором.

```
SELECT MIN(task_status.id)
FROM task_status
JOIN task ON task.task_status_id = task_status.id
JOIN project_employee ON project_employee.id = task.project_employee_id
WHERE project.id = project_employee.project_id
```

2.2.2 Уточнення даних

2.2.2.1 Уточнення імені об’єкту

Опис: функція уточнення імені об'єкта дозволяє спеціалістам змінити запити SQL, щоб всі імена об'єктів були уточнені. Можна уточнити імена об'єктів, щоб додати:

– власника об'єктів для кожного об'єкта, зазначеного в коді SQL в форматі owner.object;

– від імені об'єкта (таблиці, імені уявлення і т. п.) до імен стовпців в форматі table.column;

– ім'я псевдоніма для підзапиту для імен стовпців в форматі alias.column.

Обов’язково використовувати уточнення для об’єктів, які мають однакову назву в декількох об’єктах запиту.

Мотивація: розробник точно буде знати до якого власника/об’єкта/псевдоніма відноситься ідентифікатор. Це підвищує читабельність та уникає протиріч, якщо деякі об’єкти названі однаково, або майже однаково у різних джерелах. Також SQL сервер не буде перевіряти, чи є поточний користувач власником об'єкта, що прискорює виконання запиту.

Приклад «до застосування методу»: деякі об’єкти не уточнені власником використовуємого об’єкта.

```
SELECT MIN(task_status.id)
FROM task_status
JOIN task ON task_status_id = task_status.id
JOIN project_employee ON project_employee.id = project_employee_id
WHERE project.id = project_id
```

Приклад «після застосування методу»: додана схема БД та таблиця, стовбець якої використовується. Якщо схема встановлена за замовченням, можна її опустити.

```
SELECT MIN(dbo.task_status.id)
FROM dbo.task_status
JOIN dbo.task ON dbo.task_status_id = dbo.task_status.id
JOIN dbo.project_employee ON dbo.project_employee.id = dbo.task.project_employee_id
```

WHERE **dbo**.project.id = **dbo**.**project_employee**.project_id

2.2.2.2 Додавання або видалення квадратних дужок для ідентифікаторів

Опис**:** оберіть ім’я ідентифікатора з будь якими символами за допомогою квадратних дужок. У деяких SQL серверах є внутрішні правила назви ідентифікаторів для особливих об’єктів, тому щоб використовувати будь яке значення для ідентифікатора використовуйте квадратні дужки. Це буде працювати також для символів, які не включені в UNICODE.

Мотивація: щоб дозволити імпорт баз даних з систем з іншими правилами, SQL Server дозволяє розділяти ідентифікатори квадратними дужками. Ви можете додати квадратні дужки до своїх ідентифікаторів або видалити квадратні дужки в рамках переформатування.

Приклад «до застосування методу»: помилка використання, @ - у SQL Server символізує змінну.

SELECT **@employee_position**, COUNT(employee_id)
FROM project_employee
GROUP BY **@employee_position**;

Приклад «після застосування методу»: ідентифікатор наразі символізує ім’я стовбця.

SELECT **[@employee_position]**, COUNT(employee_id)
FROM project_employee
GROUP BY **[@employee_position]**;

2.2.2.3. Додавання точки з комою

Опис**:** вставка крапки з комою в кінці кожного оператора, якщо вони були опущені.

Мотивація**:** вони не є обов'язковими в стандарті SQL, але полегшують читання пакета. Також це допомагає візуально розділити кінець на початок наступного виразу.

Приклад «до застосування методу»: візуально не зрозуміло, де кінець попереднього запиту

```
SELECT @employee_position, COUNT(employee_id)
FROM project_employee
GROUP BY @employee_position

SELECT MIN(task_status.id)

FROM task_status
JOIN task ON task_status_id = task_status.id
JOIN project_employee ON project_employee.id = project_employee_id

WHERE project.id = project_id
```

Приклад «після застосування методу»: точка з комою візуально розділяє запити на окремі

```
SELECT @employee_position, COUNT(employee_id)
FROM project_employee
GROUP BY @employee_position;

SELECT MIN(task_status.id)

FROM task_status
JOIN task ON task_status_id = task_status.id
JOIN project_employee ON project_employee.id = project_employee_id

WHERE project.id = project_id;
```

2.2.2.4 Додавання або видалення псевдоніму (AS)

Адаптовано з методу: «Введення пояснювальної змінної»

Опис**:** ключове слово AS, яке є частиною визначення псевдоніма таблиць.

Мотивація**:** є необов'язковим, хоча його краще використовувати, тому що вони роблять текст більш читабельним. Можна швидше зрозуміти, яке значення поверне вираз з назви псевдоніму.

Приклад «до застосування методу»: відсутність псевдонімів для зовнішнього запиту.

```
SELECT tasks_number_per_employee.name, AVG(tasks_number_per_employee.tasks_number)
FROM(
    SELECT project.name, project_employee.employee_position, COUNT(task.id) as tasks_number
    FROM project_employee
    JOIN project ON project.id = project_employee.project_id
    JOIN task ON task.project_employee_id = project_employee.id
    GROUP BY project_employee.employee_position, project.name) AS tasks_number_per_employee
GROUP BY tasks_number_per_employee.name;
```

Приклад «після застосування методу»: додавання псевдонімів до об’єктів

```
SELECT tasks_number_per_employee.name AS project_name, AVG(tasks_number_per_employee.tasks_number) AS average_tasks_number_per_employee
FROM(
    SELECT project.name, project_employee.employee_position, COUNT(task.id) as tasks_number
```

FROM project_employee

JOIN project ON project.id = project_employee.project_id

JOIN task ON task.project_employee_id = project_employee.id

GROUP BY project_employee.employee_position, project.name) AS tasks_number_per_employee

GROUP BY tasks_number_per_employee.name;

2.2.2.5 Надання оператору повного ім’я

Опис: використайте повне ім’я оператора, а не його псевдонім.

Мотивація: повне ім’я зробить текст запиту більш зрозумілим, адже деталі додають конкретики. Найбільш корисно це буде для людей, які давно не працювали з SQL або тільки навчаються, бо повне ім’я дає додаткову інформацію про використовуємий оператор.

Приклад «до застосування методу»: використані псевдоніми операторів.

SELECT employee.lastname, task.title, project.name project_name
FROM project_employee
JOIN task ON task.project_employee_id = project_employee.id
JOIN project ON project_employee.project_id = project.id
LEFT employee ON employee_id = employee.id

Приклад «після застосування методу»: використані повні імена операторів.

SELECT employee.lastname, task.title, project.name **AS** project_name
FROM project_employee
INNER JOIN task ON task.project_employee_id = project_employee.id
INNER JOIN project ON project_employee.project_id = project.id
OUTER LEFT JOIN employee ON employee_id = employee.id

2.2.3 Спрощення складних запитів

2.2.3.1 Виділення частин запиту у Common Table Expression (CTE)

Адаптовано з методу: «Вилучення методу»

Опис: розбийте складний запит на декілька частин за допомогою CTE.

Мотивація: CTE роблять код більш читабельним, полегшує налагодження запитів. Такі вирази можуть посилатися на результати кілька разів протягом усього запиту. Зберігаючи результати підзапиту, ви можете повторно використовувати їх у більшому запиті. CTE можуть допомогти вам виконати багаторівневе агрегування. Використовуйте CTE для зберігання результатів агрегації, які потім можна узагальнити в основному запиті.

Приклад «до застосування методу»: запит без використання CTE.

```
SELECT tasks_number_per_employee.name, AVG(tasks_number_per_employee.tasks_number)
FROM(
    SELECT project.name, project_employee.employee_position, COUNT(task.id) as tasks_number
    FROM project_employee
    JOIN project ON project.id = project_employee.project_id
    JOIN task ON task.project_employee_id = project_employee.id
    GROUP BY project_employee.employee_position, project.name) AS tasks_number_per_employee
GROUP BY tasks_number_per_employee.name;
```

Приклад «після застосування методу»: запит з інкапсульованим підзапитом у якості CTE.

```
WITH tasks_number_per_employee AS
```

(SELECT project.name, project_employee.employee_position, COUNT(task.id) as tasks_number

FROM project_employee

JOIN project ON project.id = project_employee.project_id

JOIN task ON task.project_employee_id = project_employee.id

GROUP BY project_employee.employee_position, project.name)

SELECT tasks_number_per_employee.name, AVG(tasks_number_per_employee.tasks_number)

FROM **tasks_number_per_employee**

GROUP BY tasks_number_per_employee.name;

2.2.3.2 Заміна алгоритму об’єднання таблиць

Адаптовано з методу: «Заміна алгоритму»

Опис**:** заміна умовного виразу WHERE table1.table2Id = table2.Id на конструкцію з використанням INNER JOIN для об’єднання таблиць.

Мотивація**:** даний метод дозволяє зробити запит простіше для розуміння за рахунок спеціального виразу, який створено задля цієї операції. Таким чином можна розділити умову та вираз за яким треба поєднати таблиці.

Також це сприяє уникненню застосування неявного CROSS JOIN. У Декартовому приєднанні створюються всі можливі комбінації змінних. Це неефективне використання ресурсів бази даних, оскільки база даних виконала роботу в N разів більше, ніж потрібно. Декартові об’єднання є особливо проблематичними у великомасштабних базах даних, оскільки декартове об’єднання двох великих таблиць може створити мільярди чи трильйони результатів.

Приклад «до застосування методу»: об’єднання таблиць за ключом у фільтрі.

SELECT project.name, project_employee.employee_position, COUNT(employee_id)

FROM project_employee, project

WHERE employee_position = ‘developer’ AND **project.id = project_employee.project_id**

GROUP BY project_employee.employee_position, project.name;

Приклад «після застосування методу»: об’єднання таблиць за ключом за допомогою JOIN.

SELECT project.name, project_employee.employee_position, COUNT(employee_id)

FROM project_employee

JOIN project ON project.id = project_employee.project_id

WHERE employee_position = ‘developer’

GROUP BY project_employee.employee_position, project.name;

2.2.4 Спрощення умовних виразів

2.2.4.1 Перетворення умови на IN вираз

Адаптовано з методу: «Консолідація умовного виразу»

Опис: якщо ви вибрали одне або декілька значень в одному стовпці, ви можете скористатися цією функцією для форматування вибраних значень як використання у вигляді IN функції.

Мотивація: легше представити складний умовний вираз у вигляді списку значень, які мають співпасти за умовою.

Приклад «до застосування методу»: складна умова, яка може бути спрощена за допомогою IN.

SELECT project.name, project_employee.employee_position, COUNT(employee_id)

FROM project_employee

JOIN project ON project.id = project_employee.project_id

WHERE **employee_position = ‘developer’ OR employee_position = ‘QA’ OR employee_position = ‘manager’**

GROUP BY project_employee.employee_position, project.name;

Приклад «після застосування методу»: використання IN оператора.

SELECT project.name, project_employee.employee_position, COUNT(employee_id)

FROM project_employee

JOIN project ON project.id = project_employee.project_id

WHERE employee_position **IN (‘developer’, ‘QA’, ‘manager’)**

GROUP BY project_employee.employee_position, project.name;

2.2.4.2 Використання WHERE до/замість HAVING

Адаптовано з методу: «Консолідація умовних виразів»

Опис: використовуйте пропозицію WHERE для фільтрації зайвих рядків, тому вам не доведеться обчислювати ці значення в першу чергу. Лише після видалення нерелевантних рядків, а також після об’єднання цих рядків та їх групування слід включити пропозицію HAVING для фільтрації агрегатів.

Мотивація: добра практика використовувати оператори за їх зазначенням, так текст запитів залишається зрозумілим для всіх.

Приклад «до застосування методу»: використання HAVING не за призначенням

SELECT employee.firstname +' '+ employee.lastname as name, COUNT(task.id) AS Count

FROM project_employee

JOIN task ON task.project_employee_id = project_employee.id

JOIN employee ON employee_id = employee.id

WHERE task.task_status_id <= 3

GROUP BY employee.firstname +' '+ employee.lastname

HAVING employee.firstname +' '+ employee.lastname LIKE(%M);

Приклад «після застосування методу»: фільтрування рядків перед групуванням

SELECT project.name, project_employee.employee_position, COUNT(employee_id)

FROM project_employee

JOIN project ON project.id = project_employee.project_id

WHERE employee_position IN (‘developer’, ‘QA’, ‘manager’) **AND employee.firstname +' '+ employee.lastname LIKE(%M)**

GROUP BY project_employee.employee_position, project.name;

2.2.4.3 Заміна LIKE на оператор рівності

Опис: LIKE порівнює символи та може бути об’єднаний у пару з операторами підстановки, такими як %, тоді як оператор = порівнює рядки та числа для точних збігів. Якщо є потреба точно порівняти значення, то використовуйте «=».

Мотивація: LIKE є не досить очевидною операцією, стандартна практика порівняння зазвичай використовує оператор «=». Оператор порівняння може скористатися індексованими стовпцями, що робить запит більш ефективним.

Приклад «до застосування методу»: використання регулярного виразу для перевірки на рівність

SELECT project.name, project_employee.employee_position, COUNT(employee_id)

FROM project_employee

JOIN project ON project.id = project_employee.project_id

WHERE employee_position IN (‘developer’, ‘QA’, ‘manager’) AND employee.firstname +' '+ employee.lastname **LIKE(Mike Miller)**

GROUP BY project_employee.employee_position, project.name

HAVING employee.firstname +' '+ employee.lastname LIKE(%M);

Приклад «після застосування методу»: без використання регулярного виразу

SELECT project.name, project_employee.employee_position, COUNT(employee_id)

FROM project_employee

JOIN project ON project.id = project_employee.project_id

WHERE employee_position IN (‘developer’, ‘QA’, ‘manager’) AND employee.firstname +' '+ employee.lastname **= ‘Mike Miller’**

GROUP BY project_employee.employee_position, project.name;

2.2.4.4 Використання EXISTS або JOIN замість IN

Опис: якщо вам просто потрібно перевірити наявність значення в таблиці, віддайте перевагу EXISTS перед IN. Це також справедливо для NOT EXIST. У разі, якщо треба відфільтрувати дані, де важливим є повтор значень та є потреба у даних другої таблиці, віддайте перевагу JOIN.

Мотивація: використовуйте оператори за їх призначенням, це дозволить швидше розробникам зрозуміти текст запиту. Також оскільки процес EXISTS завершується, як тільки він знайде значення пошуку, тоді як IN буде сканувати всю таблицю. IN слід використовувати для пошуку значень у списках.

Приклад «до застосування методу»: запит з використанням IN

SELECT *

FROM Customers

WHERE ID **IN** (

SELECT CustomerID

```
FROM Orders)
```

Приклад «після застосування методу»: запит з використанням EXIST

```
SELECT *
FROM Customers
WHERE EXISTS (
    SELECT *
FROM Orders
WHERE Orders.CustomerID = Customers.ID)
```

Приклад «після застосування методу»: запит з використанням INNER JOIN

```
SELECT Customers.*
FROM Customers
JOIN Orders ON Customers.ID = Orders.CustomerID
```

2.2.5 Робота з коментарями

2.2.5.1 Додавання або видалення коментарів

Опис: додавайте опис функціоналу у коментарі, якщо це є необхідністю. Якщо немає потреби, видалите коментарі.

Мотивація: якщо запит є складним для розуміння, частини запиту не є очевидними – пишіть коментарі так, щоб розробники могли зрозуміти що відбувається у запиті та куди саме відноситься даний коментар. У разі якщо запит маленький та простий можете видалити коментарі, особливо, якщо закоментовано непрацюючий запит, нагадування про написання реалізації запиту або дещо, що не стосується тексту запиту.

Приклад «до застосування методу»: невдале використання коментарів.

```
-- to do: use one code style
```

```
SELECT CountTasks.name, AVG(CountTasks.Count) -- add alias
FROM(
-- to do: extract as CTE
    SELECT project.name, project_employee.employee_position,
COUNT(task.id) as Count -- change alias name
    FROM project_employee
    JOIN project ON project.id = project_employee.project_id
    JOIN task ON task.project_employee_id = project_employee.id
    GROUP BY project_employee.employee_position, project.name) AS
CountTasks
GROUP BY CountTasks.name;
```

Приклад «після застосування методу»: додано опис запиту, видалені нагадування про реалізацію

```
-- Description: Calculate the average number of tasks on each project per each
employee

WITH tasks_number_per_employee AS
(SELECT project.name, project_employee.employee_position, COUNT(task.id)
AS tasks_number
    FROM project_employee
        JOIN project ON project.id = project_employee.project_id
        JOIN task ON task.project_employee_id = project_employee.id
    GROUP BY project_employee.employee_position, project.name)

SELECT tasks_number_per_employee.name AS project_name,
AVG(tasks_number_per_employee.tasks_number) AS
average_tasks_number_per_employee
FROM tasks_number_per_employee
GROUP BY tasks_number_per_employee.name;
```

Висновки до другого розділу

Отже, в результаті роботи у другому розділі був створений перелік методів рефакторингу, які вже готові до споживання розробниками, яким необхідно покращити якість текстів SQL запитів. Методи розділені на групи в залежності від цілі їх використання. Загалом методи створені для того, щоб покращити читабельність, пришвидшити розуміння коду іншими розробниками, але деякі з них ще позитивно впливають на ефективність виконання, але це не є цілю даної роботи, тому розглядатися не буде.

Методи можна застосовувати не залежно від СКБД, яку використовує розробник, головне, щоб в ній підтримувалась мова SQL. Також методи не є залежними один від одного, тому застосовувати їх можна в довільному порядку.

Самі методи не є строгою інструкцією для виконання, а є лише порадами та переліком гарних практик програмування. Використовуючи той чи інший метод розробник сам приймає рішення щодо його застосування. Також методи можна використовувати й в зворотному напрямку, якщо це на погляд розробника є гарною ідеєю. Наприклад, метод «виділення як CTE» можна застосувати навпаки: підзапит вбудувати в основний запит, якщо він є простим, зрозумілим та малим.

3 РОЗРОБКА ІНСТРУМЕНТУ РОЗРАХУНКУ ЗА ARI

3.1 Формалізація задачі

Взаємодія користувача з програмним продуктом відображена через діаграму прецедентів [38]. Діаграма описує процеси, які з точки зору користувача може виконувати програма. Актором програмної системи представлено користувача, який взаємодіє з розробленим сервісом. Варіанти використання відображають функціональність продукту, що може активувати користувач. Варіанти використання системи зображені на рис. 3.1:

Рисунок 3.1. – Діаграма прецедентів взаємодії користувача з системою

3.2 Архітектура програмної системи

Розроблений програмний продукт – допоміжний сервіс у досліджені даної теми. Складається програма з трьох основних частин:

- інтерфейс користувача;
- основна частина програми, яка розраховує метрику ARI;
- модуль для роботи з файлами.

Графічний інтерфейс було розроблено за допомогою інтерфейсу програмних застосунків WindowsForms. Дана бібліотека підтримується операційною системою Windows та є складовою частиною Microsoft .NET Framework.

Основна частина програми написана мовою C#. Парадигма даної мови – об’єктно орієнтовна, тому при процесі проектування були дотримані основні принципи парадигми програмування та застосований принцип SOLID. Для наглядного представлення взаємодії модулів системи пропонується UML діаграма компонентів [38]. Дана діаграма зображена на рисунку 3.2.

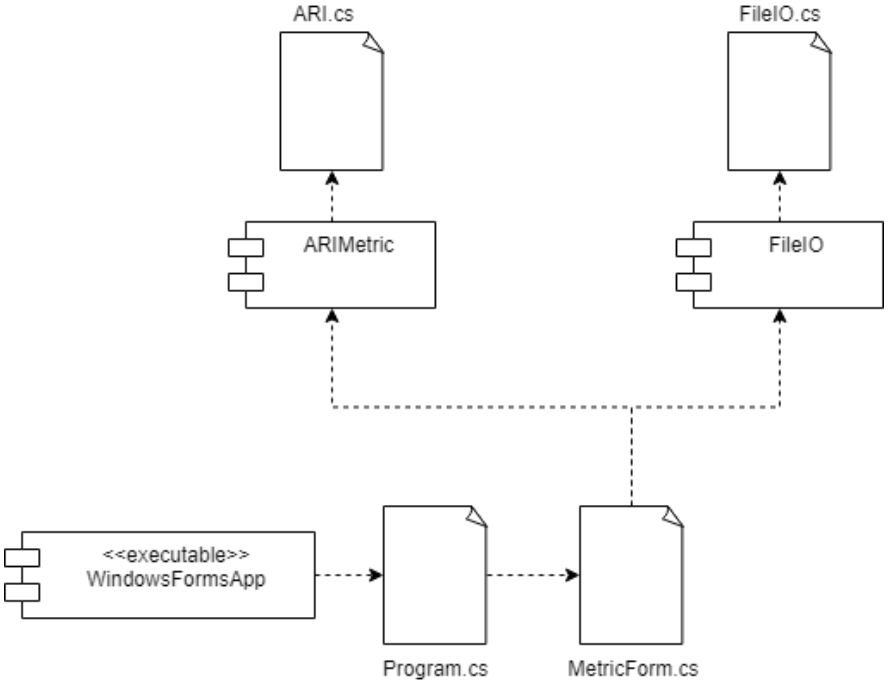


Рисунок 3.2 – Взаємодія компонентів в системі

3.3 Внутрішнє проектування

3.3.1 Вибір мови програмування

У якості мови програмування була обрана мова C#. Перевага була надана даній мові через наступні характеристики [48]:

– мова C# є високорівневою, тому людині легко сприймати її та читати;

– C# використовує об’єктно орієнтований підхід. Це дозволяє робити великомасштабні проекти з великою функціональністю, які легко підтримувати;

– мова має велику спільноту розробників, тому багата на фреймворки та бібліотеки різних напрямів, що прискорює написання додатків;

– забезпечена деталізованою документацією та має стандарти оформлення коду;

– мова має велику кількість синтаксичних конструкцій, що добре впливає на читабельність тексту;

– не потребує керування пам’ятью комп’ютеру, так як має вбудований збірник мусору.

В таблиці 3.1 відображено популярні мови програмування, які порівняні з C#.

Таблиця 3.1 – Порівняння популярних мов програмування

Мова програмування	Призначення	Основана парадигма програмування	Стандарти
C#	Настільні додатки, web додатки, програми-сервери	ООП	ISO, ECMA
C++	Системні застосунки	ООП, імеративна	ISO
Java	Настільні додатки, web додатки, програми-сервери	ООП	Java language Specification
Python	Програмні-сервери, штучний інтелект, наукове програмування	ООП, імперативна, функціональна	PEP
JavaScript	Web-клієнти	ООП, імперативна	ECMA

Для розробки настільного додатку за таблицею 3.1 були запропоновані дві мови програмування: C# та Java. За кількістю досвіду розробки на мові C# було прийнято рішення обрати дану мову.

3.3.2 Технологічна платформа

При розробці програмної системи було прийнято рішення використати бібліотеку Microsoft .NET Framework – Windows Forms [47]. Дана бібліотека підходить до швидкої та легкої розробки графічного інтерфейсу застосунку для

операційної системи Windows. Дана бібліотека має у собі основні елементи інтерфейсу, які зазвичай використовуються у стандартних програмах Windows. Windows Forms користується подієво-орієнтовний підходом, тому багато часу витрачається на очікування від користувача дій, наприклад кліку мишкою на кнопку.

3.3.3 Ієрархія та взаємодія класів системи

Для демонстрації ієрархії системи за видом взаємодії класів між собою була розроблена діаграма класів, яка зображена на рисунку 3.3.

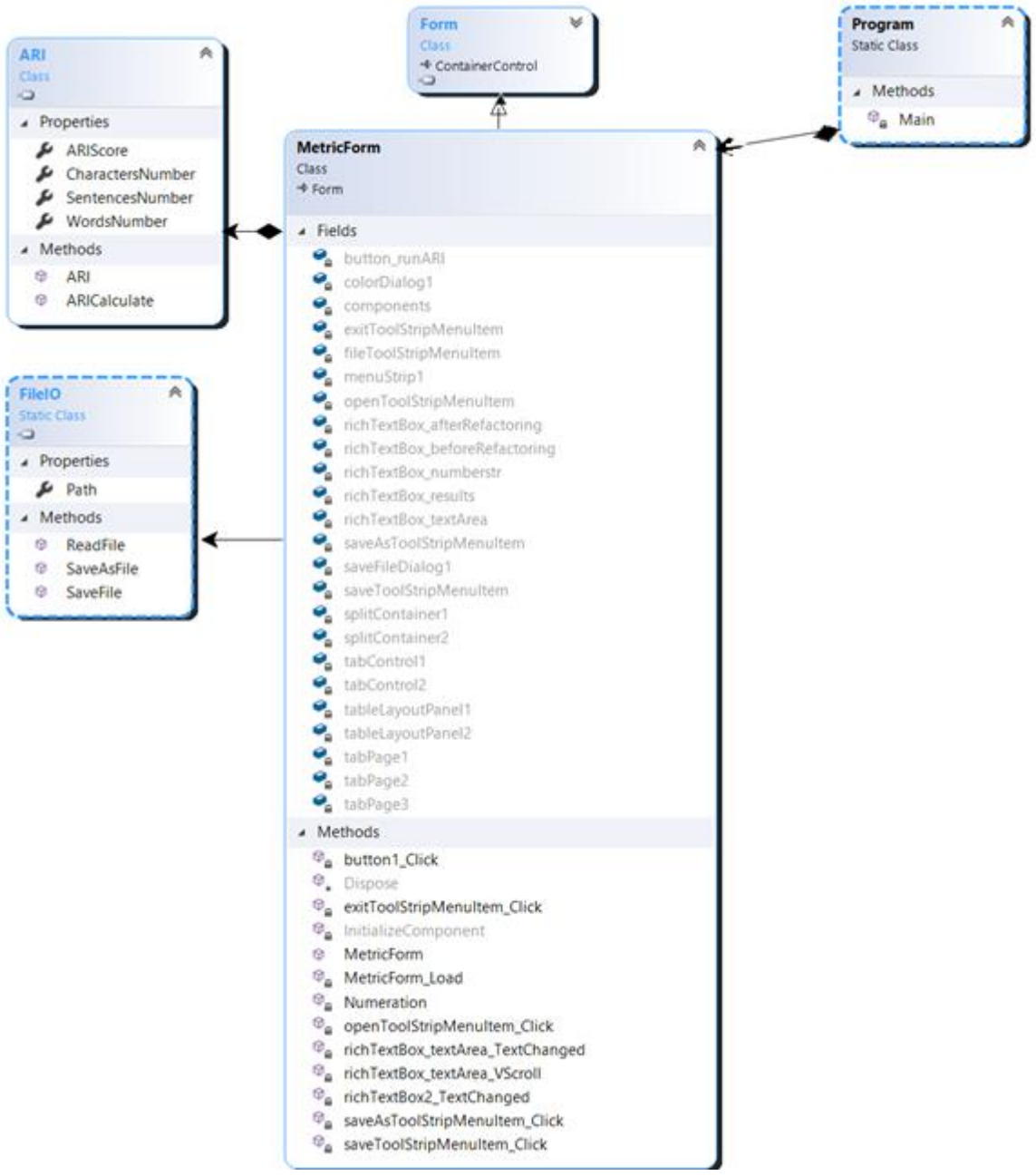


Рисунок 3.3 – Діаграма класів

3.3.4 Використані принципи проектування

При проектуванні програмної системи було застосовано наступні принципи проектування [46]:

– YAGNI;

– DRY;

– KISS;

– SOLID.

Перший принцип YAGNI – (англ. you ain’t gonna need it) вам це не потрібно. Даний принцип говорить про те, що не треба реалізовувати задачі, які не були поставлені за ціль.

Принцип DRY – (англ. don’t repeat yourself) не повторюйтесь. Не треба писати код раз за разом, бо можна винести його як окремий метод та не засмічувати текст програми.

KISS – (англ. keep it simple) – не ускладнюй. Сенс цього принципу в тому, щоб додержуватися простих та зрозумілих архітектурних рішень, застосовувати шаблони проектування.

SOLID – це збірка п’яти принципів, що пропонують додержуватися наступних правил:

– принцип єдиної відповідальності – розробка класів повинна бути відповідно до єдиної задачі, яка на нього накладається;
– принцип відкритості та закритості – класи та методи відкриті до модифікації, але закриті для змін;
– принцип заміщення Барбари Лісков – усі розроблені типи можуть бути замінені базовими типами без втрат;
– принцип розділення інтерфейсу – інтерфейси не повинні бути перевантажені та мати вузьку сферу застосування;
– принцип інверсії залежностей – абстракції вищого рівня не повинні залежати від подробиць, нижчих рівнів.

3.4 Розробка інтерфейсу користувача

Програмний застосунок для аналізу читабельності запитів, які були редаговані за методами рефакторингу має два основних вікна графічного інтерфейсу користувача. При розробці програми стоїть задача відтворення розрахунків метрики читабельності для запитів мовою SQL, щоб показати різницю текстів виражену в відношенні значень ARI метрики для текстів «до» та «після» застосування рефакторингу. При запуску програми перед користувачем з’являється головне вікно, що відображено на рисунку 3.4.

Рисунок 3.4. – Головне вікно програми

Вікно містить:

– Полосу меню, яка містить у собі випадаючий список роботи з фалами на носії. Список містить наступні пункти: відкрити файл формату .sql, зберегти файл в поточному файлі, зберегти файл, як новий в форматі .sql, закрити програму. За один виклик можна відкрити тільки один файл.

– Полосу нумерації рядків для основної області редагування тексту запитів. При написанні або видаленні тексту нумерація змінюється відповідно рядку тексту.

– В центрі вікна перша вкладка, у якій область призначена для написання та редагування тексту запитів.

– З правої сторони буде стовбець, який відображає результати розрахунку метрики ARI. Активуються розрахунки за допомогою кнопки «Run ARI Metric».

Для порівняння запитів, можна буде перемикнути в основній області вкладку на другу під назвою «Compare» та побачити два вікна. Перше призначено для тексту запитів «до» рефакторингу, друге – «після». На рисунку 3.5 відображено користувацький інтерфейс з ввімкненою другою вкладкою.

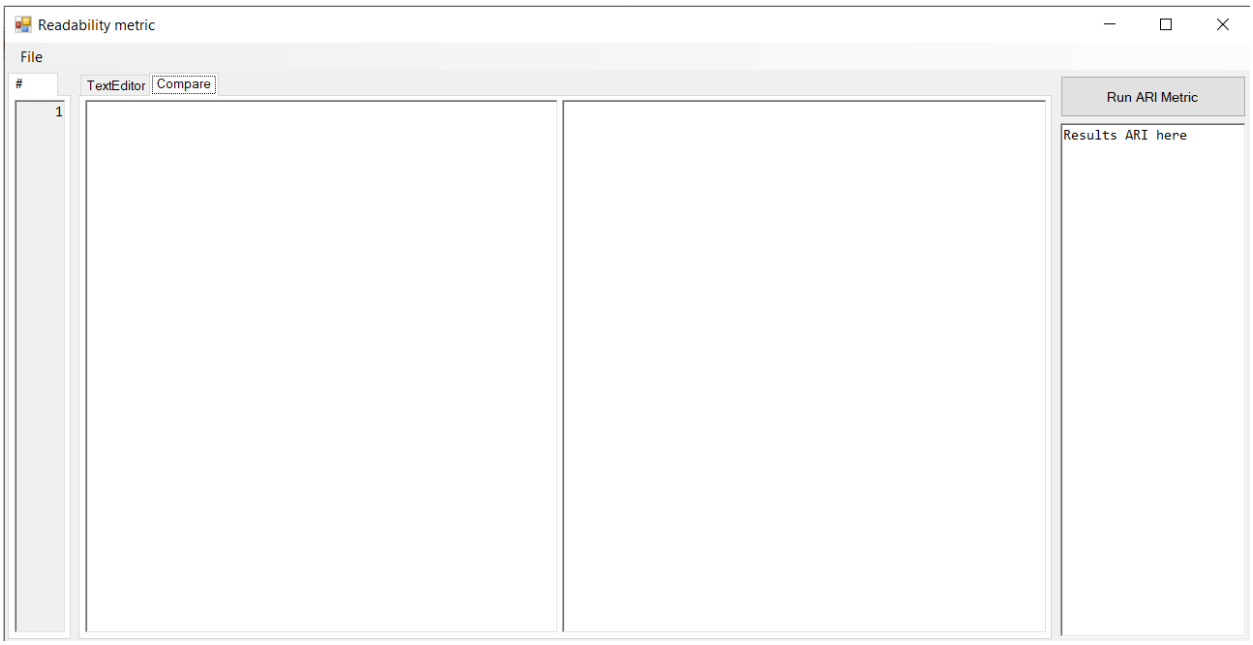


Рисунок 3.5 – Вкладка «Порівняння» головного вікна

При завантаженні файлу з текстом запитів тест автоматично буде розташований як на першій вкладці головного редактору, так і на другій вкладці у області зліва, що умовно виділена як текст запитів «до» редагування. При наборі тексту вручну треба також додавати варіант «до» вручну, щоб точна визначити вихідну позицію тексту.

При відкритті та збереженні файлу відкривається діалогове вікно, що фільтрує файли за форматом SQL для зручності пошуку. Приклад виклику вікна позначено на рисунку 3.6.

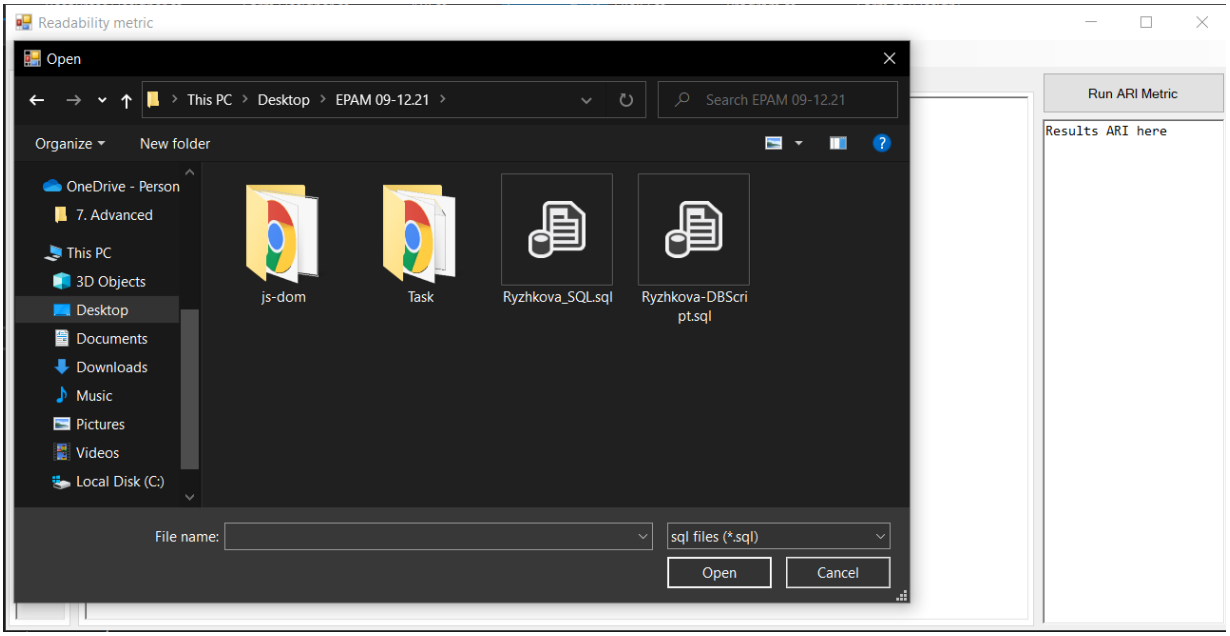


Рисунок 3.6 – Діалогове вікно

Для інформування користувача програма відображає вікно, у якому описана суть причини виникнення винятковою ситуації. Приклад такої ситуації відображено на рисунку 3.7.

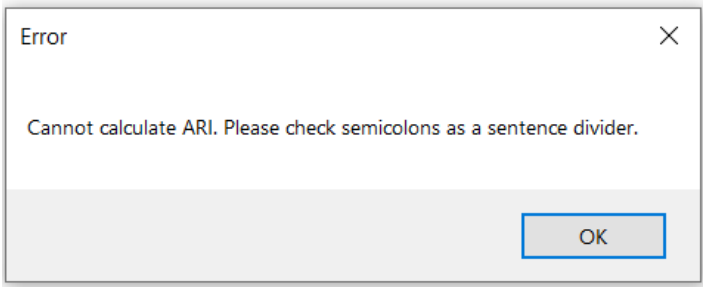


Рисунок 3.7.– Повідомлення про помилку

Розрахунок метрики відображений з правого боку на рисунку 3.8 та складається з наступної структури:

– ARI розрахунок для запитів до редагування;

– кількість символів у тексті запитів до редагування;

– кількість слів у тексті запитів до редагування;

– кількість запитів (речень) у тексті запитів до редагування;

– ARI розрахунок для запитів після редагування;

– кількість символів у тексті запитів після редагування;

– кількість слів у тексті запитів після редагування;

– кількість запитів (речень) у тексті запитів після редагування;

– значення відношення розрахунку ARI запитів «до» до «після» рефакторингу.

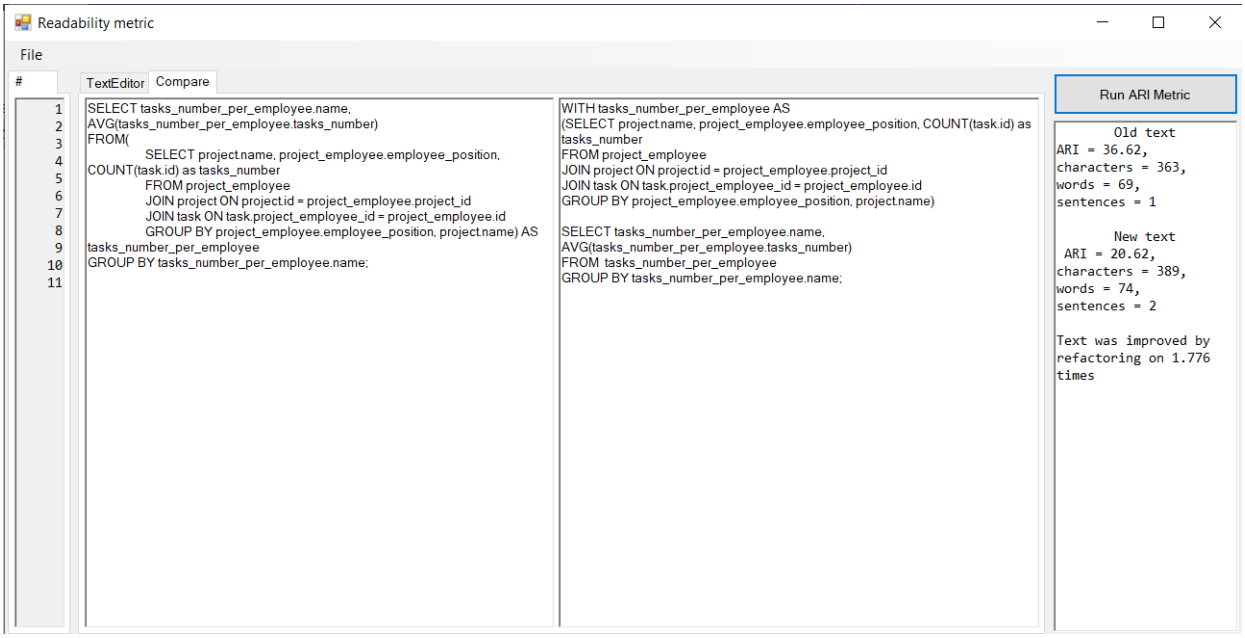


Рисунок 3.8 – Приклад результатів

3.5 Тестування та налагодження програми

3.5.1 Аналіз методів тестування та відлагодження

Тестування програмного забезпечення є дуже важливою складовою розробки. Тестування призване зробити програму якісніше, щоб витримувати несподівані ситуації, які можуть статися. Для написання тестів треба визначитися з рівнем на якому вони проводяться, видом тестування та вирішити які компоненти системи будуть тестуватися. Для досягнення доброї якості продукту необхідно покрити майже всю частину коду тестами, щоб достовіритися, що всі шляхи програми є досяжними та працюють так, як очікує користувач.

Для ефективного тестування був використаний підхід «чорного ящику», де за допомогою юніт тестів був покритий клас ARIMetric, який є ядром логіки даного застосунку. Для тестування графічного інтерфейсу та класу FileIO був застосований метод «припущення про похибку» [36].

Метод тестування «чорний ящик» застосовується для розробки тестів на основі аналізу специфікації програми. Тестування програми обмежується, тому що при

такому підході тестувальник не має доступу до тексту програми та важче перевірити усі шляхи виконання програми спираючись лише на специфікацію. Для автоматизації тестування розробник може скористатися юніт тестами, які добре підходять для здійснення повторного тестування за малий проміжок часу, документують текст програми та забезпечують якість програми. Добре написані юніт тести повинні мати наступні якості [52]:

– бути унікальними: уникати тестування одного й того ж випадку кілька разів;

– бути швидкими для виконання;

– не містити у собі логіки, яка може містити помилки та перенавантажувати тест;

– бути незалежними від середи, де вони запускаються;

– бути мінімалістичними та легко підтримуватись.

3.5.2 Проектування тестів за допомогою методу еквівалентних розбитів

У методі еквівалентного розбиття множина вхідних значень утворює вихідний простір. Для кожного набору даних вхідних значень, які можна об’єднати за виявленням одних й тих самих помилок та вихідних значень, треба зробити тест, який би покрив дану частину множини вхідних значень. Виділення класів еквівалентності є евристичним процесом, тому треба брати до уваги як допустимі значення, так й недопустимі для деякого випадку [36, 37].

Класи еквівалентності для модуля ARIMetric приведені нижче у таблиці 3.2.

Таблиця 3.2 – Класи еквівалентності

Вхідні умови	Допустимі значення	Недопустимі значення
1	2	3
Запис тексту запитів	1.Текст одного запиту	4.Пустий текст запиту

Продовження таблиці 3.2

1	2	3
	2.Текст запитів, які коректно розділені точкою з комою	
	3.Текст запитів, що містить CTE, яка візуально розділена на кілька частин	

На кожний клас еквівалентності треба виділити один тест, який перевірить множину вхідних значень для одного виду випадку. Метод, що розглядається має на вході строку з текстом запитів, а на виході значення розрахунку метрики ARI. Розглянутий клас також має інформацію про кількість речень, слів та літер в тексті, що теж виводиться користувачу разом з результатом метрики. Запропоновані тести представлені у таблиці 3.3.

Таблиця 3.3 – Метод еквівалентних розбиттів

Клас, що перевіряється	Тест	Вхідне значення ARI	Вихідне значення ARI
1	2	3	4
1	1	SELECT * FROM Table	ARI = 14.895, літери = 15, слова = 2, речення = 1
2	2	SELECT * FROM Table; SELECT * FROM Table2;	ARI = 4.405, літери = 31, слова = 6, речення = 2

Продовження таблиці 3.3

1	2	3	4
1, 3	3	WITH Table AS (SELECT * FROM Table2) SELECT * FROM Table;	ARI = 2.8, літери = 42, слова = 9, речення = 2
4	4	*Порожня строка*	ArgumentException

3.5.3 Тестування системи за допомогою модульного тестування

Юніт тестування (англ. Unit tests), або модульне тестування – процес в програмуванні, який дозволяє перевірити окремі модулі програмної системи на якість та коректність роботи [52].

Модульні тести пишуться для відкритих методів, які містять складну логіку. Такий спосіб тестування дозволяє швидко перевірити коректність роботи написаного коду, чи не призвело змінення коду до появи нових помилок, а також полегшує знаходження місць помилок.

В даній роботі використовується фреймворк MSTest для реалізації модульного тестування. Даний фреймворк є одним з найпопулярніших для .NET Framework. MSTest є рідним для середи розробки Visual Studio, адже встановлюється за замовченням. Функціонал, який попонує фреймворк повністю покриває потреби у тестуванні в даному проекті.

В реалізації модульних тестів був використаний принцип AAA [51]. AAA (англ. Arrange, Act, Asset) – принцип для написання модульних тестів, який структурує вміст тесту на три етапи: ініціалізація даними, виконання методу для його перевірки, порівняння результатів: очікуваних та отриманих при виконанні перевіряємого методу.

Для тестування був обраний модуль ARIMetric, який займається розрахунком метрики читабельності та містить важливу логіку додатку. Проходження тестів представлені на рисунку 3.9.

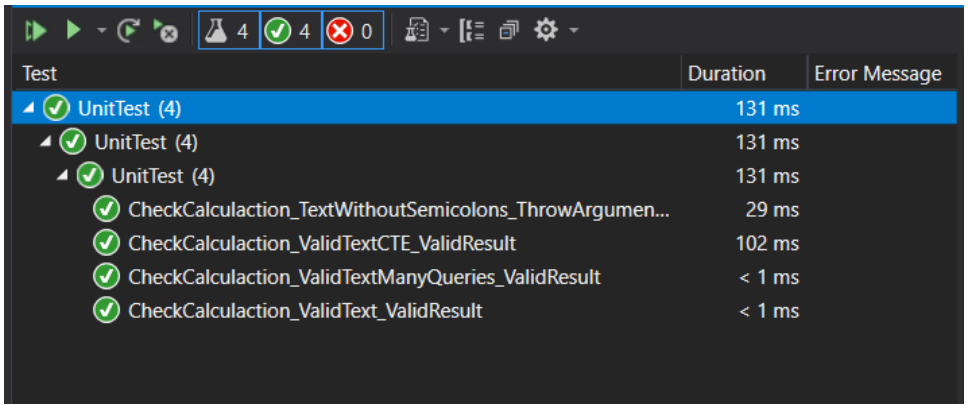


Рисунок 3.9 – Модульні тести

Всі тести були успішно пройдені, очікувані результати співпали з актуальними. Тести були розроблені відповідно до таблиці 3.3.

3.5.4 Тестування системи методом припущення про похибку

Тестування методом припущення про помилку робиться досвідченим розробником. Досвід розробника допомагає йому знайти вузькі місця, які треба протестувати та знайти помилки. Складається список вузьких місць, де передбачено буде знайдено помилку та перевіряється. При виникненні помилки – програма виправляється [36].

В даному розділі буде протестовано останні два модулі: FileIO, що відповідальний за роботу з файлами на носії, та WindowsFormsApp, що відповідальний за графічний інтерфейс користувача.

Висунемо ряд припущень про помилку для модуля FileIO:

1. Якщо відкрито файл не sql формату, чи буде зчитано текст файлу у робочу область?
2. Якщо файл зберігається в поточний файл, який відкрито та такого файлу більше не існує, чи буде повідомлення про помилку?
3. Якщо файл зберігається у якості окремого файлу, чи буде подальша робота робитися відносно нового файлу?

На таблиці 3.4 відображено список вузьких місць для модуля FileIO.

Таблиця 3.4 – Припущення про помилку FileIO

Тест	Вхідні дані	Очікувані вихідні дані	Актуальні вихідні дані
1	Файл не sql формату	Повідомлення про помилку	Відкриття файлу в робочу область
2	Збереження в неіснуючий файл	Повідомлення про помилку	Не оброблене виключення
3	Робота з файлом з нового шляху	Дані зберігаються у новий файл	Дані зберігаються у новий файл

Висунемо ряд припущень про помилку для модуля WindowsFormsApp:

1. Чи редагується текст в текстовій області «після» разом з головною?
2. Чи додається текст до області «до» при загрузці файлу з носія?
3. Чи коректно працює нумерація строк при маніпулюванні даними в текстовій області та при прокрутці?
4. Чи правильно працює адаптація інтерфейсу до різних розмірів вікна?

На таблиці 3.5 відображено список вузьких місць для модуля WindowsFormsApp.

Таблиця 3.4 – Припущення про помилку WindowsFormsApp

Тест	Вхідні дані	Очікувані вихідні дані	Актуальні вихідні дані
1	2	3	4
1	Редагування тексту у головній області, редагування тексту у області «після»	Області взаємно редагуються	Області взаємно редагуються

Продовження таблиці 3.4

1	2	3	4
2	Загрузка файлу з носія	Текст відображається на області «до»	Текст відображається на області «до»
3	Додавання нових рядків, видалення рядків, прокрутка в головній області	Номера строк відображаються коректно	Номера строк відображаються коректно
4	Розтягування вікна	Інтерфейс зберігає форму	Інтерфейс зберігає форму

При тестуванні методом припущення про помилку були виправлені всі знайдені помилки, а саме перший та другий тест для модулю FileIO. Помилки були у недостатній кількості покриття виключних ситуацій, які можуть виникнути під час користуванням програмою.

Висновки до третього розділу

Отже, в даному розділі було описано архітектуру програми для дослідження доцільності пристосування методів рефакторингу з точки зору читабельності тексту. Програма забезпечує правильний розрахунок метрики ARI. У ході проектування програми, систему було розбито на окремі модулі, щоб забезпечити слабке зачеплення. Такі модулі можна застосовувати і в інших програмних системах, якщо буде така потреба. За принципами, які пристосовувалися до даного проекту можна відмітити, що класи мають єдину відповідальність, не мають частини коду, які б не використовувалися додатком, текст програми не має дублювання. Дані принципи допомагають підтримувати текст програми чистим та легким для розуміння.

Обрані технології розробки є добре відомими та забезпечені детальною документацією, що дає змогу розробнику швидко включитися в розробку та впроваджувати необхідний функціонал.

Інтерфейс програми є простим та легким для розуміння. Всі елементи підписані відповідно до їх функціональності та є знайомими для користувачів операційної системи Windows. Інтерфейс також є адаптивним, тому можна налаштувати його для своєї роботи.

Тестування даної програми відбувалося за підходами «чорної скриньки». Були використані такі методи, як еквівалентне розбиття та припущення про помилку. Тести, що були описані за методом еквівалентного розбиття були реалізовані у якості модульних тестів для проекту. Для модульних тестів був використаний фреймворк MSTest, що є базовим для .NET Framework. За допомогою такої технології можна легко проводити тестування, швидко та робити це повторно при зміненні тексту програми, щоб бути впевненим у якості програмної системи.

4 ОЦІНКА МЕТОДІВ РЕФАКТОРИНГУ SQL ЗАПИТІВ

Так як мова SQL є спорідненою мовою до природніх мов, вирішено скористатися метриками читабельності для природніх мов. Нажаль, не існує метрики, щоб була розроблена для оцінки читаємості текстів програм, тощо, тому щоб довести результати даної роботи було використано метрику Automated Readability Index, яка була створена для реалізації на обчислювальних машинах для апроксимації складності читання до номеру класу учня в американській системі освіти. Сама метрика обчислюється за наступною формулою [29]:

$$ARI = 4.71 * \frac{\text{Кількість символів}}{\text{Кількість слів}} + 0.5 * \frac{\text{Кількість слів}}{\text{Кількість речень}} - 21.43.$$

Кількість символів розраховується тільки для літер та цифр, інші символи пропускаються. Для адаптації метрики кількість речень буде рівною кількості запитів, які розділені між собою точкою з комою.

Значення самої метрики можна трактувати так, що чим більше значення, тим складніше текст, який представлено перед читачем. Але в даному досліджені набагато ціннішою інформацією є відношення значень, які замірювалися до застосування методів рефакторингу та після.

Дослідження проведене для всіх запропонованих методів рефакторингу та наведене у таблицях 4.2 – 4.16. Розрахунки запропоновані для прикладів, що демонструють методи рефакторингу. Розрахунки виконані за допомогою програми, яка розроблювалась в третьому розділі.

4.1 Складання запитів

4.1.1 Перейменування псевдонімів та змінних

Приклад «до застосування методу»:

```
SELECT Table.name, AVG(Table.Count)
FROM(
```

```
SELECT project.name, project_employee.employee_position,
COUNT(task.id) AS Count
FROM project_employee
JOIN project ON project.id = project_employee.project_id
JOIN task ON task.project_employee_id = project_employee.id
GROUP BY project_employee.employee_position, project.name) AS Table
GROUP BY Table.name;
```

Приклад «після застосування методу»:

```
SELECT tasks_number_per_employee.name, AVG(tasks_number_per_employee.
tasks_number)
FROM(
SELECT project.name, project_employee.employee_position,
COUNT(task.id) as tasks_number
FROM project_employee
JOIN project ON project.id = project_employee.project_id
JOIN task ON task.project_employee_id = project_employee.id
GROUP BY project_employee.employee_position, project.name) AS
tasks_number_per_employee
GROUP BY tasks_number_per_employee.name;
```

Таблиця 4.1 – Розрахунок для методу «Перейменування псевдонімів та змінних»

Запит	Кількість символів	Кількість слів	ARI
«До»	283	55	30.305
«Після»	363	69	37.849

Інтерпретація результатів

Результати показали, що читабельність знизилася у 1.25 разів. Таке значення отримане через те, що при перейменуванні слова були змінені на більш довгі, що краще розкривають сенс функціоналу за який відповідний об’єкт. Щоб цей метод

покращував читабельність скорочуйте слова та намагайтеся зберегти сенс, який в них вкладено.

4.1.2 Підстановка розширенням

Приклад «до застосування методу»:

```
SELECT *
FROM project_employee
JOIN task ON task.project_employee_id = project_employee.id
JOIN project ON project_employee.project_id = project.id
JOIN employee ON employee_id = employee.id
```

Приклад «після застосування методу»

```
SELECT employee.lastname, task.title, project.name
FROM project_employee
JOIN task ON task.project_employee_id = project_employee.id
JOIN project ON project_employee.project_id = project.id
JOIN employee ON employee_id = employee.id
```

Таблиця 4.2 – Розрахунок для методу «Підстановка розширенням»

Запит	Кількість символів	Кількість слів	ARI
«До»	153	30	17.591
«Після»	189	36	21.298

Інтерпретація результатів

Результати показали, що даний метод негативно впливає на читабельність у 1.22 разів. Це стається через те, що до запиту додаються нові слова, які є переліком полів, що необхідні для вибірки. Така практика підвищить розуміння та надійність запиту, але доведеться пожертвувати читабельністю.

4.1.3 Перелічення параметрів для вставки

Приклад «до застосування методу»:

```
INSERT task
VALUES
(1, 'Create project architecture','ASP.NET project + Angular', 4, 6),
(2, 'Create calendar matrix','ASP.NET project + Angular', 1, 1),
(3, 'Test plan','Write test plan', 4, 4);
```

Приклад «після застосування методу»:

```
INSERT task(id, title, description, task_status_id, project_employee_id)
VALUES
(1, 'Create project architecture','ASP.NET project + Angular', 4, 6),
(2, 'Create calendar matrix','ASP.NET project + Angular', 1, 1),
(3, 'Test plan','Write test plan', 4, 4);
```

Таблиця 4.3 – Розрахунок для методу «Перелічення параметрів для вставки»

Запит	Кількість символів	Кількість слів	ARI
«До»	131	31	13.974
«Після»	178	40	19.53

Інтерпретація результатів

Результати показали, що даний метод негативно впливає на читабельність у 1.4 разів. Це стається через те, що до запиту додаються нові слова, які є уточненням полів таблиці, що необхідні для вставки. Така практика підвищить розуміння та надійність запиту, але доведеться пожертвувати читабельністю.

4.1.4 Уникайте сортування даних

Приклад «до застосування методу»:

```
SELECT TOP(1) task_status.id
FROM task_status
JOIN task ON task.task_status_id = task_status.id
JOIN project_employee ON project_employee.id = task.project_employee_id
WHERE project.id = project_employee.project_id
ORDER BY task.id
```

Приклад «після застосування методу»:

```
SELECT MIN(task_status.id)
FROM task_status
JOIN task ON task.task_status_id = task_status.id
JOIN project_employee ON project_employee.id = task.project_employee_id
WHERE project.id = project_employee.project_id
```

Таблиця 4.4 – Розрахунок для методу «Уникайте сортування даних»

Запит	Кількість символів	Кількість слів	ARI
«До»	184	41	20.208
«Після»	170	36	18.812

Інтерпретація результатів

Результати показали, що даний метод позитивно впливає на читабельність у 1.074 разів. Це стається через те, що з методу видаляються зайві сортування даних, та тому запит стає легше читати, бо він скоротився.

4.2 Уточнення даних

4.2.1 Уточнення імені об’єкту

Приклад «до застосування методу»:

```
SELECT MIN(task_status.id)
FROM task_status
JOIN task ON task_status_id = task_status.id
```

```
JOIN project_employee ON project_employee.id = project_employee_id
WHERE project.id = project_id
```

Приклад «після застосування методу»:

```
SELECT MIN(dbo.task_status.id)
FROM dbo.task_status
JOIN dbo.task ON dbo.task_status_id = dbo.task_status.id
JOIN dbo.project_employee ON dbo.project_employee.id =
dbo.task.project_employee_id
WHERE dbo.project.id = dbo.project_employee.project_id
```

Таблиця 4.5 – Розрахунок для методу «Уточнення імені об’єкту»

Запит	Кількість символів	Кількість слів	ARI
«До»	147	32	16.207
«Після»	196	44	21.585

Інтерпретація результатів

Результати показали, що даний метод негативно впливає на читабельність у 1.331 разів. Це стається через те, що до об’єктів додаються їх джерела, які перенавантажують читача уточнюючими словами. З іншого боку уточнення даних надає більше інформації про об’єкти БД.

4.2.2 Додавання або видалення квадратних дужок для ідентифікаторів

Приклад «до застосування методу»:

```
SELECT @employee_position, COUNT(employee_id)
FROM project_employee
GROUP BY @employee_position;
```

Приклад «після застосування методу»:

```
SELECT [@employee_position], COUNT(employee_id)
FROM project_employee
GROUP BY [@employee_position];
```

Таблиця 4.6 – Розрахунок для методу «Додавання або видалення квадратних дужок для ідентифікаторів»

Запит	Кількість символів	Кількість слів	ARI
«До»	79	13	13.692
«Після»	79	13	13.693

Інтерпретація результатів

Результати показали, що даний метод ніяк не впливає на читабельність. Це стається через те, що до ідентифікаторів додаються квадратні дужки, які візуально підкреслюють слова, але ніяк їх не доповнюють і не перенавантажують.

4.2.3 Додавання точки з комою

Приклад «до застосування методу»:

```
SELECT employee_position, COUNT(employee_id)
FROM project_employee
GROUP BY employee_position

SELECT MIN(task_status.id)

FROM task_status
JOIN task ON task_status_id = task_status.id
JOIN project_employee ON project_employee.id = project_employee_id

WHERE project.id = project_id
```

Приклад «після застосування методу»:

```
SELECT employee_position, COUNT(employee_id)
FROM project_employee
GROUP BY employee_position;

SELECT MIN(task_status.id)

FROM task_status
JOIN task ON task_status_id = task_status.id
JOIN project_employee ON project_employee.id = project_employee_id

WHERE project.id = project_id;
```

В даному прикладі розглядається запит «До» візуально у якості одного запиту (речення), а запит «Після», який розділений точкою з комою, у якості двох.

Таблиця 4.7 – Розрахунок для методу «Додавання точки з комою»

Запит	Кількість символів	Кількість слів	ARI
«До»	226	45	24.725
«Після»	226	45	13.475

Інтерпретація результатів

Результати показали, що даний метод позитивно впливає на читабельність у 1.83. Це стається через те, що візуально запити можна розділити точкою з комою. У разі, якщо не розділяти їх – запити, які погано отформатовані та оформлені складно візуально розділити один від одного.

4.2.4 Додавання псевдоніму

Приклад «до застосування методу»:

```
SELECT tasks_number_per_employee.name, AVG(tasks_number_per_employee.
tasks_number)
```

```
FROM(
    SELECT project.name, project_employee.employee_position,
COUNT(task.id) as tasks_number
    FROM project_employee
    JOIN project ON project.id = project_employee.project_id
    JOIN task ON task.project_employee_id = project_employee.id
    GROUP BY project_employee.employee_position, project.name) AS
tasks_number_per_employee
GROUP BY tasks_number_per_employee.name;
```

Приклад «після застосування методу»:

```
SELECT tasks_number_per_employee.name AS project_name,
AVG(tasks_number_per_employee.tasks_number) AS
average_tasks_number_per_employee
FROM(
    SELECT project.name, project_employee.employee_position,
COUNT(task.id) as tasks_number
    FROM project_employee
    JOIN project ON project.id = project_employee.project_id
    JOIN task ON task.project_employee_id = project_employee.id
    GROUP BY project_employee.employee_position, project.name) AS
tasks_number_per_employee
GROUP BY tasks_number_per_employee.name;
```

Таблиця 4.8 – Розрахунок для методу «Додавання псевдоніму»

Запит	Кількість символів	Кількість слів	ARI
«До»	363	69	37.849
«Після»	407	78	42.147

Інтерпретація результатів

Результати показали, що даний метод негативно впливає на читабельність у 1.11 разів. Це стається через те, що до запиту додається уточнююча інформація, про значення об’єкта БД. Додаткові слова негативно впливають на читабельність, але для швидшого розуміння сенсу запиту деталі важливі.

4.2.5 Надання оператору повного імені

Приклад «до застосування методу»:

```
SELECT employee.lastname, task.title, project.name project_name
FROM project_employee
JOIN task ON task.project_employee_id = project_employee.id
JOIN project ON project_employee.project_id = project.id
LEFT employee ON employee_id = employee.id
```

Приклад «після застосування методу»:

```
SELECT employee.lastname, task.title, project.name AS project_name
FROM project_employee
INNER JOIN task ON task.project_employee_id = project_employee.id
INNER JOIN project ON project_employee.project_id = project.id
OUTER LEFT JOIN employee ON employee_id = employee.id
```

Таблиця 4.9 – Розрахунок для методу «Надання оператору повного імені»

Запит	Кількість символів	Кількість слів	ARI
«До»	200	38	22.359
«Після»	221	43	24.277

Інтерпретація результатів

Результати показали, що даний метод негативно впливає на читабельність у 1.09 разів. Це стається через те, що до запиту додаються слова для доповнення імені оператора, що використовується у запиті.

4.3 Спрощення складних запитів

4.3.1 Виділити як Common Table Expression (CTE)

Приклад «до застосування методу»:

```
SELECT tasks_number_per_employee.name,
AVG(tasks_number_per_employee.tasks_number)
FROM(
    SELECT project.name, project_employee.employee_position,
COUNT(task.id) as tasks_number
    FROM project_employee
    JOIN project ON project.id = project_employee.project_id
    JOIN task ON task.project_employee_id = project_employee.id
    GROUP BY project_employee.employee_position, project.name) AS
tasks_number_per_employee
GROUP BY tasks_number_per_employee.name;
```

Приклад «після застосування методу»:

```
WITH tasks_number_per_employee AS
(SELECT project.name, project_employee.employee_position, COUNT(task.id) as
tasks_number
FROM project_employee
JOIN project ON project.id = project_employee.project_id
JOIN task ON task.project_employee_id = project_employee.id
GROUP BY project_employee.employee_position, project.name)

SELECT tasks_number_per_employee.name,
AVG(tasks_number_per_employee.tasks_number)
FROM  tasks_number_per_employee
GROUP BY tasks_number_per_employee.name;
```

Візуально відокремлений запит за допомогою CTE можна представити у якості окремого запиту (речення), тому у запиті «Після» буде два речення.

Таблиця 4.10 – Розрахунок для методу «Виділити як Common Table Expression»

Запит	Кількість символів	Кількість слів	Кількість речень	ARI
«До»	363	69	1	37.849
«Після»	389	74	2	21.829

Інтерпретація результатів

Результати показали, що даний метод позитивно впливає на читабельність у 1.733 разів. Це стається через те, що запит умовно поділяється на основний запит та підзапити, які інкапсульовано у CTE. Візуально підзапит стає самостійним запитом, що розглядається як окреме речення.

4.3.2 Заміна алгоритму об’єднання таблиць

Приклад «до застосування методу»:

```
SELECT project.name, project_employee.employee_position,
COUNT(employee_id)
FROM project_employee, project
WHERE employee_position = ‘developer’ AND project.id =
project_employee.project_id
GROUP BY project_employee.employee_position, project.name;
```

Приклад «після застосування методу»:

```
SELECT project.name, project_employee.employee_position,
COUNT(employee_id)
FROM project_employee
JOIN project ON project.id = project_employee.project_id
```

WHERE employee_position = ‘developer’

GROUP BY project_employee.employee_position, project.name;

Таблиця 4.11 – Розрахунок для методу «Заміна алгоритму об’єднання таблиць»

Запит	Кількість символів	Кількість слів	ARI
«До»	204	33	24.186
«Після»	207	34	24.246

Інтерпретація результатів

Результати показали, що даний метод майже не впливає на читабельність. Це стається через те, що конструкція замінюється іншою майже з такою ж сигнатурою, але замінена конструкція з використання фільтру не є доречною практикою.

4.4 Спрощення умовних виразів

4.4.1 Використання WHERE замість HAVING

Приклад «до застосування методу»:

SELECT employee.firstname +' '+ employee.lastname as name, COUNT(task.id) AS Count

FROM project_employee

JOIN task ON task.project_employee_id = project_employee.id

JOIN employee ON employee_id = employee.id

WHERE task.task_status_id <= 3

GROUP BY employee.firstname +' '+ employee.lastname

HAVING employee.firstname +' '+ employee.lastname LIKE(%M);

Приклад «після застосування методу»:

SELECT employee.firstname +' '+ employee.lastname as name, COUNT(task.id) AS Count

FROM project_employee

JOIN task ON task.project_employee_id = project_employee.id

JOIN employee ON employee_id = employee.id

WHERE task.task_status_id <= 3 AND employee.firstname +' '+ employee.lastname LIKE(%M)

GROUP BY employee.firstname +' '+ employee.lastname;

Таблиця 4.12 – Розрахунок для методу «Використання WHERE замість HAVING»

Запит	Кількість символів	Кількість слів	ARI
«До»	270	51	29.005
«Після»	267	51	28.728

Інтерпретація результатів

Результати показали, що даний метод позитивно впливає на читабельність у 1.01 разів. Це стається через те, що прибирається оператор, який був використаний не за призначенням. Метод майже не впливає на читабельність, тому можна говорити лише про доречність використання операторів.

4.4.2 Заміна LIKE на оператор рівності

Приклад «до застосування методу»:

SELECT project.name, project_employee.employee_position, COUNT(employee_id)

FROM project_employee

JOIN project ON project.id = project_employee.project_id

WHERE employee_position IN (‘developer’, ‘QA’, ‘manager’) AND employee.firstname +' '+ employee.lastname LIKE(‘Mike Miller’)

GROUP BY project_employee.employee_position, project.name;

Приклад «після застосування методу»:

SELECT project.name, project_employee.employee_position, COUNT(employee_id)

FROM project_employee

JOIN project ON project.id = project_employee.project_id

WHERE employee_position IN (‘developer’, ‘QA’, ‘manager’) AND employee.firstname +' '+ employee.lastname = ‘Mike Miller’

GROUP BY project_employee.employee_position, project.name;

Таблиця 4.13 – Розрахунок для методу «Заміна LIKE на оператор рівності»

Запит	Кількість символів	Кількість слів	ARI
«До»	268	45	29.121
«Після»	264	44	28.83

Інтерпретація результатів

Результати показали, що даний метод позитивно впливає на читабельність у 1.01 разів. Це стається через те, що прибирається оператор LIKE, який був використаний не за призначенням. Метод майже не впливає на читабельність, тому можна говорити лише про доречність використання операторів.

4.4.3 Перетворення умови на IN вираз

Приклад «до застосування методу»:

SELECT project.name, project_employee.employee_position, COUNT(employee_id)

FROM project_employee

JOIN project ON project.id = project_employee.project_id

WHERE employee_position = ‘developer’ OR employee_position = ‘QA’ OR employee_position = ‘manager’

GROUP BY project_employee.employee_position, project.name;

Приклад «після застосування методу»: використання IN оператора.

```
SELECT project.name, project_employee.employee_position,
COUNT(employee_id)
FROM project_employee
JOIN project ON project.id = project_employee.project_id
WHERE employee_position IN (‘developer’, ‘QA’, ‘manager’)
GROUP BY project_employee.employee_position, project.name;
```

Таблиця 4.14 – Розрахунок для методу «Перетворення умови на IN вираз»

Запит	Кількість символів	Кількість слів	ARI
«До»	254	42	27.83
«Після»	218	37	24.821

Інтерпретація результатів

Результати показали, що даний метод позитивно впливає на читабельність у 1.12 разів. Це стається через те, що прибираються дубльовані ідентифікатори об’єктів, що використовувалися при фільтруванні за умовою.

4.4.4 Використання EXIST або JOIN замість IN

Приклад «до застосування методу»: запит з використанням IN

```
SELECT *
FROM Customers
WHERE ID IN (
    SELECT CustomerID
FROM Orders)
```

Приклад «після застосування методу»: запит з використанням EXIST

```
SELECT *
FROM Customers
WHERE EXISTS (
    SELECT *
```

```
FROM Orders
WHERE Orders.CustomerID = Customers.ID)
```

Приклад «після застосування методу»: запит з використанням INNER JOIN

```
SELECT Customers.*
FROM Customers
JOIN Orders ON Customers.ID = Orders.CustomerID
```

Таблиця 4.15 – Розрахунок для методу «Використання EXIST або JOIN замість IN»

Запит	Кількість символів	Кількість слів	ARI
«До»	54	10	9.004
«Після з EXIST»	78	13	13.33
«Після з JOIN»	67	11	12.758

Інтерпретація результатів

Результати показали, що даний метод негативно впливає на читабельність. Це стається через те, що змінюється сигнатура конструкцій на більш довгі.

4.5 Робота з коментарями

4.5.1 Додавання або видалення коментарів

Приклад «до застосування методу»:

```
-- to do: use one code style
SELECT CountTasks.name, AVG(CountTasks.Count) -- add alias
FROM(
-- to do: extract as CTE
    SELECT project.name, project_employee.employee_position,
COUNT(task.id) as Count -- change alias name
    FROM project_employee
    JOIN project ON project.id = project_employee.project_id
```

```
        JOIN task ON task.project_employee_id = project_employee.id
        GROUP BY project_employee.employee_position, project.name) AS
CountTasks
    GROUP BY CountTasks.name;
```

Приклад «після застосування методу»:

```
    -- Description: Calculate the average number of tasks on each project per each
employee

    WITH tasks_number_per_employee AS
    (SELECT project.name, project_employee.employee_position, COUNT(task.id)
AS tasks_number
        FROM project_employee
            JOIN project ON project.id = project_employee.project_id
            JOIN task ON task.project_employee_id = project_employee.id
        GROUP BY project_employee.employee_position, project.name)

    SELECT tasks_number_per_employee.name AS project_name,
    AVG(tasks_number_per_employee.tasks_number) AS
average_tasks_number_per_employee
    FROM tasks_number_per_employee
    GROUP BY tasks_number_per_employee.name;
```

Таблиця 4.16 – Розрахунок для методу «Додавання або видалення коментарів»

Запит	Кількість символів	Кількість слів	ARI
«До»	361	71	38.018
«Після»	504	96	27.298

Інтерпретація результатів

Результати показали, що даний метод позитивно впливає на читабельність, адже коментарів стало менше. Читабельність покращилася, бо стало менше тексту для читання, але погіршилося розуміння тексту запиту без додаткової інформації.

Висновки до четвертого розділу

Дослідивши результати п’ятого розділу можна зробити висновок, що багато методів рефакторингу тільки заважають на шляху до покращення читабельності. Лише деякі будуть покращувати стан читабельності, а саме:

— уникання сортування даних – 1.074;

— додавання точки з комою – 1.83;

— виділити як Common Table Expression – 1.733;

— використання WHERE замість HAVING – 1.01;

— заміна LIKE на оператор рівності – 1.01;

— перетворення умови на IN вираз – 1.12;

— видалення коментарів – 1.39.

Усі з перелічених методів покращують текст не дуже значно, але вкупі можуть давати гарні результати. Найкращі результати дають ті методи в яких використовується розділення запиту на декілька частин, які вважаються окремими реченнями. Методи, які майже не змінюють запит відповідно до розрахунку ARI, частіше за все замінюють мовні конструкції на більш доречні та органічні мові SQL.

Методи, які негативно впливають на читабельність частіше додають до самого запиту додаткову інформацію, що добре впливає на розуміння функціональності. Щоб використовувати ці методи, треба бути уважним та не вносити зайвих змін до запиту, а робити це тільки за необхідністю. Для підтримки балансу між читабельністю та інформативністю даних можна використовувати форматування даних, що буде позитивно впливати на візуальну складову запиту.

5 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

5.1 Вимоги безпеки при виконанні розрахункових робіт на робочому місці за персональним комп`ютером

У даному розділі проводиться аналіз умов безпеки праці на місці, де розроблювалася програма для замірів метрики ARI. Аналіз безпеки життєдіяльності працівників проводиться за санітарними нормами України.

Робоче місце працівника обов’язково буде складатися з комп’ютера, на якому буде виконуватися програма, та меблів для комфортної роботи, освітлення. Виходячи з набору робочого місця є ризики виникнення небезпечних ситуацій, які можуть завдати шкоди працівникам, наприклад: поганий рівень освітлення, високий рівень шуму, електричні пристрої з високою напругою, неправильна організація робочої середи, високий рівень електромагнітних випромінювань, монотонність та малорухливість праці тощо.

Для безпечної роботи працівників необхідно дотримуватися правил охорони праці та технік безпеки для роботи за комп’ютером. Для охорони праці в сфері інформаційних технологій використовуються наступні документи [1, 2, 3, 6, 8, 10, 19, 23, 25].

Перелічені документи надають список вимог до роботи з електронно-обчислювальними пристроями на робочому місці. Для безпечної роботи працівника за дисплейними терміналами треба додержуватися вимог, які прописано у документі [2]. Робота за дисплейними терміналами може сказатися на продуктивності працівника та безпосередньо на його зір. Умовами праці, при яких робота за персональним комп’ютером є безпечною:

— правильна організація робочого місця;

— нормальні значення характеристик робочого місця (шум, мікроклімат, освітлення, тощо);

— нормальні значення інформаційних характеристик взаємодії людини з електронно обчислювальною машиною.

Для забезпечення безпеки роботи за персональним комп’ютером виконуються наступні вимоги, які виконуються кожен день перед початком роботи:

- огляд робочого місця на виявлення ознак пошкоджень. У разі виявлення – звернутися до безпосереднього керівника та повідомити;
- відрегулювати освітлення на робочому місці та переконатися у відсутності відблисків на екрані комп’ютера та зустрічного світла;
- очистити екран від забруднень;
- перевірити загальну організацію робочого місця.

Під час роботи за комп’ютером треба додержуватися мінімальних вимог до мікроклімату пов’язані з нервово-емоційним напруженням. відповідно до ДСН 3.3.6.042-99 [6]: температура повітря 22-24 град.С, відносна вологість 60-40%, швидкість руху повітря не більш 0,1 м/сек.

Електронні пристрої не мають становити загрози для працівників. Під час роботи за комп’ютером забороняється

— самостійно налагоджувати техніку на робочому місці без спеціаліста, який відповідний за дану роботу;
— тримати біля комп’ютера носіїв інформації, що не використовується в поточній роботі;
— використання комп’ютеру, якщо дисплей монітору не працює стабільно;
— доторкатися до системного блоку при включеному живленні;
— приймання їжі та напоїв на робочому місці;
— потрапляння вологи до складових персонального комп’ютера.

У разі закінчення роботи за комп’ютером необхідно вимикати його та відключати від електричної мережі.

5.1.1 Шкідливі фактори на робочому місці

Шкідливі виробничі фактори – фактори, систематичний вплив яких на працюючого у визначених умовах продовж певного часу спричинить захворювання,

зменшення працездатності. У залежності від рівня факторів впливу, шкідливі фактори можуть визначатися як небезпечні для життя.

Основні шкідливі фактори на робочому місці для інженера-програміста пов’язані з використанням комп’ютеру.

При написанні програми «ARI metrics for SQL» було проведено багато часу за персональним комп’ютером, тому можна виділити шкідливі фактори, пов’язані з безпечністю розробки:

— напруження зору;
— нервово-емоційне напруження;
— напруження опорного апарату;
— недостатня кількість фізичної активності;
— недостатня кількість природнього освітлення;
— підвищений рівень електромагнітного випромінювання;
— підвищений рівень яскравості екранів.

Для зниження впливу шкідливих факторів робоче місце інженера-програміста має відповідати нормам, наведеним у документах [2-3]. Далі у наступних підрозділах описані гранично допустимі характеристики для приміщення робочого місця, освітленості, шуму та вібрацій, мікроклімату.

5.1.2 Характеристики робочого місця

Характеристики приміщень робочого простору мають відповідати [3] документу. Площа робочого місця на одну людину повинна становити не менше ніж 6 м^2, а об’єм не менше 20 м^3. Розміщення не має бути у підвальних та цокольних поверхах будівлі.

Приміщення, де була розроблена програма для дослідження відповідає стандартам та має наступні характеристики:

— кількість робітників: 1 людина;
— площа приміщення: 24м^2;

— висота приміщення: 2.6 м;

— об’єм приміщення: 62.4 м^3.

В приміщенні відсутні умови, що можуть призвести до надзвичайних ситуацій, тому приміщення відноситься до звичайного виду та відповідає стандартам.

5.1.3 Освітлення

Освітлення встановлене в приміщенні є штучним. Даний вид освітлення не несене шкідливих впливів на стан працівника, якщо воно правильно спроектоване. Освітлення впливає на працездатність, продуктивність, психічний стан людини, фізичний стан, зокрема на зір працівника.

Невідповідність правилам освітлення робочого місця може призвести до погіршення функцій зору людини, втомленості, роздратування, тощо.

Робітник також може підтримувати свій зір самостійно. Для цього можна робити перерви на декілька хвилин, щоб відпочити від яскравості дисплеїв. Під час перерви корисно робити вправи на зір. Для безпосередньої роботи за комп’ютером користуватися окулярами, які поглинають частину синього світла екрану.

При правильному підборі обладнання, проектуванні світла можна досягнути належного рівня безпеки. При виконанні роботи освітленість повинна знаходитися у межах від 200 до 400 Лк. Освітленість робочого місця складає 205 Лк, що відповідає нормам. Відповідно до ДБН В.2.5-28-2018 у робочому приміщенні є природне і штучне освітлення.

Для розрахунку світового потоку штучного освітлення на робочому місці у приміщені була застосована формула:

$$F = \frac{E*K*S*Z}{\eta}, \quad (5.1)$$

де F – світловий потік, що розраховується, Лм;

E – нормована мінімальна освітленість, Лк; E = 300 Лк;

S – площа приміщення (у нашому випадку S = 24м^2);

Z – відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1,1... 1,2, візьмемо значення Z = 1,1);

К – коефіцієнт запасу, що враховує зменшення світлового потоку освітлювачів в процесі експлуатації (значення варіюється в залежності від характеру робіт, типу приміщення та періоду робочого часу, в нашому випадку К = 1,5);

η – коефіцієнт використання світлового потоку. Виражається відношенням світлового потоку, що потрапляє на робочу поверхню, до загального потоку всіх освітлювачів, і обчислюється в долях одиниці. Залежить від характеристик освітлювача, об'єму приміщення, кольору і фактурних характеристик поверхонь стін і стелі, що характеризуються коефіцієнтами відбиття від стін ($\rho_{ст.}$) і стелі ($\rho_{стелі}$)), в нашому випадку значення коефіцієнтів дорівнюють $\rho_{ст}$ = 50% і $\rho_{стелі}$= 30%.

Розрахуємо індекс за формулою 5.2:

$$I = \frac{S}{h(A+B)}, \tag{5.2}$$

де S – площа приміщення, S = 24м^2;

h – розрахункова висота стелі, h = 2,6 м;

A – ширина приміщення, A = 4 м;

B – довжина приміщення, B = 6 м.

$$I = \frac{24}{2,6*(4+6)} = 0,92.$$

Отримавши індекс приміщення I, за таблицею із документу [54] знаходимо $ŋ = 0,35$.

Підставимо всі значення у формулу для визначення світлового потоку F:

$$F = \frac{300*1,5*24*1,1}{0,35} = 33942 \text{ Лм}$$

Для освітлення використані освітлювачі з люмінесцентними лампами типу ЛБ 40-1, світловий потік яких складає F = 4320 Лм. Розрахуємо мінімальну необхідну кількість світильників за формулою 5.3:

$$N = \frac{F}{Fл}, \tag{5.3}$$

де N – кількість ламп, що визначається;

F - світловий потік, прийняте значення F = 33942 Лм;

$F_л$ - світловий потік лампи, $F_л$ = 4320 Лм.

$$\mathrm{N} = \frac{33942}{4320} = 8$$

У приміщенні розташовані світильники типу відкритий дволамповий (ВД). Кожен світильник комплектується двома лампами. Загалом необхідно обладнати приміщення щонайменше 4 світильниками із 8 працюючими лампами в них.

5.1.4 Мікроклімат

Для тривалих робіт в приміщенні повинні підтримуватися характеристики, які є допустимими для мікроклімату. Параметри мікроклімату, які є нормальними для перебування в них наведені у таблиці 5.1 з документу ДСН 3.3.6.042-99:

Таблиця 5.1 – Параметри мікроклімату для приміщень з комп'ютерами

Період року	Параметри мікроклімату	Величина
Холодний	Температура повітря в приміщені	22 – 24 °С
	Відносна вологість	40 – 60%
	Швидкість руху повітря	До 0,1 м/с
Теплий	Температура повітря в приміщені	23 – 25
	Відносна вологість	40 – 60 %
	Швидкість руху повітря	0,1 – 0,2 м/с

Показниками мікроклімату у приміщені, де розроблювалася програма для дослідження порівнювалися відповідно до холодного періоду року. Показники у приміщенні наступні:

— Температура повітря в приміщені: 22 °С;

— Відносна вологість: 45%;

— Швидкість руху повітря: 0 м/с.

У приміщені розташовано два джерела тепловиділення: персональний комп’ютер та опалювальна система з використанням радіаторів. Всі показники відповідають нормам у холодний період року.

5.1.5 Шум та вібрація

Зайві звуки та вібрації негативно сприяють робочому процесу людини та завдають їй шкоди. Робітники, які працюють в шумних умовах зазвичай страждають від головних болей, дратівливості, стомленості, болю у вухах, тощо. Погіршується не тільки здоров’я людини, але шум та вібрація може ускладнювати комунікацію між працівниками в одному приміщені.

Допустимий рівень шуму на робочому місці встановлюється наказом о ДСН 3.3.6.039-99 [10]. Значення до 55 дБА гучності шуму та до 33дБ рівня вібрацій є нормальними. У приміщенні, де розроблювалася програма рівень шуму становить 40 дБА, а рівень вібрацій – 30 дБ. Основним джерелом шуму та вібрацій є комп’ютер. Для зниження рівня шуму можна використати звукопоглинальні матеріали, але рівні шуму та вібрацій у приміщені є припустимі нормам.

5.2 Дії працівників у надзвичайних ситуаціях

Неможна точно сказати, які надзвичайні ситуації точно можуть статися під час роботи у приміщені, але можна назвати найочікуванішу – це ураження електричним струмом. Користуючись наказом Міністерства охорони здоров’я України «Порядок надання домедичної допомоги постраждалим при ураженні електричним струмом та блискавкою» [11] можна скласти список дій допомоги постраждалому при розглядаємій надзвичайній ситуації:

— переконатися, що більше нікому не загрожує небезпека;

— при можливості припинити дію електричного струму на постраждалого;

— провести огляд постраждалого, визначити наявність ознак життя та свідомості;
— викликати швидку медичну допомогу;
— у разі відсутності дихання постраждалого– розпочати серцево-легеневу реанімацію;
— якщо постраждалий без свідомості – надайте йому стабільне положення
— накласти стерильну пов’язку на уражені місця;
— забезпечити нагляд за постраждалим до приїзду швидкої;
— при погіршенні стану постраждалого зателефонуйте диспетчеру швидкої медичної допомоги.

Обов’язково робити кожен крок швидко, адже кожна хвилина може коштувати постраждалому життя. Якщо постраждалий здається мертвим – не припиняйте серцево-легеневу реанімацію, адже біологічна смерть може ще не настала та є шанс на рятування.

Для безпечної роботи у приміщені, де потенційно є загроза пожежної небезпеки треба додержуватися наступних правил прописаних у документі «Про затвердження Правил пожежної безпеки» [23].

До дій, які треба робити під час виникнення пожежі у робочому приміщенні відносяться наступні:

- повідомити службу пожежної безпеки за номером 101, включити пожежну сигналізацію. При необхідності викликати інші оперативні служби;
- повідомити керівництво про пожежу;
- власноруч вжити заходи гасіння пожежі за допомогою вогнегасника;
- евакуюйтесь з приміщення на безпечну територію. У разі задимлення пересувайтеся повзком та прикладіть до дихальних шляхів мокру тканину, якщо це можливо.

Додаткові рекомендації до основних дій:

- відкриваючи двері переконайтеся, що ручка двері не тепла та за дверями немає вогню. Завжди закривайте двері для меншого розповсюдження вогню;
- не намагайтеся забрати майно, краще скоріше евакуюватися;
- якщо ви не можете самостійно вибратися з приміщення – приверніть до себе увагу.

Висновки до п’ятого розділу

У даному розділі був проведений аналіз документів про безпеку працівника на робочому місці, відповідність характеристик приміщення, у якому розроблювалася програма, до норм, які затверджені державними установами.

В результаті можна зробити висновок, що робоче місце задовольняє вимогам охорони праці та безпеки життєдіяльності. У ході розрахунків було підтверджено, що освітлення, шум та вібрація, мікроклімат є у границях норми.

Для безпеки у надзвичайних ситуаціях пов’язаних з ураженням електричним струмом була складена інструкція домедичної допомоги, яка готова до споживання. Також було розроблено план дій при виникненні пожежі на робочому місці.

ЗАГАЛЬНІ ВИСНОВКИ

У дипломній роботі було досліджено тему рефакторингу мови запитів SQL, проаналізовано сучасний стан проблеми, були розроблені методи, які сприяють покращенню читабельності та розуміння функціоналу запиту, розроблено програму, яка розраховує метрику ARI.

Під час аналізу предметної області було досліджено розглядаєму тему. Було проаналізовано літературу та програмні додатки. Висвітлено проблеми розглядаємих джерел. Наразі рефакторинг SQL запитів розглядається спеціалістами, але здебільшого методи направлені на зміну схеми бази даних, що унеможливлює використання цих методів для будь якої реляційної бази даних.

В ході розробки нових методів рефакторингу був використаний підхід адаптації методів рефакторингу для ООП мов. В результаті було отримано шістнадцять методів розділених на п’ять груп. Всі ці методи направлені на підвищення розуміння тексту запиту, його функціоналу та покращення читабельності.

У ході розробки програми для розрахунку метрики ARI, яка підтвердить зміни у характеристиці читабельності, були використані сучасні технології та підходи до розробки програмного забезпечення.

Графічний інтерфейс користувача програмного застосунку розроблено таким чином, що він є простим, інтуїтивно зрозумілим, адаптованим до різних розмірів екранів, гнучкий для налаштування.

Оцінка розроблених методів рефакторингу показала, що не всі методи можуть позитивно сказатися на читабельності. Такі результати наразі показують, те що акцент у розроблених методах зроблено на підвищення розуміння функціоналу запиту, але це негативно впливає на легкість прочитання. Отже, однозначно рекомендовано використовувати ті методи, які показали позитивний вплив на читабельність.

Було проведено аналіз робочого місця на предмет безпеки праці за нормативними документами, які призначено для інженерів-програмістів. Розрахунки

показали, що місце відповідає нормам для використання працівником та програмний застосунок у даній середі розробляти безпечно.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Закон України «Про охорону праці» від 21.11.2002 р. Редакція № 220-IV

2. НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» затверджені наказом Міністерства соціальної політики України від 14 лютого 2018 №207

3. ДСанПіН 3.3.2.007-98 Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин затверджені постановою Головного державного санітарного лікаря України від 10.12.1998 № 7

4. НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» затверджені наказом Міністерства соціальної політики України від 14 лютого 2018 №207

5. Примірна інструкція з охорони праці під час експлуатації електронно-обчислювальних машин, затверджена наказом Міністерства доходів і зборів України від 05.09.2013 № 443

6. ДСН 3.3.6.042-99 Державні санітарні норми мікроклімату виробничих приміщень затверджені постановою Головного державного санітарного лікаря України від 1 грудня 1999 року № 42

7. ДСН 239-96 Державні санітарні норми і правила захисту населення від впливу електромагнітних випромінювань затверджені Міністерством охорони здоров'я від 1 серпня 1996 року №239

8. ДБН В.2.5-28:2018 Державні будівельні норми України. Природне і штучне освітлення затверджені наказом Міністерства регіонального розвитку, будівництва та житлово-комунального господарства України від 3 жовтня 2018 року №264

9. Наказ Міністерства охорони здоров’я «Про затвердження Державних санітарних норм допустимих рівнів шуму в приміщеннях житлових та громадських будинків і на території житлової забудови» від 22 лютого 2019 року №463

10. ДСН 3.3.6.039-99 Державні санітарні норми виробничої загальної та локальної вібрації затверджені постановою Головного санітарного лікаря України від 1 грудня 1999 року №39

11. Порядок надання домедичної допомоги постраждалим при ураженні електричним струмом та блискавкою №398, затверджений Міністерством охорони здоров’я України від 16 червня 2014

12. Кодекс цивільного захисту України. Редакція від 17.06.2021

13. Кодекс законів про працю України від 10 грудня 1971 р. № 322-VIII. Редакція від 15.07.2021

14. Закон України від 23 вересня 1999 р. № 1105-XIV “Про загальнообов`язкове державне соціальне страхування”. Редакція від 15.07.2021

15. Постанова Кабінету Міністрів України від 01 серпня 1992 р. № 442 «Про затвердження Порядку проведення атестації робочих місць за умовами праці». Редакція від 05.10.2016

16. Постанова Кабінету Міністрів України “Про затвердження Порядку розслідування та обліку нещасних випадків, професійних захворювань та аварій на виробництві” №337. Редакція від 05.01.2021

17. НПАОП 0.00-4.12-05 Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці, затверджене наказом Держнаглядохоронпраці від 26.01.2005 №15. Редакція від 30.01.2017

18. НПАОП 0.00-7.14-17 Вимоги безпеки та захисту здоров’я під час використання виробничого обладнання працівниками, затверджене наказом Міністерства соціальної політики України від 28.12.2017 № 2072. Зареєстрованого в Мін’юсті України 23.01.2018 за №97/31549

19. ДСТУ 7299:2013 Дизайн і ергономіка. Робоче місце оператора. Взаємне розташування елементів робочого місця. Загальні вимоги ергономіки, затверджено та

введено в дію наказом міністерства економічного розвитку і торгівлі України 14.10.2013 № 1231. Зареєстрованого в Мін'юсті України 14.02.2012 за №226/20539

20. «Перша медична долікарська допомога» [Ел. ресурс]. URL: https://www.bsmu.edu.ua/blog/6893-persha-medichna-dolikarska-dopomoga/.Дата звернення: 01.12.2021.

21. «Інструкція з першої медичної допомоги» [Ел. ресурс]. URL: https://www.sop.com.ua/article/263-nstruktsya-z-nadannya-domedichno-dopomogi. Дата звернення: 01.12.2021.

22. «Перша медична допомога при ураженні електричиним струмом» [Ел. ресурс]. URL: https://bozhedarivska-selrada.gov.ua/news/1576497483/. Дата звернення: 01.12.2021.

23. НАПБ А.01.001-2014 Правила пожежної безпеки в Україні, затверджений наказом Міністерства внутрішніх справ України від 30.12.2014 № 1417 і зареєстрований у Міністерстві юстиції України 05.03.2015 за № 252/26697

24. Державні будівельні норми України «Природне і штучне освітлення» ДБН В.2.5-28:2018, затверджені наказом Міністерства регіонального розвитку, будівництва та житлово-комунального господарства України 03.10.2018 № 264, введено в дії з 01.03.2019

25. Санітарні норми виборничого шуму, ультразвуку та інфразвуку ДСН 3.3.6-037-99, затверджені наказом Міністерства Охорони Здоров'я України 01.12.1999 №37

26. Л.В. Дементий, А.Л. Юсина «Обеспечение безопасности жизнедеятельности»: учб. посібник – Краматорськ: ДГМА, 2008. – 300 с.

27. Фаулер М., Бек К., Брант Д., Опайдак У., Робертс Д. Рефакторинг: Улучшение проекта существующего кода. – Санкт-Петербург: ООО «Диалектика», 2019 – 448 с.

28. Bohm L., Refactoring legacy T-SQL for improved performance: Modern practices for SQL Server applications – Apress, Berkeley, CA, 2020 – 236 c.

29. Code Readability Management of High-level Programming Languages: A Comparative Study, International Journal of Advanced Computer Science and Applications / Muhammad U.T., Muhammad B.B., Muhammad B., Adnan S. – Пакістан, 2020 – 8 с.

30. Buse, R.PL., Westley R.W., A metric for software readability, Proceedings of the 2008 international symposium on Software testing and analysis, pp. 121-130, Seattle, WA, USA, July 20-24, 2008.

31. Donald K. B., Oracle Silver Bullets: Real-world Oracle Permormance Secrets – Rampant Techpress, 2005 – 200 с.

32. Senter, R.J.; Smith, E.A. (November 1967). «Automated Readability Index». Wright-Patterson Air Force Base: iii. AMRL-TR-6620. Retrieved March 18, 2012.

33. S. Fakhoury, D. Roy, A. Hassan and V. Arnaoudova, «Improving Source Code Readability: Theory and Practice», 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC), pp. 2-12, Montreal, QC, Canada, 2019.

34. Pahal, Ankit, and Rajender S. Chillar. «Code Readability: A Review of Metrics for Software Quality», International Journal of Computer Trends and Technology (IJCTT) – Volume 46 Number 1- April 2017

35. Маерс Г. Искусство тестирования программ / Пер. с анг. Гузикевич А. – М.: Вильямс, 2012. – 272

36. Якість програмного забезпечення та тестування [Текст]: методичні вказівки до лабораторних робіт / уклад.: В. І. Шинкаренко, О. С. Куроп’ятник, Г. В. Забула, Д. О. Пєтін, Є. В. Лукін, Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во ПФ «Стандарт-Сервіс», 2018. – 50 с.

37. Макконнелл С. М15 Совершенный код. Мастер-класс / Пер. с англ. — М. : Издательство «Русская редакция», 2010. — 896 стр.

38. Буч Г., Рамбо Д., Якобсон И. Б90 Язык UML. Руководство пользователя. 2-е изд.: Пер. с англ. Мухин Н. – М.: ДМК Пресс, 2006. – 496 с.

39. «Рефакторинг» [Ел. ресурс]. URL: https://refactoring.guru/ru/refactoring. Дата звернення: 20.11.2021

40. «Индекс Колман – Лиау» [Ел. ресурс]. URL: https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81

_%D0%9A%D0%BE%D0%BB%D0%BC%D0%B0%D0%BD_%E2%80%94_%D0%9B%D0%B8%D0%B0%D1%83. Дата звернення: 01.11.2021

41. «Автоматический индекс удобночитаемости» [Ел. ресурс]. URL: https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D0%B4%D0%B5%D0%BA%D1%81_%D1%83%D0%B4%D0%BE%D0%B1%D0%BE%D1%87%D0%B8%D1%82%D0%B0%D0%B5%D0%BC%D0%BE%D1%81%D1%82%D0%B8. Дата звернення: 01.11.2021

42. «Gunning fog index» [Ел. ресурс]. URL: https://en.wikipedia.org/wiki/Gunning_fog_index. Дата звернення: 01.11.2021

43. «Flesch-Kincaid readability tests» [Ел. ресурс]. URL: https://en.wikipedia.org/wiki/Flesch%E2%80%93Kincaid_readability_tests. Дата звернення: 01.11.2021

44. «SQL Promt» [Ел. ресурс]. URL: https://www.apexsql.com/sql-tools-refactor.aspx. Дата звернення: 10.02.2021

45. «ApexSQL» [Ел. ресурс]. URL: https://www.red-gate.com/products/sql-development/sql-prompt/. Дата звернення: 10.02.2021

46. «Принципы для разработки: KISS, DRY, YAGNI, BDUF, SOLID, APO и бритва Оккама» [Ел. ресурс]. URL: https://habr.com/ru/company/itelma/blog/546372/. Дата звернення: 10.02.2021

47. «Windows Forms overview (Общие сведения о Windows Forms)» [Ел. ресурс]. URL: https://docs.microsoft.com/ru-ru/dotnet/desktop/winforms/windows-forms-overview?view=netframeworkdesktop-4.8. Дата звернення: 11.11.2021

48. «C# documentation» [Ел. ресурс]. URL: https://docs.microsoft.com/en-us/dotnet/csharp/. Дата звернення: 11.11.2021

49. Джоунс, Эйри, Стивенз, Райан К., Плю, Рональд Р., Гарретт, Роберт Ф., Кригель, Алекс. Функции SQL. Справочник программиста. : Пер. с англ. – М. : ООО «И.Д. Вильямс», 2007. – 768 с. : ил. – Парал. тит. англ.

50. Радченко, Г.И. Объектно-ориентированное программирование / Г.И. Радченко, Е.А. Захаров. – Челябинск: Издательский центр ЮУрГУ, 2013. – 167 с.

51. «Unit testing and the Arrange, Act and Assert (AAA) Pattern» [Ел. ресурс]. URL: https://medium.com/@pjbgf/title-testing-code-ocd-and-the-aaa-pattern-df453975ab80. Дата звернення: 21.11.2021

52. «Рекомендации по модульному тестированию для .NET Core и .NET Standart» [Ел. ресурс]. URL: https://docs.microsoft.com/ru-ru/dotnet/core/testing/unit-testing-best-practices. Дата звернення: 21.11.2021

ДОДАТКИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор
Українського державного
університету науки і технологій
Борис БОНДАР

ІНСТРУМЕНТ РОЗРАХУНКІВ МЕТРИКИ ARI ДЛЯ МОВИ ЗАПИТІВ SQL

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01211-01-ЛЗ

Завідувач кафедри КІТ

_____ проф. Вадим ГОРЯЧКІН

Керівник розробки

_____ доц. Олександр ІВАНОВ

Виконавець

_____ Анастасія РИЖКОВА

Нормоконтролер

_____ доц. Олена КУРОП'ЯТНИК

2021

ЗАТВЕРДЖЕНО
1116130.01211-01

ІНСТРУМЕНТ РОЗРАХУНКІВ МЕТРИКИ ARI ДЛЯ МОВИ ЗАПИТІВ SQL

Технічне завдання
1116130.01211-01

Аркушів 22

2021

АНОТАЦІЯ

Документ 01116130.01211-01 «Інструмент розрахунку метрики ARI для мови запитів SQL. Технічне завдання» входить до складу програмної документації до програми, яка реалізує програмне забезпечення «ARI Metric for SQL».

У даному документі представлені призначення та область застосування програмної системи, вимоги, стадії, строки виконання проекту та техніко-економічні відомості.

ЗМІСТ

Вступ 5

1 Підстава для розробки 6

2 Призначення розробки 7

2.1 Функціональне призначення 7

2.2 Експлуатаційне призначення 7

3 Вимоги до програми 8

3.1 Вимоги до функціональних характеристик 8

3.2 Вимоги надійності 8

3.3 Умови експлуатації 9

3.4 Вимоги до складу і параметрів технічних засобів 9

3.5 Вимоги до інформаційної та програмної сумісності 10

3.6 Вимоги до маркування і упаковки 10

3.7 Вимоги до транспортування і зберігання 10

4 Вимоги до програмної документації 11

5 Визначення витрат на проектування програми 12

6 Стадії та етапи розробки 20

7 Порядок контролю та прийому 21

Бібліографічний список 22

ВСТУП

Інформаційні технології – це важлива складова повсякденного життя, тому наразі створюється та вдосконалюються все більше і більше програмних продуктів. Велика кількість спеціалістів з інформаційних технологій залучається до проектів, які тільки почали розроблюватися або вже існують багато років та потребують нарощування функціоналу або підтримки. Для ефективної роботи спеціалістів необхідно підтримувати тексти програм у зрозумілому людині вигляді, адже здебільшого текст програми не пишеться, а читається. Для покращення читабельності та зрозумілості тексту програми необхідно застосовувати рефакторинг коду. Набагато легше читати текст програми на якому був проведений рефакторинг, тому розробники проводять даний етап регулярно по завершенню завдання або по гострій необхідності, як окрему задачу на проекті. Сам процес рефакторингу призваний для удосконалення тексту програми без зміни функціональності продукту. Методи рефакторингу використовуються для очищення тексту програми та мінімізації можливостей появи нових помилок.

Наразі в багатьох проектах використовуються реляційні бази даних, де за допомогою СКБД та мови запитів SQL можна маніпулювати даними (вставляти, оновлювати, видаляти та обирати). Так як запити є частиною загального тексту програми є потреба у методах рефакторингу для мови запитів, але наразі даний підхід мало розкритий у спеціалістів інформаційних технологій.

Для оцінки доцільності використання методів рефакторингу для мови запитів необхідний програмний продукт, який би порівнював метрику читаємості для запиту, який ще не був оброблений методами рефакторингу та на якому вже були застосовані.

Програмний продукт «Інструмент розрахунку метрики ARI для мови запитів SQL» призначено для порівняння метрики ARI, яка показує характеристику читабельності тексту. Подібних застосунків ще немає, особливо для мови запитів, яка має свій власний синтаксис.

Ключові слова: Рефакторинг, SQL, ARI, читабельність.

1 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є навчальний план спеціальності 121 «Інженерія програмного забезпечення», затверджений ректором Українського державного університету науки і технологій проф. Пшіньком О.М.

Тема дипломної роботи: Аналіз можливостей та пристосування методів рефакторингу до запитів на мові SQL.

Керівник дипломної роботи: доц. Іванов О.П.

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

2.1 Функціональне призначення

Функціональним призначенням розробки є:

— візуалізація зміни запитів за допомогою методів рефакторингу;

— автоматичний розрахунок метрики читабельності ARI, кількості слів, кількості символів, кількості запитів;

— відображення отриманих результатів, порівняння результатів.

2.2 Експлуатаційне призначення

Експлуатаційне призначення полягає у можливості автоматизувати розрахунок метрики читаємості ARI для мови запитів SQL, що пришвидшить розглянутий процес дослідження.

3 ВИМОГИ ДО ПРОГРАМИ

3.1 Вимоги до функціональних характеристик

Програмний застосунок, що розроблюється розрахований для використання на комп’ютері та спроектовано з можливостями розширення функціоналу.

Вимоги до функціональних характеристик програмного забезпечення:

— можливість відкривати файл з форматом .sql з фізичного носія;

— можливість написання програми вручну у текстовому редакторі;

— можливість розраховувати метрику ARI для запитів «до» та «після» редагування.

Формат вхідних даних – файл з розширенням SQL, який є текстовим та містить у собі запити розділені точкою з комою. Вхідними даними також можуть бути тексти запитів розділені точкою з комою, що написані вручну у програмному застосунку, або скопійовані до програмного застосунку з іншого джерела.

Формат вихідних даних – дані розрахунків метрики ARI, кількість символів, кількість слів, кількість запитів для запиту «до» та «після» редагування методами рефакторингу, висновок виражений у дробовому значенні, що показує наскільки змінилася метрика ARI у відношенні розрахунку метрики для запиту «до» до запиту «після».

3.2 Вимоги надійності

Вимоги до надійності програмного застосунку будуть враховувати лише ті ситуації, які безпосередньо зв’язані з роботою застосунку.

Основні вимоги надійності:

— програма має проводити перевірку даних перед їх споживанням;

— наявність резервної копії програми на зовнішньому носії;

— збій у роботі додатку не впливає на подальшу роботу застосунку;

3.3 Умови експлуатації

Для нормального функціонування програмного застосунку рекомендовано виконання наступних вимог:

- програма повинна використовуватися на ЕОМ, де робоче місце відповідає умовам мікроклімату:
  - температура 21° – 25°С;
  - вологість 40 - 60%;
- ЕОМ повинні відповідати вимогам стандартів чинних в Україні, нормативним актам з охорони праці [2];
- стан технічних засобів повинен бути безпечним до їх використання за нормами та стандартами;

Для забезпечення запуску програмного продукту необхідно дотримуватися наступних умов:

- програма призначена для роботи на ЕОМ з операційною системою Windows XP і вище;
- для роботи з програмою необхідні базові навички роботи з персональним комп’ютером та операційною системою Windows;

3.4 Вимоги до складу і параметрів технічних засобів

Програмний продукт розрахований для функціонування на персональному комп’ютері, що має такі характеристики для виконання та встановлення:

- жорсткий диск, об’ємом вільного місця не менше 100 Мб;
- центральний процесор з тактовою частотою 1ГГц та більше;
- оперативна пам’ять об’ємом 4096 Мб;
- маніпулюючий пристрій миша;
- клавіатура;
- монітор 800*600 і вище;
- USB-роз’єм стандарту 3.0 і вище або наявність CD/DVD приводу.

3.5 Вимоги до інформаційної та програмної сумісності

Вимогою до інформаційної та програмної сумісності є використання операційної системи Windows XP та вище.

3.6 Вимоги до маркування і упаковки

Програмний застосунок має маркуватися як зображено на рисунку 3.6.1.

Сервіс «ARI metric for SQL», версія 1.0
Рижкова Анастасія Андріївна
Кафедра КІТ, 2021

Рисунок 3.6.1 – Маркувальний штамп продукту

3.7 Вимоги до транспортування і зберігання

Транспортування повинно здійснюватися за наступними способами:

- розповсюдження програмного застосунку через інтернет;
- розповсюдження програмного застосунку на CD/DVD дисках;
- розповсюдження програмного застосунку за допомогою USB накопичувачів.

Зберігання не обмежене програмним продуктом.

4 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу програмної документації входять наступні документи, що повинні додержуватися вимогам державного стандарту до оформлення документів:

- специфікація;
- текст програми;
- опис програми;
- керівництво користувача. Керівництво з розрахунку ARI.

5 ВИЗНАЧЕННЯ ВИТРАТ НА ПРОЕКТУВАННЯ ПРОГРАМИ

Основна мета розробки техніко-економічного обґрунтування (ТЕО) – дати фінансову оцінку передбачуваних витрат та одержуваного корисного результату, а також оцінити прибутковість проекту і, в кінцевому підсумку, економічну доцільність його розробки та впровадження.

Початковим етапом розрахунку величини трудових витрат розробників є оцінка розміру програмного забезпечення. Основні відмінності методик, що застосовуються в оцінці трудовитрат, полягають у використовуваному типі критерію оцінки якості [4].

Згідно моделі COCOMO, розмір проекту S вимірюється в рядках коду LOC (KLOC), а трудовитрати в людино-місяцях. Використані формули містяться у документі [1].

$$E = a \cdot S^b \cdot EAF, \qquad (5.1)$$

де E – витрати праці на проект (в людино-місяцях);

S^b – розмір коду (в KLOC);

EAF – фактор уточнення витрат (effort adjustment factor).

Для простих систем, a = 2,4; b = 1,05

Розмір програмного коду було підраховано за допомогою інструменту вбудованного у Visual Studio 2019 Community – Code Metrics Results.

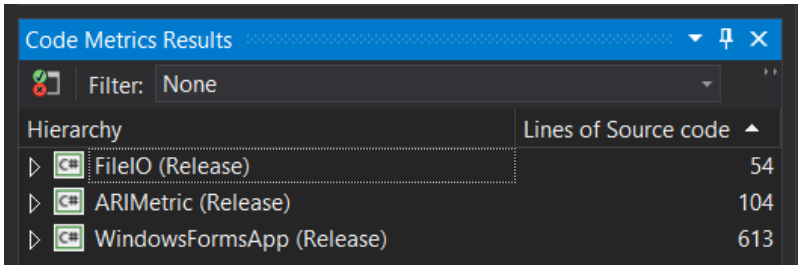


Рисунок 5.1 – Підрахунок розміру програмного коду

Отже розмір програмного коду становить 771 рядки:

$$E = 2{,}4 \cdot 0{,}771^{1{,}05} \cdot 1 = 1{,}64$$

Отже, згідно моделі COCOMO, орієнтовні трудовитрати на проект складуть приблизно 1,64 людино-місяці.

Нижче наведені розрахунки вартості розробки «Автоматизована система оцінки схожості програм». Основними статтями витрат прийняті:

– основна заробітна плата;
– відрахування на соціальні потреби;
– накладні витрати;
– витрати на персональний комп’ютер і ліцензійні базові програмні засоби.

Основна заробітна плата (ОЗП) оцінює працю інженера-програміста зі створення програмного продукту і визначається виходячи з кількості розробників, часу виконання розробки (годин), а також заробітної плати в розрахунку на одну годину. Розрахунок заробітної платні проводиться по формі табл. 5.1.

Таблиця 5.1 – Фонд місячної заробітної плати

№ п/п	Посада виконавця	Оклад, грн/міс	Кількість		Сума зарплати грн
			чол	місяців	
1	інженер-програміст	15750 [2]	1	1,64	25830

Описаний в проекті програмний продукт був розроблений одним програмістом в період з 27.09.21 до 22.11.21, що складає 40 днів або приблизно 8 робочих тиждень. Витрати робочого часу прийняті за 40 годин у тиждень. Погодинна ставка кваліфікованого інженера–програміста складає 93,75 грн/год. Таким чином, витрачено робочого часу:

$$t_{\text{розробки}} = N_{\text{чол}} \times N_{\text{тиж}} \times N_{\text{год}}, \quad (5.2)$$

де $N_{\text{чол}}$ – кількість виконавців, чол;

$N_{\text{тиж}}$ – тривалість розробки;

$N_{\text{год}}$ – витрати робочого часу, год;

$$t_{\text{розробки}} = 1 \cdot 8 \cdot 40 = 320 \text{ чол/год.}$$

ОЗП визначається за формулою:

$$\text{ОЗП} = t_{\text{розробки}} \cdot N \cdot K_{KB}, \quad (5.3)$$

де $t_{\text{розробки}}$ – витрати праці у чол/год;

N – погодинна ставка;

K_{KB} – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. Коефіцієнт кваліфікації розробника (k) - ступінь підготовленості виконавця до дорученої йому роботі (він визначається залежність від стажу праці та

становить:

— для працюючих до 2 років- 0,75;

— від 2 до 3 років 1,0;

— від 3 до 5 років - 1,1-1,2;

— від 5 до 7 років - 1,3-1,4;

— понад 7 років - 1,5-1,6.

В даному випадку KKB приймається 0,75. ОЗП складає:

$$\text{ОЗП} = 320 \cdot 93{,}75 \cdot 0{,}75 = 22500 \text{ грн.}$$

Відрахування на соціальні потреби встановлюються у відсотках від суми заробітної плати (22% [4]):

$$C_{\text{соц}} = \frac{\text{ОЗП} \cdot 22\%}{100\%} \quad (5.4)$$

$$C_{\text{соц}} = \frac{22500 \cdot 22\%}{100\%} = 4950\text{грн.}$$

Отримані результати за (5.3) та (5.4) підсумовуються. Вони складають 27450грн. та визначають основні прямі витрати.

Накладні витрати враховують загальногосподарчі витрати по забезпеченню проведення роботи: витрати на опалення, електроенергію, амортизація будівель, зарплату адміністративного персоналу та інше. Вони визначаються в процентах (30 – 40%) від суми прямих витрат:

$$С_{накл} = \frac{(ОЗП + С_{соц}) \cdot 35\%}{100\%}; \quad (5.5)$$

$$С_{накл} = \frac{(22500 + 4950) \cdot 35\%}{100\%} = 9607{,}05 \text{грн.}$$

Протягом усього терміну використання нової техніки підприємство щорічно витрачає певні кошти, пов’язані з її експлуатацією.

Експлуатаційні витрати на персональний комп’ютер визначаються протягом терміну розробки програмного засобу в залежності від вартості комп’ютеру. В експлуатаційні витрати входять:

– вартість витратних матеріалів;

– витрати на ремонт;

– заробітна плата ремонтника;

– оренда приміщення;

– додаткові витрати – прибирання приміщення, охорона, оренда, комунальні послуги;

– амортизаційні витрати на персональний комп'ютер і програмне забезпечення;

– витрати на електроенергію ($С_{ел}$), які визначаються за формулою:

$$С_{ел} = P \cdot B \cdot T_{розр}, \quad (5.6)$$

де Р – потужність комп’ютера та допоміжних споживачів електричної енергії, приймається 0,45 кВт/год;

B – вартість 1 кВт/годин для непобутових споживачів,

складає 1,8 грн [3];

Tрозр – час роботи з ЕВМ, приймається рівним робочому часу.

Витрати на електроенергію визначаються так:

$$C_{\text{ел}} = 0{,}45 \cdot 1{,}8 \cdot 320 = 259{,}2 \text{ грн.}$$

Витрати на витратні матеріали ($C_{\text{вм}}$) протягом всього терміну експлуатації приблизно 10% від вартості комп'ютеру. Вартість робочої станції приймається 18 000 грн. [6], термін експлуатації – 5 років. Отже, можна визначити ці витрати за період створення програмного засобу:

$$C_{\text{вм}} = B_{КОМ} \cdot \frac{N_{\text{Д}}}{N_{\text{екпс}} \cdot 365} \cdot \frac{10\%}{100\%}, \quad (5.7)$$

де $B_{КОМ}$ – вартість персонального комп'ютеру;

$N_{\text{Д}}$ – кількість днів розробки програмного продукту;

$N_{\text{експ}}$ – термін експлуатації персонального комп'ютеру.

Витрати на витратні матеріали визначаються так:

$$C_{\text{вм}} = 18000 \cdot \frac{40}{5 \cdot 365} \cdot \frac{10}{100} = 40 \text{ грн.}$$

Заробітна плата ремонтника ($C_{\text{рем}}$) визначена наступним чином: на ремонт 50 комп'ютерів потрібен один інженер-системотехнік. Його середньомісячна заробітна плата приймається 9000 грн. Тоді в перерахунку на один комп'ютер його заробітна плата за період розробки програмного продукту складає:

$$\text{С}_{\text{рем}} = \frac{\text{С}'\text{рем}}{N_{\text{ком}}} \cdot T_{\text{міс}}, \quad (5.8)$$

де $C_{\text{рем}}^{\prime}$ – середньомісячна заробітна плата;

$N_{КОМ}$ – кількість комп'ютерів на одного ремонтника.

$T_{мес}$ – час розробки програмного продукта, міс.

Заробітна плата ремонтника ($C_{\text{рем}}$) буде складати:

$$C_{\text{рем}} = \frac{90\acute{0}0}{50} \cdot 1{,}64 = 295{,}2 \text{ грн.}$$

За статистикою витрати на комплектуючи вироби (C_{KOM}) для ремонту персонального комп'ютера складає 10% від його вартості за термін його експлуатації, тобто рівні витратам на витратні матеріали:

$$C_{KOM} = C_{BM} = 40\text{грн.} \quad (5.9)$$

Амортизаційні відрахування на персональний комп'ютер (АПК) визначені з положення, що амортизаційний період в даний час дорівнює терміну морального старіння обчислювальної техніки і складає 3 роки. Отже, за 3 роки амортизаційні відрахування на персональний комп'ютер дорівнюють вартості комп'ютера. За період проектування амортизаційні відрахування складуть:

$$\text{АКП} = \text{В}_{KOM} \cdot \frac{N_д}{N_{\text{експ}} \cdot 365}; \quad (5.10)$$

$$\text{АКП} = 18000 \cdot \frac{1{,}64}{3 \cdot 12} = 820.$$

Амортизаційні відрахування на програмне забезпечення (АПЗ) залежать від його циклу заміни. Якщо прийняти термін морального старіння для Windows 5 років та Visual Studio за 2 рік то амортизаційні відрахування на програмне забезпечення дорівнюють його вартості.

Для функціонування персонального комп'ютера використовувалася операційна система Windows 10, для написання програмного забезпечення - програмне середовище Visual Studio 2019 Community.

$$\text{АКП}_w = 13800 \cdot \frac{1{,}64}{5 \cdot 12} = 377{,}2$$

$$\text{АКП}_w = 0 \cdot \frac{1{,}64}{2 \cdot 12} = 0.$$

Розрахунок амортизаційних відрахувань на програмне забезпечення зведений в табл. 5.2. Додаткові витрати ($C_{\text{ДОД}}$): прибирання приміщень, охорона, комунальні послуги важко оцінити точно і прийняти рівними 50% заробітної плати інженера- програміст, тобто 7875 гривень на місяць.

Оренду приміщень на 10м^2 для однієї людини приймемо рівною 2500 гривень на місяць [5]. Тобто за весь період розробки – 4 100 грн. Сумарні експлуатаційні витрати на один персональний комп'ютер складають:

$$C_{\text{експ}} = C_{\text{ел}} + C_{\text{ВМ}} + C_{\text{рем}} + \text{АПК} + \text{АПО} + C_{\text{ор}} + C_{\text{дод}}; \quad (5.11)$$

$$C_{\text{експ}} = 259{,}2 + 40 + 400 + 820 + 377{,}2 + 4100 + 7875 = 13971{,}40 \text{ грн}$$

Результати розрахунків зведено у табл. 5.3.

Таблиця 5.2 – Використовуване програмне забезпечення

Найменування програмного забезпечення	Вартість програмного забезпечення, грн	Джерело придбання	Амортизаційні відрахування, грн
Windows 10	13800	http://mtsoft.kiev.ua/product/windows-10-professional	377,2
Visual Studio 2019 Community	0	https://docs.microsoft.com/en-us/visualstudio/releases/2019/release-notes	0
Всього:	13800		377,2

Таблиця 5.3 – Експлуатаційні витрати на ПК і ПЗ.

Найменування витрат	Витрати, грн
1	2
Витрати на електроенергію	259,2
Вартість витратних матеріалів	40
Витрати на ремонт	400

Продовження таблиці 5.3

1	2
Амортизація персонального комп’ютера	820
Амортизація програмного забезпечення	377,2
Оренда приміщення	4100
Додаткові витрати	7875
Всього	13971,4

Таким чином, витрати на створення програмного продукту складають:

$$C_{\text{розробки}} = \text{ОЗП} + C_{\text{соц}} + C_{\text{накл}} + C_{\text{експ}}; \quad (5.12)$$

$$C_{\text{розробки}} = 22500 + 4950 + 9607{,}05 + 13971{,}40 = 51028{,}45 \text{ грн.}$$

Розрахунок витрат зведено у табл. 5.4.

Таблиця 5.4 – Кошторис витрат на розробку програмного засобу

Найменування витрат	Витрати, грн
Основна заробітна плата	22500,00
Відрахування на соціальні потреби	4950,00
Накладні витрати	9607,05
Експлуатаційні витрати	13971,4
Всього	51028,45

За отриманими значеннями техніко-економічних показників проекту складено кошторис витрат на розробку сучасного програмного забезпечення для оцінки схожості програм. За результатами розрахунків, приблизна вартість розробки складає 51028,45 грн.

6 СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

№	Стадія розробки	Етап розробки	Термін
1	Технічне завдання	Постановка задачі	01.10.20 – 30.12.20
		Розробка формату вхідних та вихідних даних	14.01.21 – 30.01.21
		Розробка програмних вимог	03.02.21 – 28.02.21
		Затвердження технічного завдання	02.03.21 – 31.03.21
2	Робочий проект	Розробка логіки програми	01.04.21 – 30.06.21
		Розробка графічного інтерфейсу користувача	01.07.21 - 07.08.21
		Тестування та відлагодженя	10.08.21 – 31.08.21
		Розробка програмної документації	01.09.21 – 30.10.21
3	Впровадження	Підготовка та передача програми	01.11.21 – 12.12.21

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙОМУ

Контроль якості програмного продукту здійснюється за допомогою виконання процесу тестування з метою знаходження помилок та їх ліквідації.

Контроль за виконанням роботи здійснюється науковим керівником дипломної роботи.

Прийом програмного продукту та роботи здійснюється уповноваженою комісією.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Борисенков, Методика оценки трудозатрат по разработке программного обеспечения информационных систем [Ел. ресурс] /Борисенков Евгений //URL: https://dspace.enu.kz/bitstream/handle/data/12881/METODIKA-TRUDOZATRAT.pdf?sequence=1&isAllowed=y – Дата звернення: 23.11.2021.
2. «Зарплати розробників за червень-липень 2021» [Ел. ресурс] URL: https://jobs.dou.ua/salaries/#period=jun2021&city=Dnipro&title=Junior%20Software%20Engineer&language=C%23%2F.NET&spec=&exp1=0&exp2=1 – Дата звернення: 23.11.2021.
3. «Мінестрество енергетики України. Новини» [Ел. ресурс] URL: http://mpe.kmu.gov.ua/minugol/control/publish/article?art_id=245583544 – Дата звернення: 23.11.2021.
4. «Розміри ЄСВ-2021» [Ел. ресурс] URL: https://services.dtkt.ua/catalogues/indexes/13 – Дата звернення: 23.11.2021.
5. «Аренда офисов в Днепре» [Ел. ресурс] URL: https://besplatka.ua/ru/dnepropetrovsk/nedvizhimost/arenda-komercheskoy-nedvizhimosti/arenda-ofisov?prop[136][to]=2500&prop[112][to]=10&currency=UAH – Дата звернення: 23.11.2021.
6. «Комп'ютери та ноутбуки» [Ел. ресурс] URL: https://rozetka.com.ua/dell_n3001vn3500ua03_2201_ubu/p290435868/ – Дата звернення: 23.11.2021.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор
Українського державного
університету науки і технологій
Борис БОНДАР

ІНСТРУМЕНТ РОЗРАХУНКІВ МЕТРИКИ ARI ДЛЯ МОВИ ЗАПИТІВ SQL

Робочий проект
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01211-01-ЛЗ

Завідувач кафедри КІТ

_____ проф. Вадим ГОРЯЧКІН

Керівник розробки

_____ доц. Олександр ІВАНОВ

Виконавець

_____ Анастасія РИЖКОВА

Нормоконтролер

_____ доц. Олена КУРОП'ЯТНИК

2021

ЗАТВЕРДЖЕНО
1116130.01211-01-ЛЗ

ІНСТРУМЕНТ РОЗРАХУНКІВ МЕТРИКИ ARI ДЛЯ МОВИ ЗАПИТІВ SQL

Специфікація

1116130.01211-01

Аркушів 2

2021

Позначення	Найменування	Примітка
	Документація	
1116130.01211-01-ЛЗ	Лист затвердження	
1116130.01211-01 12 01-ЛЗ	Лист затвердження	
1116130.01211-01 12 01	Текст програми	
1116130.01211-01 13 01-ЛЗ	Лист затвердження	
1116130.01211-01 13 01	Опис програми	
1116130.01211-01 31 01-ЛЗ	Лист затвердження	
1116130.01211-01 31 01	Опис застосування	
1116130.01211-01 І3 01-ЛЗ	Лист затвердження	
1116130.01211-01 І3 01	Керівництво користувача. Керівництво з розрахунку ARI	

ЗАТВЕРДЖЕНО
1116130.01211-01 13 01-ЛЗ

ІНСТРУМЕНТ РОЗРАХУНКІВ МЕТРИКИ ARI ДЛЯ МОВИ ЗАПИТІВ SQL

Опис програми

1116130.01211-01 13 01

Аркушів 21

2021

АНОТАЦІЯ

Документ 1116130.01211-01 13 01 «Інструмент для розрахунків метрики ARI для мови запитів SQL. Опис програми» входить до складу програмної документації до програми, яка реалізує програмне забезпечення «ARI metric for SQL» .

У даному документі представлений опис програми: функціональне призначення, опис логічної структури, використані технічні засоби, виклик і завантаження, вхідні і вихідні дані, порядок роботи з програмою. Програми написані мовою C# з використанням бібліотеки WindowsForms для реалізації графічного інтерфейсу користувача. Об'єм пам'яті, що займають програми комплексу, складає 30 Мб. Конфігурація комп'ютера стандартна.

ЗМІСТ

1 Загальні відомості 4

2 Функціональне призначення 5

3 Опис логічної структури 6

3.1 Алгоритми програми 6

3.2 Використані методи 7

3.3 Структура програми з описом функцій складових частин і зв’язки 7

3.4 Зв’язки програми з іншими програмами 9

4 Використані технічні засоби 10

5 Виклик та завантаження 11

6 Вхідні дані 12

7 Вихідні дані 13

8 Опис призначеного для користувача інтерфейсу 14

8.1 Опис станів програми 14

8.2 Опис переходів між станами програми 15

8.3 Опис керування діалогом 15

8.4 Формування екранів 18

9 Порядок роботи з програмою 19

10 Повідомлення 20

Бібліографічний список 21

1 ЗАГАЛЬНІ ВІДОМОСТІ

Програмний продукт має назву «ARI metric for SQL» він реалізує функціонал необхідний для розрахунку метрики читабельності для порівняння результатів застосування методів рефакторингу.

Для функціонування даного продукту треба застосувати програмний засіб такий як операційна система Windows.

Програма реалізована на мові C# у програмному середовищі Visual Studio 2019 Community.

2 ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Функціональним призначенням розробки є:

— візуалізація зміни запитів за допомогою методів рефакторингу;

— автоматичний розрахунок метрики читабельності ARI, кількості слів, кількості символів, кількості запитів;

— відображення отриманих результатів, порівняння результатів.

3 ОПИС ЛОГІЧНОЇ СТРУКТУРИ

3.1Алгоритми програми

Алгоритм роботи користувача з програмним застосунком є легким для розуміння. Алгоритмом є наступний перелік дій: заповнення основної текстової області, заповнення текстової області призначеної для початкового стану запитів, обробка введених даних, виведення результатів розрахунку метрики ARI.

Більш деталізований алгоритм наведено нижче:

- до початку роботи необхідно отримати от користувача тексти запитів, які можна отримати двома шляхами:
  - відкриття файлу з текстом запитів, які розділені точкою з комою, з розширенням sql: якщо було обрано файл іншого формату – буде виведено повідомлення про помилку. Операція вважається успішною, якщо у головній текстовій області та області «до» (у разі порівняння запитів) відображається текст запитів;
  - вручну додавши текст запитів на мові SQL з розділенням запитів точкою з комою. Операція вважається завершеною, якщо користувач написав запити та додав їх на головну текстову область та область «до» (у разі порівняння запитів).
- користувач може спробувати скористатися методами рефакторингу, які були представлені у даній роботі для покращення запитів. Для цього користувач повинен працювати у головній текстовій області або у області «після»;
- після стадії підготовки запитів користувач натискає кнопку «Run ARI Metric», яка запускає функцію розрахунку метрики читабельності ARI окремо для запитів «до» та «після» рефакторингу. У разі, якщо одне з текстових областей буде порожнім – буде сформовано результат тільки для заповненої області. У разі, якщо обидві області не мають тексту запитів – будо виведено повідомлення про помилку;

— по завершенню обробки для користувача у текстовій області результатів буде виведено розрахунки з інформацією про кількість символів, слів та запитів у тексті, а також значення метрики ARI. У разі, якщо заповнено дві області: «до» та «після», буде виведено результат, який показує у скільки разів було покращено або погіршено читабельність тексту запитів.

3.2 Використані методи

При розробці даної програмної системи було використано мову C#. Мова C# підтримує об’єктно-орієнтовану парадигму програмування, тому було додержано основні принципи: інкапсуляцію, наслідування та поліморфізм.

При розробці модулю розрахунку метрики читабельності ARI була використана математична формула:

$$ARI = 4.71 * \frac{\text{Кількість символів}}{\text{Кількість слів}} + 0.5 * \frac{\text{Кількість слів}}{\text{Кількість речень}} - 21.43.$$

3.3 Структура програми з описом функцій складових частин і зв’язки

Структура програми може бути описана за допомогою UML діаграми класів. Діаграма класів віборжає сутності програмної системи та їх взаємодію між собою. У кожній сутності відображено атрибути та методи. Розглядаються тільки функціональні компоненти програмної системи, які безпосереньо пов’язані з роботою додатку. Діаграму класів зображено на рисунку 3.3.1.

Опис сутностей діграми класів представлено за допомогою CRC-карток. Картки представлені у вигляді таблиць, де зверху написано ім’я сутності, зліва – за що ця сутність відповідає, заправа – зв’язки з іншими сутностями.

Таблиця 3.3.1 – Form

Form	
Базовий клас для відображення та обробки даних інтерфейсу користувача	

Таблиця 3.3.2 – MetricForm

MetricForm	
Клас графічного інтерфейсу для відображення результатів розрахунків для текстів запитів введені користувачем	Наслідується від Form Зв’язок «композиція» з MetricForm

Таблиця 3.3.3 – ARI

ARI	
Клас для розрахунку метрики читабельності ARI для мови SQL	Зв’язок «композиція» з MetricForm

Таблиця 3.3.4 – FileIO

FileIO	
Клас для читання та зберігання текстових файлів формату sql	Зв’язок «асоціація» з MetricForm

Таблиця 3.3.5 – Program

Program	
Клас для запуску програмного застосунку	

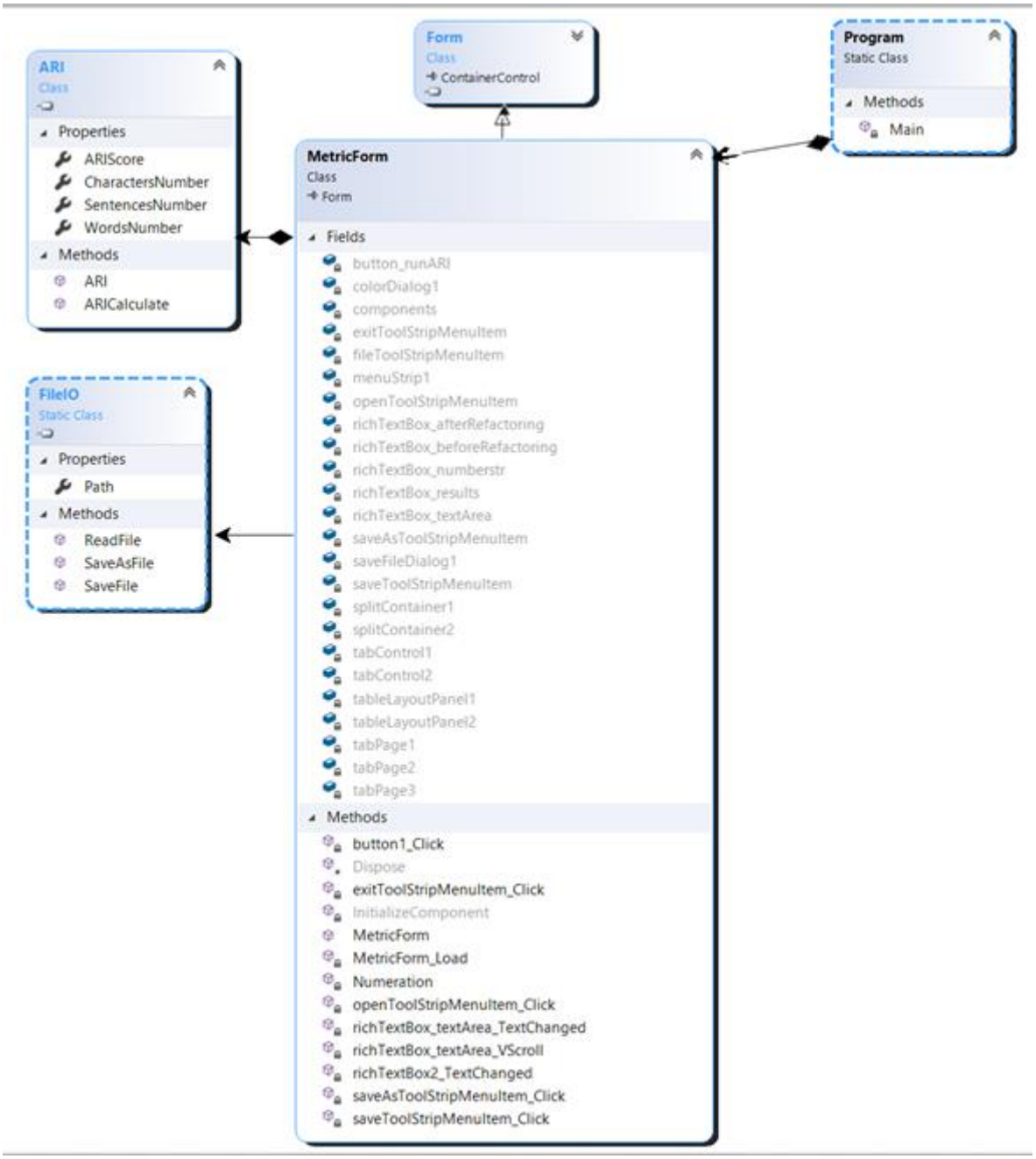


Рисунок 3.3.1 – Діаграма класів

3.4 Зв’язки програми з іншими програмами

Даний програмний застосунок може працювати автономно, але залежить від середовища операційної системи Windows. У разі, якщо операційна система не може забезпечити стабільну роботу, то програмний застосунок також не зможе предоставити розроблений набір функцій.

4 ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Для використання продукту необхідно мати комп’ютер відповідний до нормального використання операційної системи Windows.

Програмний продукт розрахований для функціонування на персональному комп’ютері, що має такі мінімальні характеристики для виконання та встановлення:

- жорсткий диск, об’ємом вільного місця не менше 100 Мб;
- центральний процесор з тактовою частотою 1ГГц та більше;
- оперативна пам’ять об’ємом 4096 Мб;
- маніпулюючий пристрій миша;
- клавіатура;
- монітор 800*600 і вище;
- USB-роз’єм стандарту 3.0 і вище або наявність CD/DVD приводу.

5 ВИКЛИК ТА ЗАВАНТАЖЕННЯ

Для виклику програмного застосунку необхідно виконати запуск файлу ARIMericForSQL.exe. Програмний застосунок потрібно запускати лише у операційній системі Windows. Розмір програми становить 30Мб.

Після завантаження програми перед користувачем повинно з’явитися вікно, яке зображено на рисунку 5.1.

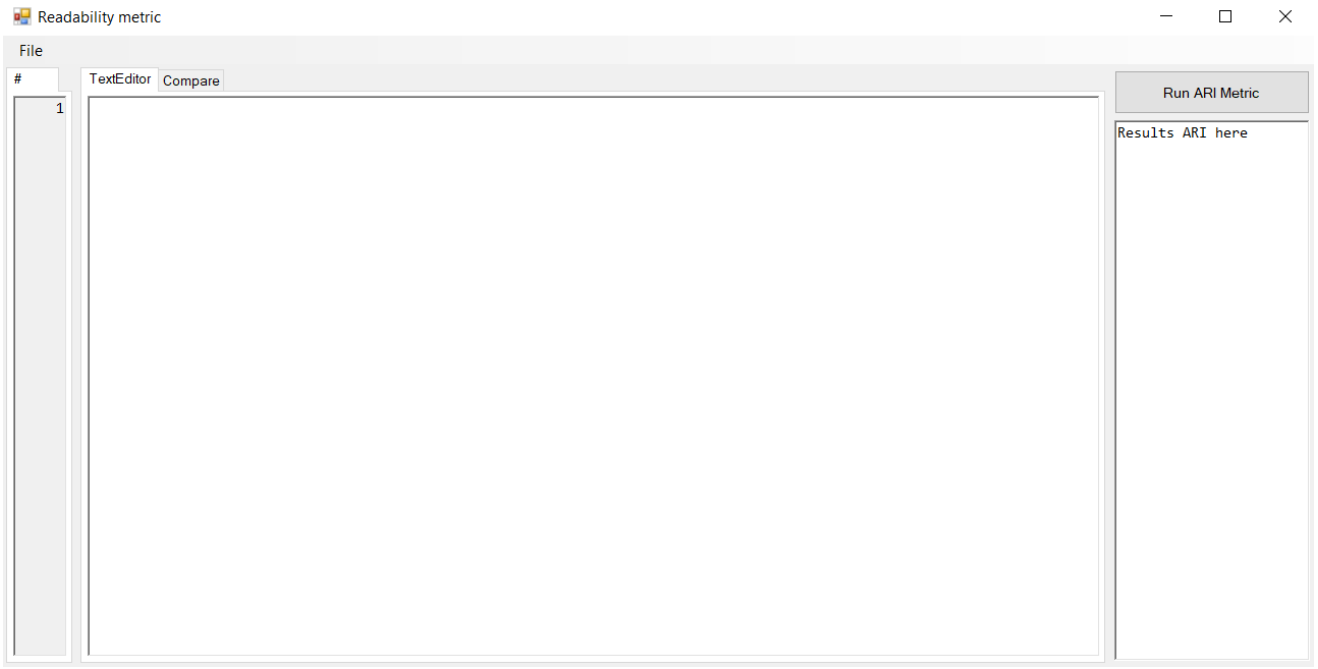


Рисунок 5.1 – Головне вікно програми

6 ВХІДНІ ДАНІ

Вхідними даними у програмі є:

— шлях до файлу sql формату з текстом запитів на мові SQL розділеними точкою з комою;

— текст запитів до якого були застосовані методи рефакторингу.

7 ВИХІДНІ ДАНІ

Вихідними даними у програмі є:

— результат розрахунку для порівняння, у наступному вигляді:

- ARI значення для тексту «до»;
- кількість символів для тексту «до»;
- кількість слів для тексту «до»;
- кількість запитів для тексту «до»;
- ARI значення для тексту «після»;
- кількість символів для тексту «після»;
- кількість слів для тексту «після»;
- кількість запитів для тексту «після»;

— результати розрахунку для одної текстової області «до» чи «після»:

- ARI значення для тексту запитів;
- кількість символів для тексту запитів;
- кількість слів для тексту запитів;
- кількість запитів для тексту запитів;

— отредагований запит можна зберегти у відкритий документ або у новий.

8 ОПИС ПРИЗНАЧЕНОГО ДЛЯ КОРИСТУВАЧА ІНТЕРФЕЙСУ

8.1 Опис станів програми

Стани у яких може перебувати програма описані в таблиці 8.1.1

Таблиця 8.1.1 – Повідомлення користувача

№ стану	Стан	Опис стану	Рекомендовані дії
1	2	3	4
1	Завантаження програмного застосунку	Програма завантажується до ОЗУ	Чекати на повну загрузку програмного застосунку
2	Програма з незаповненими даними	Програма запущена та готова до роботи	Починати працювати з додатком
3	Програма отримує дані для заповнення текстових областей	Користувач наповнює програмний застосунок даними	Ввести всі необхідні дані до прогмного застосунку
4	Програма з усіма заповненими даними	Користувач завершив введення даних	Натисніть кнопку «Run ARI metric» для початку обробки текстів
5	Програма у процесі розрахунку метрики	Програма розраховує для всіх заповнених текстових областей метрику ARI	Дочекайтеся виведення результатів

Продовження таблиці 8.1.1

1	2	3	4
6	Виведення результатів	Програма вивела результати на графічний інтерфейс	Перегляньте отримані результати та за необхідності повторіть експеремент або завершіть програму
7	Збереження тексту запитів до файлу	Програма зберігає текст запиту за вказаним шляхом або в поточний документ	Перегляньте отрриманий файл

8.2 Опис переходів між станами програми

Схема переходів між станами програми представлена на рисунку 8.2.1.

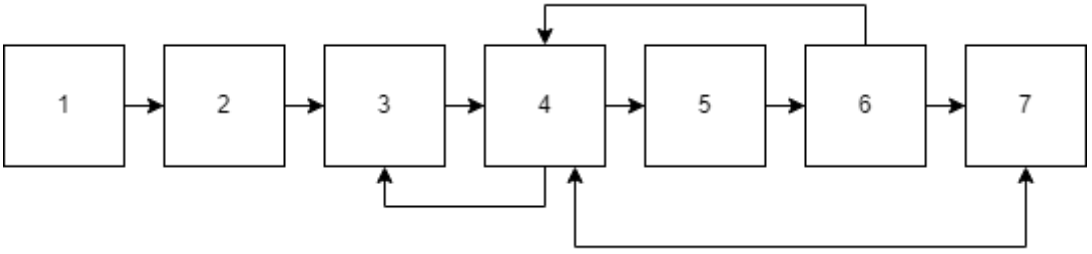


Рисунок 8.2.1 – Схема переходів між станами програми

Вихід з програми можливий за станами: 2, 3, 4, 6, 7.

8.3 Опис керування діалогом

Робота програми виконується у два етапи:

— підготовка даних до почтаку екперементу;

— виведення результатів експеременту.

На рисунку 5.1 представлене головне вікно програми, яке відображається одразу пусля запустку застосунку. Щоб почати підготовку до проведення

експеременту необхідно ввести текст запиту, до якого ще не було затосовано методів рефакторингу. Набраний вручну текст треба внести до області «до» для порівняння результатів розрахунків з розрахунками для області «після».

У разі, якщо користувач хоче відкрити файл з текстом запитів, можна скористатися функцією відриття файлів з верхнього меню у вкладці «File»., як показано на рисунку 8.3.1

Рисунок 8.3.1 – Відкриття файлу

Перед користувачем з’явиться вікно з каталогом файлів на ком’ютері, де повинен знайти потрібний файл на натиснути кнопку відкриття. Приклад діалогового вікна представлено на малюнку 8.3.2. У разі, якщо користувач спробує відкрити файл не sql формату, то на екрані з'явиться повідомлення, яке відображено на рисунку 8.3.3.

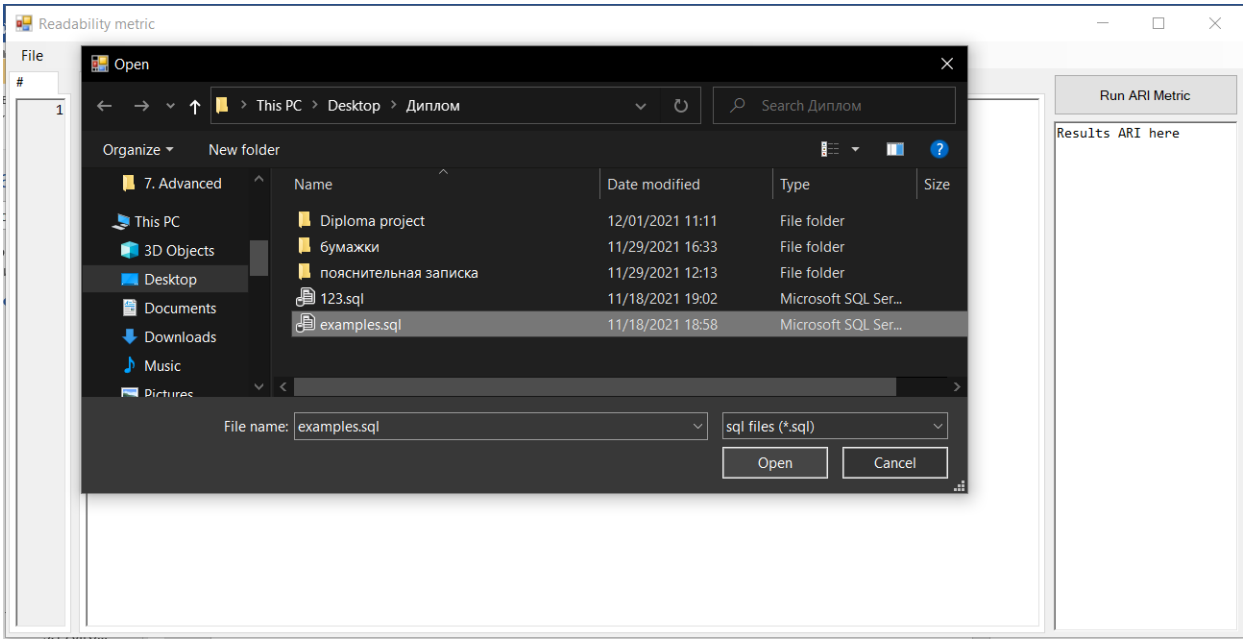


Рисунок 8.3.2 – Діалогове вікно для шляху файлу

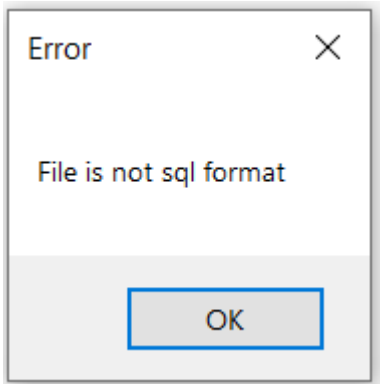


Рисунок 8.3.3 – Повідомлення при спробі відкриття файлу не з форматом sql

Таким чином наповнення відкритого файлу одразу буде завантажено до області «до» та головній для редагування.

Після цього користувач повинен провести рефакторинг запитів на свій розсуд. Робиться дай процес вручну.

Після завершення підготовки треба рохрахувати метрику ARI натиснувши кнопку «Run ARI Metric». Через деякий час результати з’являться у області, що розташована одразу під кнопкою. Приклад результатів наведно на рисунку 8.3.4. У разі, якщо користувач розпочав розрахунок метрики без належного заповнення текстових областей, то перед ним з’явиться помилка зображена на рисунку 8.3.5.

Рисунок 8.3.4 – Виведення розрахунків

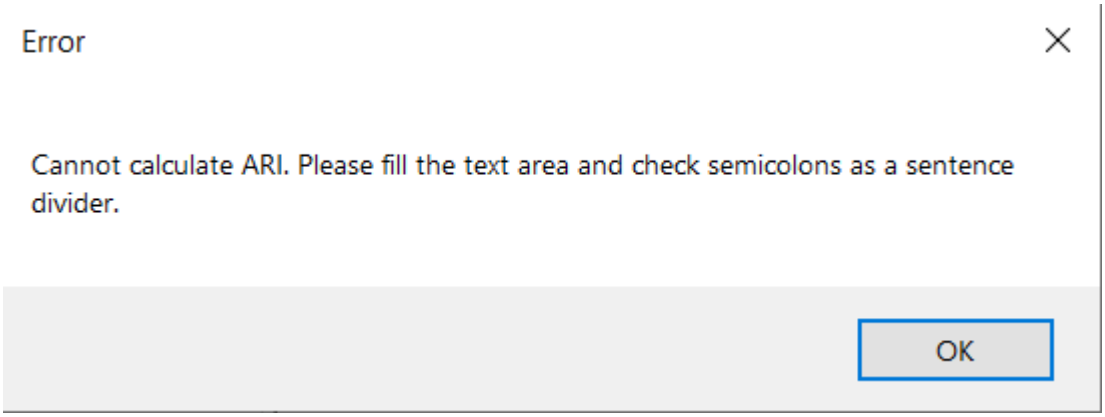


Рисунок 8.3.5 – Помилка виконання розрахунку метрики

Для збереження нового тексту запитів необхідно скористатися операціями збереження: «Save» – збереження до поточного файлу та «Save as» – збереження до нового файлу через діалогове вікно як на малюнку 8.3.2. Знайти ці операції можна у верхньому меню, як показано на рисунку 8.3.1.

8.4 Формування екранів

Програма має головне вікно, яке має у собі вкладки: для головної області редагування на першій та вікна для порівняння запитів на другій. Також програмний застосунок має два види вікна, які описуєть помики представлені на рисунках 8.3.3 та 8.3.5. У програмному застосунку є діалогове вікно, що відповідає за формування шляху файлу.

Головний вид програми був наведений на рисунку 5.1, інші види головного вікна зображена на рисунках 8.3.1 та 8.3.4.

9 ПОРЯДОК РОБОТИ З ПРОГРАМОЮ

У разі виникнення труднощів пов’язаних з продуктом «ARI Metric for SQL», можна звернутися за адресою електроної пошти ariforsql@tracking.com.

Підтримка та опрацювання заяв з 9:00 до 18:00 по буднях.

10 ПОВІДОМЛЕННЯ

У таблиці 10.1 описані повідомлення користувачу з перекладом на державну мову, що можуть з’явитися під час роботи з програмним застосунком.

Таблиця 10.1 – Повідомлення користувача

Текст повідомлення	Опис ситуації	Рекомендовані дії
Файл не sql формату	Користувач намагається завантажити файл, який не є sql	Завантажити файл sql формату або набрати текст вручну
Неможливо розрахувати ARI. Будь-ласка, заповніть текстові області та перевірте точки з комою, які є розділювачами речень	Користувач намагається розрахувати метрику без заповнення текстових областей даними	Заповними хоча б одну з текстових областей призначену для зберігання тексту запитів
Поточний шлях є не валідним	Користувач намагається зберігти файл у поточний до якого більше немає доступу за вказаним раньше шляхом	Збережіть новий файл
Файл всже існує. Ви хочете замінити його?	Користувач намагається зберегти файл у вже існуючий	Натисніть «так» у разі, якщо ви хочете замінити файл, «ні» – якщо не хочете.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Пышкин Е.В. Основные концепции и механизмы объектно-ориентированного программирования. – СПб.: БХВ-Петербург, 2005. – 640 с.

ЗАТВЕРДЖЕНО
1116130.01211-01 І3 01-ЛЗ

ІНСТРУМЕНТ РОЗРАХУНКІВ МЕТРИКИ ARI ДЛЯ МОВИ ЗАПИТІВ SQL

Керівництво користувача. Керівництво з розрахунку ARI

1116130.01211-01 І3 01

Аркушів 14

2021

АНОТАЦІЯ

Документ 01116130.01211-01 І3 01 «Інструмент розрахунку метрики ARI для мови запитів SQL. Керівництво користувача. Керівництво з розрахунку ARI» входить до складу програмної документації до програми, яка реалізує програмне забезпечення «ARI Metric for SQL».

У даному документі представлені призначення та умови застосування, підготовка до роботи, опис операцій, аварійні ситуації та рекомендації до засвоєння.

ЗМІСТ

Вступ 4

1 Призначення та умови застосування 5

2 Підготовка до роботи 6

3 Опис операцій 7

3.1 Відкриття файлу з текстом запитів 7

3.2 Редагування тексту запитів у текстових областях 8

3.3 Розрахунок метрики читабельності ari 10

3.4 Збереження файлу з текстом запитів, який було редаговано методами рефакторингу 11

4 Аварійні ситуації 13

5 Рекомендації щодо засвоєння 14

ВСТУП

Інструмент для дослідження дозволяє розрахувати метрику ARI та зробити висновки щодо доцільності впровадження обраних методів рефакторингу.

Сфера застосування: рефакторинг SQL запитів, робота з реляційними базами даних.

Користувачі – розробники, які використовують реляційні бази даних, мовою запитів якої є SQL.

Користувач повинен мати навички розоби з операційною системою Windows та досвід роботи з персональним комп’ютером.

Адміністратор повинен мати навички встановлення та налаштування програмного застосунку у операційній системі Windows.

Практичне значення результатів поляє у перевірці доцільності проведенного рефакторингу.

1 ПРИЗНАЧЕННЯ ТА УМОВИ ЗАСТОСУВАННЯ

Інструмент для дослідження дозволяє перевірити методи рефакторингу на доцільність застосування, якщо головною метою рефакторингу стоїть задача покращення читабельності тексту запитів. Для точної оцінки впливу методів рефакторингу необхідно застосувати даний програмний застосунок.

Умови використання програмного застосунку:

- жорсткий диск, об’ємом вільного місця не менше 100 Мб;
- центральний процесор з тактовою частотою 1ГГц та більше;
- оперативна пам’ять об’ємом 4096 Мб;
- маніпулюючий пристрій миша;
- клавіатура;
- монітор 800*600 і вище;
- USB-роз’єм стандарту 3.0 і вище або наявність CD/DVD приводу;
- операційна система Windows.

2 ПІДГОТОВКА ДО РОБОТИ

На фізичному носії даних є виконувальний файл ARIforSql.exe за допомогою якого адміністратор запускає та встановлює програму ан персональний комп’ютер.

Для запуску програми необхідно скористатися запцском файлу ARIforSql.

3 ОПИС ОПЕРАЦІЙ

Інструмент для дослідження складається з наступних операцій:

— відкрийте файл з текстом запитів;

— редагуйте текст запитів у текстових областях;

— розраховуйте метрики читабельності ARI;

— збережіть файл з текстом запитів, який було редаговано методами рефакторингу.

3.1 Відкриття файлу з текстом запитів

Користувач для заповнення даними застосунку може скористатися відкриттям файлу або зробити це вручну. У разі, якщо користувач бажає відкрити файл формату sql з текстом запитів, можна скористатися функцією відриття файлів з верхнього меню у вкладці «File», як показано на рисунку 3.3.1

Рисунок 3.3.1 – Відкриття файлу

Перед користувачем з’явиться вікно з каталогом файлів на ком’ютері, де повинен знайти потрібний файл на натиснути кнопку відкриття. Приклад діалогового вікна представлено на малюнку 3.3.2. У разі, якщо користувач спробує відкрити файл не sql формату, то на екрані з'явиться повідомлення, яке відображено на рисунку 8.3.3.

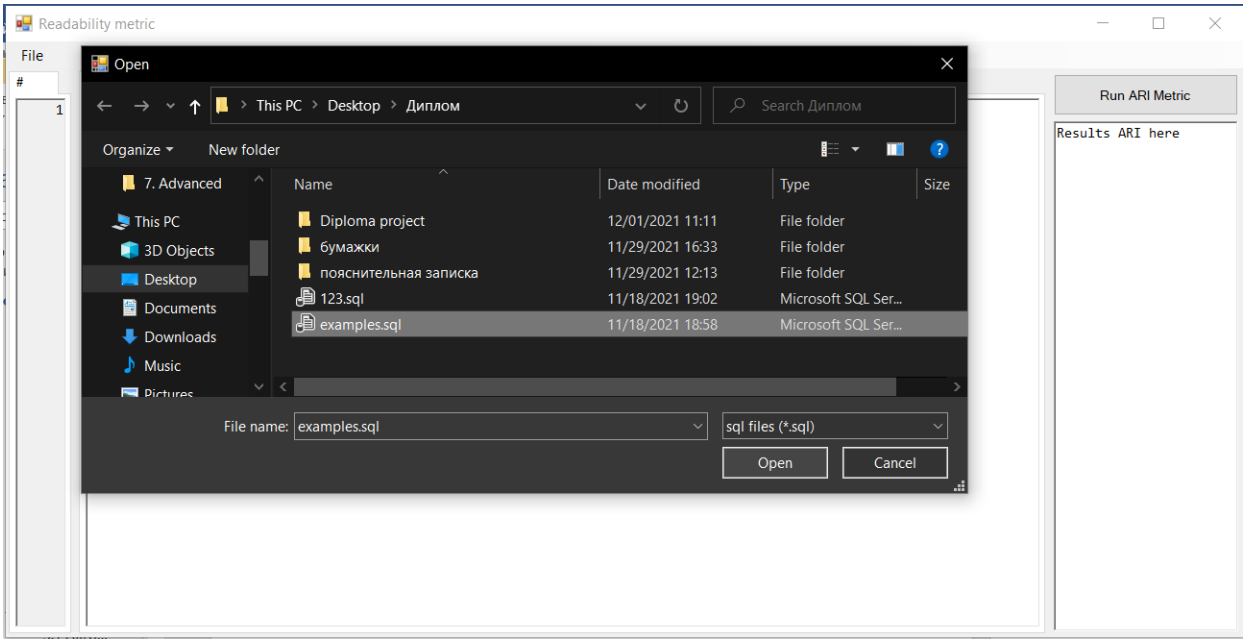


Рисунок 3.3.2 – Діалогове вікно для шляху файлу

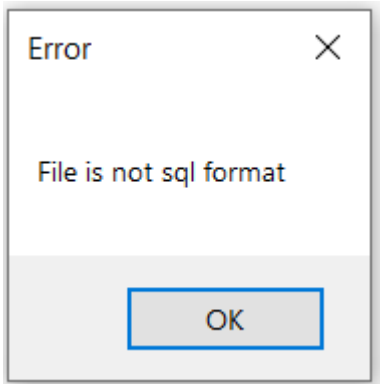


Рисунок 3.3.3 – Повідомлення при спробі відкриття файлу не з форматом sql

Таким чином наповнення відкритого файлу одразу буде завантажено до області «до» та головній для редагування.

3.2 Редагування тексту запитів у текстових областях

У разі, якщо користувач бажає наповними застосунок вручну, то можна скористатися головною областю для редагування тексту або областю «до». По завершенню набору тексту до використання методів рефакторингу треба перенести текст у текстову область «до», або залишити у будь-якій області для розрахування метрики без порівняння.

Головну текстову область позначено на рисунку 3.2.1 та текстову область «до» на рисунку 3.2.2.

Рисунок 3.2.1 – Головна текстова область

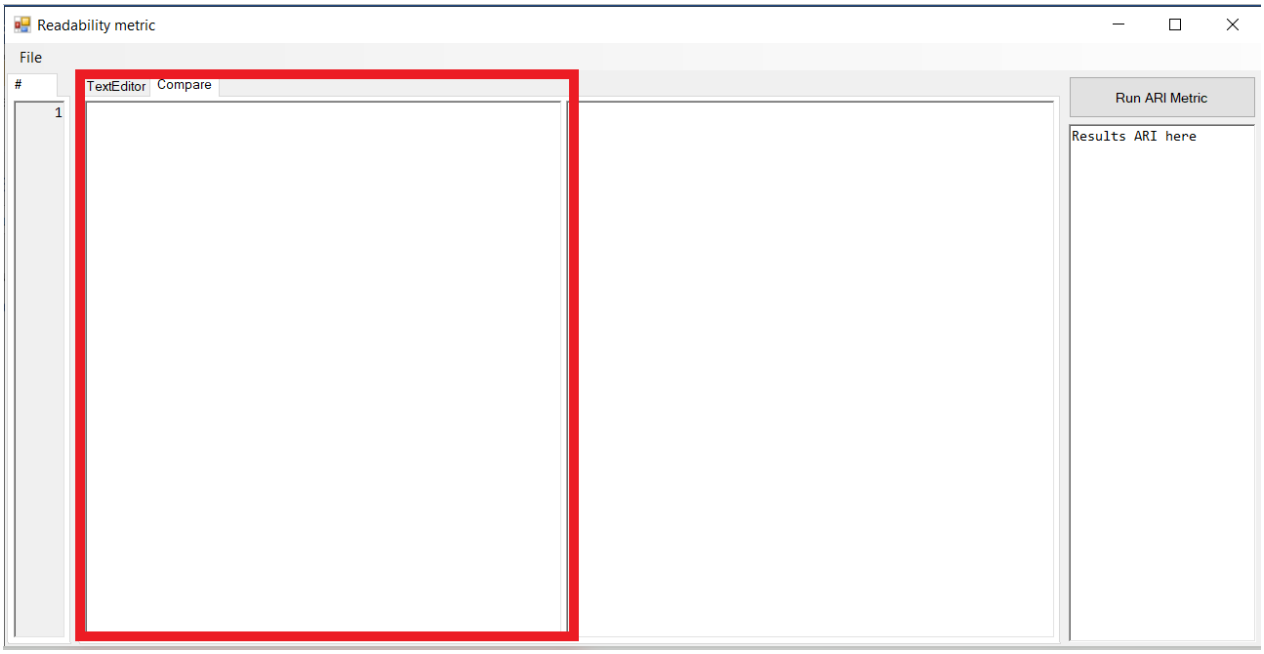


Рисунок 3.2.2 – Текстова область «до»

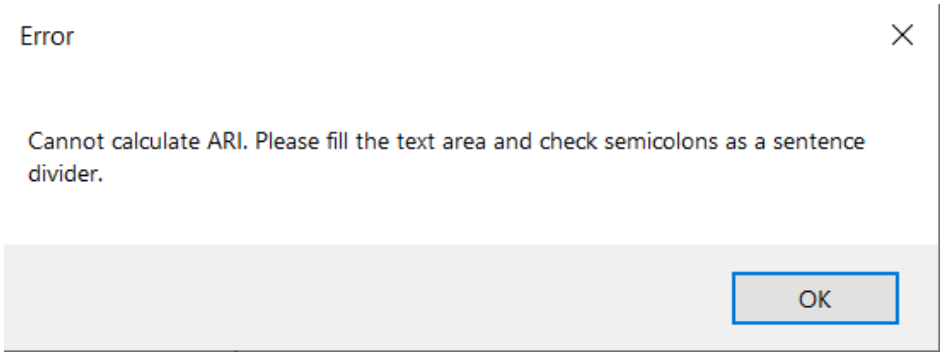
Для порівняння запитів без методів рефакторингу та з – спристайтеся текстовою областю «після» (рисунок 3.2.3), де до тексту запитів застосуйте рефакторинг. Дану процедуру можна робити як з області «після». так й з головної області для редагування тексту.

Рисунок 3.2.3 – Текстова область «після»

3.3 Розрахунок метрики читабельності ARI

Після завершення підготовки треба рохрахувати метрику ARI натиснувши кнопку «Run ARI Metric». Через деякий час результати з’являться у області, що розташована одразу під кнопкою. Приклад результатів наведно на рисунку 3.3.1. У разі, якщо користувач розпочав розрахунок метрики без належного заповнення текстових областей, то перед ним з’явиться помилка зображена на рисунку 3.3.2.

Рисунок 3.3.1 – Виведення розрахунків



3.4 Збереження файлу з текстом запитів, який було редаговано методами рефакторингу

Для збереження отредагованого тексту запитів можна скористатися операціями збереження: «Save» – збереження до поточного файлу та «Save as» – збереження до нового файлу через діалогове вікно як на рисунку 3.4.1. Знайти запропоновані операції можна у верхньому меню, як показано на рисунку 3.4.2.

Якщо викинє помилка пов’язана зі шляхом файлу, наприклад, якщо користувач буде здійснювати збереження у файл, якого не існує – буде виведено повідомлення про помилку, як на рисунку 3.4.3.

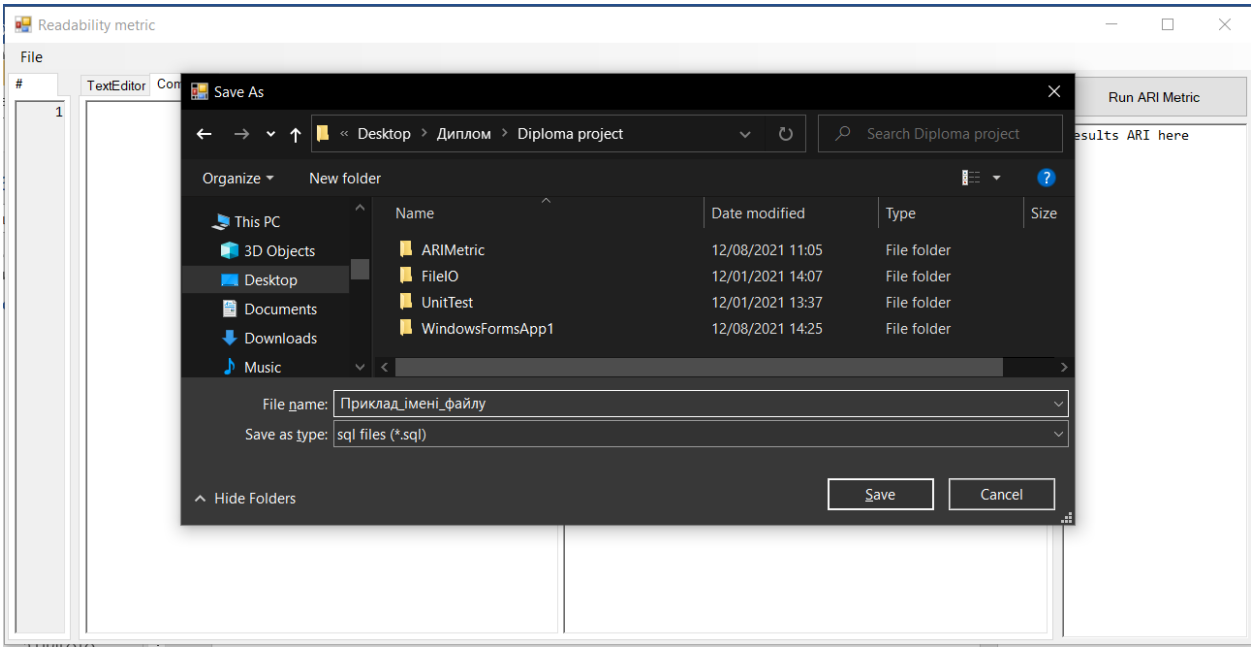


Рисунок 3.4.1 – Діалогове вікно для операції «Save as»

Рисунок 3.4.2 – Операції збереження

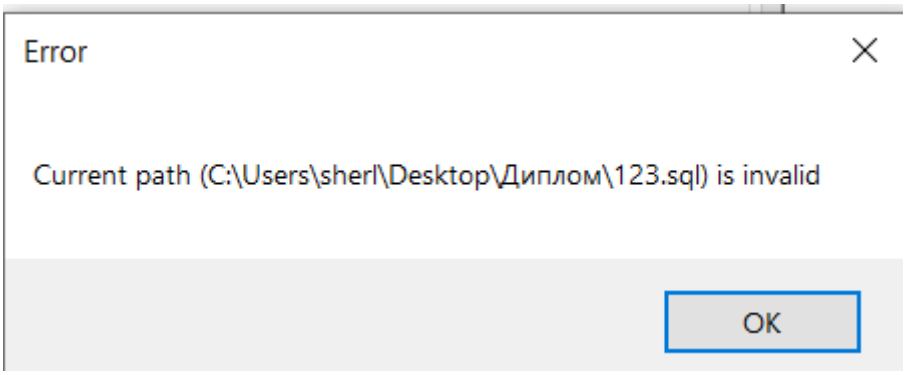


Рисунок 3.4.3 – Повідомлення про помилку шляху

4 АВАРІЙНІ СИТУАЦІЇ

Під час роботи інструменту для аналізу доцільності використання методів рефакторингу можуть виникнути аварійні ситуації наступного характеру:

— нестача вільного місця у оперативній пам’яті;

— нестача вільного місця на жорсткому диску.

У разі виникнення такого роду проблем спробуйте збільшити об’єм вільного місця шляхом очищення жорсткого диску та закриттям інших програм для очищення оперативної пам’яті.

У разі інших проблем зверніться до технічного спеціаліста.

5 РЕКОМЕНДАЦІЇ ЩОДО ЗАСВОЄННЯ

Щоб засвоїти роботу з програмою запустіть програмний застосунок, виконайте дії зазначені у описі операцій. Рекомендовані дії під час користування застосунтом наступні:

— запусть додаток;

— заповніть текстом запитів на мові SQL вручну одну з текстових областей;

— нажиміть кнопку рохрахунку метрики ARI;

— подивіться результат, проаналізуйте самостійно;

— збережіть новий файл з вашим текстои зпитів на персональний комп’ютер;

— перевірте наявність файлу за встановленим шляхом та зміст файлц.

ЗАТВЕРДЖЕНО

1116130.01211-01 12 01-ЛЗ

ІНСТРУМЕНТ РОЗРАХУНКІВ МЕТРИКИ ARI ДЛЯ МОВИ ЗАПИТІВ SQL

Текст програми

1116130.01211-01 12 01

Аркушів 20

2021

АНОТАЦІЯ

Документ 1116130.01211-01 «Інтсрумент розрахунків метрики ARI для мови запитів SQL. Текст програми» входить до складу програмної документації до програми, яка реалізує програмне забезпечення «ARI Metric for SQL».

У даному документі представлений опис модулів програмної системи та текст програми. Програмне забезпечення реалізовувалося на мові C# з використанням бібліотеки WindowsForm.

ЗМІСТ

1 Схема взамодії програмних модулів 4

2 Текст програми 5

2.1 Модуль WindowsFormsApp 5

2.1.1 Текст програми файлу Form.cs 5

2.1.2 Текст програми файлу Program.cs 8

2.1.3 Текст програми файлу Form.Designer.cs 8

2.2 Модуль AriMetric 15

2.2.1 Текст програми файлу ARI.cs 15

2.3 Модуль FileIO 17

2.3.1 Текст програми файлу FileIO.cs 17

2.4 Модуль UnitTest 18

2.4.1 Текст програми файлу UnitTest.cs 18

1 СХЕМА ВЗАМОДІЇ ПРОГРАМНИХ МОДУЛІВ

На рисунку 1.1 представлена схема взаємодії модулів програмної системи.

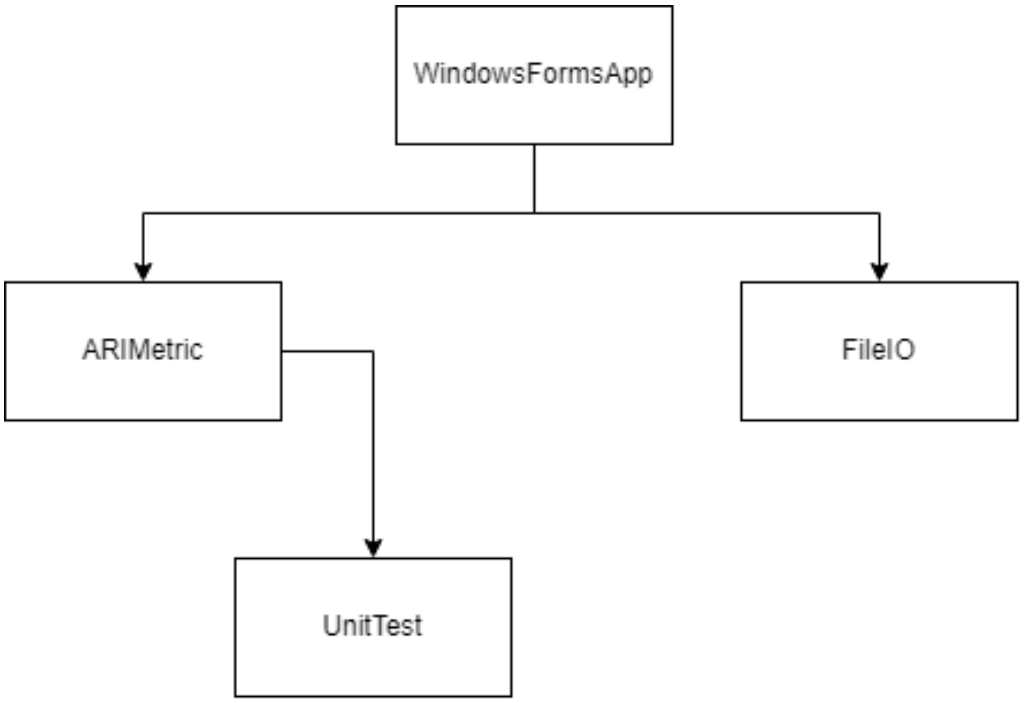


Рисунок 1.1 – Схема модулів програмної системи

WindowsFormsApp – прорамний модуль відповідальний за графічний інтерфейс користувача, де відображаються розрахунки пов’язані з експериментом, початок та завершення програми.

ARIMetric – прорамний модуль відповідальний за розрахунок метрики читабельості ARI.Вхідні дані отпримує з модулю WindowsFormApp.

FileIO – прорамний модуль відповідальний за читання та запис файлів з форматом sql. Вхідні дані отримує з модулю WindowsAppForm.

UnitTest – модуль програми відповідальний за модульне тестування складових програмного проекту.

2 ТЕКСТ ПРОГРАМИ

2.1 Модуль WindowsFormsApp

2.1.1 Текст програми файлу Form.cs

```
using System;
using ARIMetric;
using System.Drawing;
using System.Windows.Forms;

namespace WindowsFormsApp1
{
    /// <summary>
    /// Metric Form class for UI representation and UI actions processing
    /// </summary>
    public partial class MetricForm : Form
    {
        public MetricForm()
        {
            InitializeComponent();
        }
        private void Numeration(RichTextBox richTextBox)
        {
            Point pt = new Point(0, 0);
            int firstIndex = richTextBox.GetCharIndexFromPosition(pt);
            int firstLine = richTextBox.GetLineFromCharIndex(firstIndex);

            pt.X = ClientRectangle.Width;
            pt.Y = ClientRectangle.Height;

            int lastIndex = richTextBox.GetCharIndexFromPosition(pt);
            int lastLine = richTextBox.GetLineFromCharIndex(lastIndex);

            richTextBox_numberstr.SelectionAlignment = HorizontalAlignment.Right;
            richTextBox_numberstr.Clear();

            for (int i = firstLine; i <= lastLine; i++)
            {
                richTextBox_numberstr.Text += $"{i + 1}\n";
            }
        }
        private void MetricForm_Load(object sender, EventArgs e)
        {
            Numeration(richTextBox_textArea);
        }

        private void richTextBox_textArea_TextChanged(object sender, EventArgs e)
        {
            Numeration(richTextBox_textArea);
        }

        private void richTextBox_textArea_VScroll(object sender, EventArgs e)
        {
            Numeration(richTextBox_textArea);
        }

        private void exitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }
```

```
        private void openToolStripMenuItem_Click(object sender, EventArgs e)
        {
            OpenFileDialog dialog = new OpenFileDialog();
            dialog.Filter = "sql files (*.sql)|*.sql|All files (*.*)|*.*";
            if (dialog.ShowDialog() == DialogResult.OK)
            {
                try
                {
                    richTextBox_textArea.Text = FileIO.FileIO.ReadFile(dialog.FileName);
                    if (!FileIO.FileIO.CheckSQLFormat(dialog.FileName))
                    {
                        MessageBox.Show("File is not sql format", "Error");
                    }
                    else
                    {
                        richTextBox_beforeRefactoring.Text = richTextBox_textArea.Text;
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message, "Error");
                }
            }
        }

        private void saveToolStripMenuItem_Click(object sender, EventArgs e)
        {
            try
            {
                FileIO.FileIO.SaveFile(richTextBox_textArea.Text);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Error");
            }
        }

        private void saveAsToolStripMenuItem_Click(object sender, EventArgs e)
        {
            SaveFileDialog dialog = new SaveFileDialog();
            dialog.Filter = "sql files (*.sql)|*.sql|All files (*.*)|*.*";
            if (dialog.ShowDialog() == DialogResult.OK)
            {
                try
                {
                    FileIO.FileIO.SaveAsFile(richTextBox_textArea.Text, dialog.FileName);
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message, "Error");
                }
                MessageBox.Show("Your text was successfully saved", "Info");
            }
        }

        private void button1_Click(object sender, EventArgs e)
        {
            richTextBox_results.Text = string.Empty;

            try
            {
                if (richTextBox_beforeRefactoring.Text != string.Empty &&
richTextBox_afterRefactoring.Text != string.Empty) {
                    ARI metricOldText = new ARI();
```

```
                    metricOldText.ARICalculate(richTextBox_beforeRefactoring.Text);
                    ARI metricNewText = new ARI();
                    metricNewText.ARICalculate(richTextBox_afterRefactoring.Text);

                    double compareScore = Math.Round(metricOldText.ARIScore /
metricNewText.ARIScore, 3);

                    richTextBox_results.Text = $"\tOld text\n"
                        + $"ARI = {metricOldText.ARIScore},\n"
                        + $"characters = {metricOldText.CharactersNumber},\n"
                        + $"words = {metricOldText.WordsNumber},\n"
                        + $"sentences = {metricOldText.SentencesNumber}\n"
                        + $"\n\tNew text\n "
                        + $"ARI = {metricNewText.ARIScore},\n"
                        + $"characters = {metricNewText.CharactersNumber},\n"
                        + $"words = {metricNewText.WordsNumber},\n"
                        + $"sentences = {metricNewText.SentencesNumber}\n"
                        + (compareScore >= 1 ?
                        $"\nText was improved by refactoring on
{Math.Round(metricOldText.ARIScore / metricNewText.ARIScore, 3)} times" :
                        $"\nText was degraded by refactoring on
{Math.Round(metricNewText.ARIScore / metricOldText.ARIScore, 3)} times");
                }
                else if (richTextBox_afterRefactoring.Text != string.Empty)
                {
                    ARI metricNewText = new ARI();
                    metricNewText.ARICalculate(richTextBox_afterRefactoring.Text);

                    richTextBox_results.Text =
                        $"ARI = {metricNewText.ARIScore},\n"
                        + $"characters = {metricNewText.CharactersNumber},\n"
                        + $"words = {metricNewText.WordsNumber},\n"
                        + $"sentences = {metricNewText.SentencesNumber}\n";
                }
                else
                {
                    ARI metricOldText = new ARI();
                    metricOldText.ARICalculate(richTextBox_beforeRefactoring.Text);

                    richTextBox_results.Text =
                        $"ARI = {metricOldText.ARIScore},\n"
                        + $"characters = {metricOldText.CharactersNumber},\n"
                        + $"words = {metricOldText.WordsNumber},\n"
                        + $"sentences = {metricOldText.SentencesNumber}\n";
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Error");
            }
        }

        private void richTextBox_textArea_Leave(object sender, EventArgs e)
        {
            richTextBox_afterRefactoring.Text = richTextBox_textArea.Text;
        }

        private void richTextBox_afterRefactoring_Leave(object sender, EventArgs e)
        {
            richTextBox_textArea.Text = richTextBox_afterRefactoring.Text;
        }

        private void richTextBox_afterRefactoring_TextChanged(object sender, EventArgs e)
        {
```

```
            Numeration(richTextBox_afterRefactoring);
        }

        private void richTextBox_afterRefactoring_VScroll(object sender, EventArgs e)
        {
            Numeration(richTextBox_afterRefactoring);
        }
    }
}
```

2.1.2 Текст програми файлу Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApp1
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new MetricForm());
        }
    }
}
```

2.1.3 Текст програми файлу Form.Designer.cs

```
namespace WindowsFormsApp1
{
   partial class MetricForm
   {
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.IContainer components = null;

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
      protected override void Dispose(bool disposing)
      {
         if (disposing && (components != null))
         {
            components.Dispose();
         }
         base.Dispose(disposing);
      }
```

```
#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
   this.menuStrip1 = new System.Windows.Forms.MenuStrip();
   this.fileToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
   this.openToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
   this.saveToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
   this.saveAsToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
   this.exitToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
   this.tableLayoutPanel1 = new System.Windows.Forms.TableLayoutPanel();
   this.splitContainer1 = new System.Windows.Forms.SplitContainer();
   this.tabControl1 = new System.Windows.Forms.TabControl();
   this.tabPage1 = new System.Windows.Forms.TabPage();
   this.richTextBox_textArea = new System.Windows.Forms.RichTextBox();
   this.tabPage2 = new System.Windows.Forms.TabPage();
   this.splitContainer2 = new System.Windows.Forms.SplitContainer();
   this.richTextBox_beforeRefactoring = new System.Windows.Forms.RichTextBox();
   this.richTextBox_afterRefactoring = new System.Windows.Forms.RichTextBox();
   this.tableLayoutPanel2 = new System.Windows.Forms.TableLayoutPanel();
   this.richTextBox_results = new System.Windows.Forms.RichTextBox();
   this.button_runARI = new System.Windows.Forms.Button();
   this.tabControl2 = new System.Windows.Forms.TabControl();
   this.tabPage3 = new System.Windows.Forms.TabPage();
   this.richTextBox_numberstr = new System.Windows.Forms.RichTextBox();
   this.colorDialog1 = new System.Windows.Forms.ColorDialog();
   this.saveFileDialog1 = new System.Windows.Forms.SaveFileDialog();
   this.menuStrip1.SuspendLayout();
   this.tableLayoutPanel1.SuspendLayout();
   ((System.ComponentModel.ISupportInitialize)(this.splitContainer1)).BeginInit();
   this.splitContainer1.Panel1.SuspendLayout();
   this.splitContainer1.Panel2.SuspendLayout();
   this.splitContainer1.SuspendLayout();
   this.tabControl1.SuspendLayout();
   this.tabPage1.SuspendLayout();
   this.tabPage2.SuspendLayout();
   ((System.ComponentModel.ISupportInitialize)(this.splitContainer2)).BeginInit();
   this.splitContainer2.Panel1.SuspendLayout();
   this.splitContainer2.Panel2.SuspendLayout();
   this.splitContainer2.SuspendLayout();
   this.tableLayoutPanel2.SuspendLayout();
   this.tabControl2.SuspendLayout();
   this.tabPage3.SuspendLayout();
   this.SuspendLayout();
   //
   // menuStrip1
   //
   this.menuStrip1.ImageScalingSize = new System.Drawing.Size(20, 20);
   this.menuStrip1.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
   this.fileToolStripMenuItem});
   this.menuStrip1.Location = new System.Drawing.Point(0, 0);
```

```
        this.menuStrip1.Name = "menuStrip1";
        this.menuStrip1.Size = new System.Drawing.Size(1289, 28);
        this.menuStrip1.TabIndex = 0;
        this.menuStrip1.Text = "menuStrip1";
        //
        // fileToolStripMenuItem
        //
        this.fileToolStripMenuItem.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {
        this.openToolStripMenuItem,
        this.saveToolStripMenuItem,
        this.saveAsToolStripMenuItem,
        this.exitToolStripMenuItem});
        this.fileToolStripMenuItem.Name = "fileToolStripMenuItem";
        this.fileToolStripMenuItem.Size = new System.Drawing.Size(46, 24);
        this.fileToolStripMenuItem.Text = "File";
        //
        // openToolStripMenuItem
        //
        this.openToolStripMenuItem.Name = "openToolStripMenuItem";
        this.openToolStripMenuItem.Size = new System.Drawing.Size(141, 26);
        this.openToolStripMenuItem.Text = "Open";
        this.openToolStripMenuItem.Click += new System.EventHandler(this.openToolStripMenuItem_Click);
        //
        // saveToolStripMenuItem
        //
        this.saveToolStripMenuItem.Name = "saveToolStripMenuItem";
        this.saveToolStripMenuItem.Size = new System.Drawing.Size(141, 26);
        this.saveToolStripMenuItem.Text = "Save";
        this.saveToolStripMenuItem.Click += new System.EventHandler(this.saveToolStripMenuItem_Click);
        //
        // saveAsToolStripMenuItem
        //
        this.saveAsToolStripMenuItem.Name = "saveAsToolStripMenuItem";
        this.saveAsToolStripMenuItem.Size = new System.Drawing.Size(141, 26);
        this.saveAsToolStripMenuItem.Text = "Save as";
        this.saveAsToolStripMenuItem.Click += new
System.EventHandler(this.saveAsToolStripMenuItem_Click);
        //
        // exitToolStripMenuItem
        //
        this.exitToolStripMenuItem.Name = "exitToolStripMenuItem";
        this.exitToolStripMenuItem.Size = new System.Drawing.Size(141, 26);
        this.exitToolStripMenuItem.Text = "Exit";
        this.exitToolStripMenuItem.Click += new System.EventHandler(this.exitToolStripMenuItem_Click);
        //
        // tableLayoutPanel1
        //
        this.tableLayoutPanel1.ColumnCount = 2;
        this.tableLayoutPanel1.ColumnStyles.Add(new
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 5.740884F));
        this.tableLayoutPanel1.ColumnStyles.Add(new
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 94.25912F));
        this.tableLayoutPanel1.Controls.Add(this.splitContainer1, 1, 0);
        this.tableLayoutPanel1.Controls.Add(this.tabControl2, 0, 0);
        this.tableLayoutPanel1.Dock = System.Windows.Forms.DockStyle.Fill;
        this.tableLayoutPanel1.Location = new System.Drawing.Point(0, 28);
```

```
        this.tableLayoutPanel1.Name = "tableLayoutPanel1";
        this.tableLayoutPanel1.RowCount = 1;
        this.tableLayoutPanel1.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 50F));
        this.tableLayoutPanel1.Size = new System.Drawing.Size(1289, 592);
        this.tableLayoutPanel1.TabIndex = 1;
        //
        // splitContainer1
        //
        this.splitContainer1.Dock = System.Windows.Forms.DockStyle.Fill;
        this.splitContainer1.Location = new System.Drawing.Point(76, 3);
        this.splitContainer1.Name = "splitContainer1";
        //
        // splitContainer1.Panel1
        //
        this.splitContainer1.Panel1.Controls.Add(this.tabControl1);
        //
        // splitContainer1.Panel2
        //
        this.splitContainer1.Panel2.Controls.Add(this.tableLayoutPanel2);
        this.splitContainer1.Size = new System.Drawing.Size(1210, 586);
        this.splitContainer1.SplitterDistance = 1008;
        this.splitContainer1.TabIndex = 0;
        //
        // tabControl1
        //
        this.tabControl1.Controls.Add(this.tabPage1);
        this.tabControl1.Controls.Add(this.tabPage2);
        this.tabControl1.Dock = System.Windows.Forms.DockStyle.Fill;
        this.tabControl1.Location = new System.Drawing.Point(0, 0);
        this.tabControl1.Name = "tabControl1";
        this.tabControl1.SelectedIndex = 0;
        this.tabControl1.Size = new System.Drawing.Size(1008, 586);
        this.tabControl1.TabIndex = 1;
        //
        // tabPage1
        //
        this.tabPage1.Controls.Add(this.richTextBox_textArea);
        this.tabPage1.Location = new System.Drawing.Point(4, 25);
        this.tabPage1.Name = "tabPage1";
        this.tabPage1.Padding = new System.Windows.Forms.Padding(3);
        this.tabPage1.Size = new System.Drawing.Size(1000, 557);
        this.tabPage1.TabIndex = 0;
        this.tabPage1.Text = "TextEditor";
        this.tabPage1.UseVisualStyleBackColor = true;
        //
        // richTextBox_textArea
        //
        this.richTextBox_textArea.Dock = System.Windows.Forms.DockStyle.Fill;
        this.richTextBox_textArea.Font = new System.Drawing.Font("Consolas", 9F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
        this.richTextBox_textArea.Location = new System.Drawing.Point(3, 3);
        this.richTextBox_textArea.Name = "richTextBox_textArea";
        this.richTextBox_textArea.Size = new System.Drawing.Size(994, 551);
        this.richTextBox_textArea.TabIndex = 0;
        this.richTextBox_textArea.Text = "";
```

```
        this.richTextBox_textArea.VScroll += new System.EventHandler(this.richTextBox_textArea_VScroll);
        this.richTextBox_textArea.TextChanged += new
System.EventHandler(this.richTextBox_textArea_TextChanged);
        this.richTextBox_textArea.Leave += new System.EventHandler(this.richTextBox_textArea_Leave);
        //
        // tabPage2
        //
        this.tabPage2.Controls.Add(this.splitContainer2);
        this.tabPage2.Location = new System.Drawing.Point(4, 25);
        this.tabPage2.Name = "tabPage2";
        this.tabPage2.Padding = new System.Windows.Forms.Padding(3);
        this.tabPage2.Size = new System.Drawing.Size(1000, 557);
        this.tabPage2.TabIndex = 1;
        this.tabPage2.Text = "Compare";
        this.tabPage2.UseVisualStyleBackColor = true;
        //
        // splitContainer2
        //
        this.splitContainer2.Dock = System.Windows.Forms.DockStyle.Fill;
        this.splitContainer2.Location = new System.Drawing.Point(3, 3);
        this.splitContainer2.Name = "splitContainer2";
        //
        // splitContainer2.Panel1
        //
        this.splitContainer2.Panel1.Controls.Add(this.richTextBox_beforeRefactoring);
        //
        // splitContainer2.Panel2
        //
        this.splitContainer2.Panel2.Controls.Add(this.richTextBox_afterRefactoring);
        this.splitContainer2.Size = new System.Drawing.Size(994, 551);
        this.splitContainer2.SplitterDistance = 489;
        this.splitContainer2.TabIndex = 0;
        //
        // richTextBox_beforeRefactoring
        //
        this.richTextBox_beforeRefactoring.Dock = System.Windows.Forms.DockStyle.Fill;
        this.richTextBox_beforeRefactoring.Location = new System.Drawing.Point(0, 0);
        this.richTextBox_beforeRefactoring.Name = "richTextBox_beforeRefactoring";
        this.richTextBox_beforeRefactoring.Size = new System.Drawing.Size(489, 551);
        this.richTextBox_beforeRefactoring.TabIndex = 0;
        this.richTextBox_beforeRefactoring.Text = "";
        //
        // richTextBox_afterRefactoring
        //
        this.richTextBox_afterRefactoring.Dock = System.Windows.Forms.DockStyle.Fill;
        this.richTextBox_afterRefactoring.Location = new System.Drawing.Point(0, 0);
        this.richTextBox_afterRefactoring.Name = "richTextBox_afterRefactoring";
        this.richTextBox_afterRefactoring.Size = new System.Drawing.Size(501, 551);
        this.richTextBox_afterRefactoring.TabIndex = 1;
        this.richTextBox_afterRefactoring.Text = "";
        this.richTextBox_afterRefactoring.VScroll += new
System.EventHandler(this.richTextBox_afterRefactoring_VScroll);
        this.richTextBox_afterRefactoring.TextChanged += new
System.EventHandler(this.richTextBox_afterRefactoring_TextChanged);
        this.richTextBox_afterRefactoring.Leave += new
System.EventHandler(this.richTextBox_afterRefactoring_Leave);
```

```
        //
        // tableLayoutPanel2
        //
        this.tableLayoutPanel2.ColumnCount = 1;
        this.tableLayoutPanel2.ColumnStyles.Add(new
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 50F));
        this.tableLayoutPanel2.Controls.Add(this.richTextBox_results, 0, 1);
        this.tableLayoutPanel2.Controls.Add(this.button_runARI, 0, 0);
        this.tableLayoutPanel2.Dock = System.Windows.Forms.DockStyle.Fill;
        this.tableLayoutPanel2.Location = new System.Drawing.Point(0, 0);
        this.tableLayoutPanel2.Name = "tableLayoutPanel2";
        this.tableLayoutPanel2.RowCount = 2;
        this.tableLayoutPanel2.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 8.532423F));
        this.tableLayoutPanel2.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 91.46758F));
        this.tableLayoutPanel2.Size = new System.Drawing.Size(198, 586);
        this.tableLayoutPanel2.TabIndex = 0;
        //
        // richTextBox_results
        //
        this.richTextBox_results.Dock = System.Windows.Forms.DockStyle.Fill;
        this.richTextBox_results.Font = new System.Drawing.Font("Consolas", 9F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
        this.richTextBox_results.Location = new System.Drawing.Point(3, 52);
        this.richTextBox_results.Name = "richTextBox_results";
        this.richTextBox_results.Size = new System.Drawing.Size(192, 531);
        this.richTextBox_results.TabIndex = 3;
        this.richTextBox_results.Text = "Results ARI here";
        //
        // button_runARI
        //
        this.button_runARI.Dock = System.Windows.Forms.DockStyle.Fill;
        this.button_runARI.Location = new System.Drawing.Point(3, 3);
        this.button_runARI.Name = "button_runARI";
        this.button_runARI.Size = new System.Drawing.Size(192, 43);
        this.button_runARI.TabIndex = 0;
        this.button_runARI.Text = "Run ARI Metric";
        this.button_runARI.UseVisualStyleBackColor = true;
        this.button_runARI.Click += new System.EventHandler(this.button1_Click);
        //
        // tabControl2
        //
        this.tabControl2.Controls.Add(this.tabPage3);
        this.tabControl2.Dock = System.Windows.Forms.DockStyle.Fill;
        this.tabControl2.Location = new System.Drawing.Point(3, 3);
        this.tabControl2.Name = "tabControl2";
        this.tabControl2.SelectedIndex = 0;
        this.tabControl2.Size = new System.Drawing.Size(67, 586);
        this.tabControl2.TabIndex = 1;
        //
        // tabPage3
        //
        this.tabPage3.Controls.Add(this.richTextBox_numberstr);
        this.tabPage3.Font = new System.Drawing.Font("Consolas", 9F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
```

```
        this.tabPage3.Location = new System.Drawing.Point(4, 25);
        this.tabPage3.Name = "tabPage3";
        this.tabPage3.Padding = new System.Windows.Forms.Padding(3);
        this.tabPage3.Size = new System.Drawing.Size(59, 557);
        this.tabPage3.TabIndex = 0;
        this.tabPage3.Text = "#";
        this.tabPage3.UseVisualStyleBackColor = true;
        //
        // richTextBox_numberstr
        //
        this.richTextBox_numberstr.DetectUrls = false;
        this.richTextBox_numberstr.Dock = System.Windows.Forms.DockStyle.Fill;
        this.richTextBox_numberstr.Font = new System.Drawing.Font("Consolas", 9F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
        this.richTextBox_numberstr.Location = new System.Drawing.Point(3, 3);
        this.richTextBox_numberstr.Name = "richTextBox_numberstr";
        this.richTextBox_numberstr.ReadOnly = true;
        this.richTextBox_numberstr.RightToLeft = System.Windows.Forms.RightToLeft.Yes;
        this.richTextBox_numberstr.ScrollBars = System.Windows.Forms.RichTextBoxScrollBars.None;
        this.richTextBox_numberstr.Size = new System.Drawing.Size(53, 551);
        this.richTextBox_numberstr.TabIndex = 1;
        this.richTextBox_numberstr.Text = "";
        //
        // MetricForm
        //
        this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.ClientSize = new System.Drawing.Size(1289, 620);
        this.Controls.Add(this.tableLayoutPanel1);
        this.Controls.Add(this.menuStrip1);
        this.Name = "MetricForm";
        this.Text = "Readability metric";
        this.Load += new System.EventHandler(this.MetricForm_Load);
        this.menuStrip1.ResumeLayout(false);
        this.menuStrip1.PerformLayout();
        this.tableLayoutPanel1.ResumeLayout(false);
        this.splitContainer1.Panel1.ResumeLayout(false);
        this.splitContainer1.Panel2.ResumeLayout(false);
        ((System.ComponentModel.ISupportInitialize)(this.splitContainer1)).EndInit();
        this.splitContainer1.ResumeLayout(false);
        this.tabControl1.ResumeLayout(false);
        this.tabPage1.ResumeLayout(false);
        this.tabPage2.ResumeLayout(false);
        this.splitContainer2.Panel1.ResumeLayout(false);
        this.splitContainer2.Panel2.ResumeLayout(false);
        ((System.ComponentModel.ISupportInitialize)(this.splitContainer2)).EndInit();
        this.splitContainer2.ResumeLayout(false);
        this.tableLayoutPanel2.ResumeLayout(false);
        this.tabControl2.ResumeLayout(false);
        this.tabPage3.ResumeLayout(false);
        this.ResumeLayout(false);
        this.PerformLayout();

    }

    #endregion
```

```
        private System.Windows.Forms.MenuStrip menuStrip1;
        private System.Windows.Forms.ToolStripMenuItem fileToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem openToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem saveToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem saveAsToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem exitToolStripMenuItem;
        private System.Windows.Forms.TableLayoutPanel tableLayoutPanel1;
        private System.Windows.Forms.SplitContainer splitContainer1;
        private System.Windows.Forms.TabControl tabControl1;
        private System.Windows.Forms.TabPage tabPage1;
        private System.Windows.Forms.TabPage tabPage2;
        private System.Windows.Forms.TabControl tabControl2;
        private System.Windows.Forms.TabPage tabPage3;
        private System.Windows.Forms.RichTextBox richTextBox_textArea;
        private System.Windows.Forms.RichTextBox richTextBox_numberstr;
        private System.Windows.Forms.SplitContainer splitContainer2;
        private System.Windows.Forms.RichTextBox richTextBox_beforeRefactoring;
        private System.Windows.Forms.RichTextBox richTextBox_afterRefactoring;
        private System.Windows.Forms.ColorDialog colorDialog1;
        private System.Windows.Forms.SaveFileDialog saveFileDialog1;
        private System.Windows.Forms.TableLayoutPanel tableLayoutPanel2;
        private System.Windows.Forms.RichTextBox richTextBox_results;
        private System.Windows.Forms.Button button_runARI;
    }
}
```

2.2 Модуль ARIMetric

2.2.1 Текст програми файлу ARI.cs

```
using System;
using System.Linq;
using System.Text.RegularExpressions;

namespace ARIMetric
{
    /// <summary>
    /// ARI - Automated Readability Index - ARI metric calculator
    /// </summary>
    public class ARI
    {
        private double _ARIScore;
        public double WordsNumber { get; private set; }
        public double SentencesNumber { get; private set; }
        public double CharactersNumber { get; private set; }
        public double ARIScore
        {
            get
            {
                return Math.Round(_ARIScore, 3);
            }
            private set
            {
                _ARIScore = value;
            }
```

```
        }

        public ARI()
        {
            WordsNumber = 0;
            CharactersNumber = 0;
            SentencesNumber = 0;
            ARIScore = 0;
        }
        /// <summary>
        /// Calculate number of characters (digits and letters) in the queries text
        /// </summary>
        /// <param name="queries">string which consist sql queries</param>
        private void CalculateCharactersNumber(string queries)
        {
            CharactersNumber = queries.Count(ch => char.IsDigit(ch) || char.IsLetter(ch));
        }

        /// <summary>
        /// Calculate number of words (numbers and words) in the queries text
        /// </summary>
        /// <param name="queries">string which consist sql queries</param>
        private void CalculateWordsNumber(string queries)
        {
            //exclude situation with non-digit or non-letter in a row
            bool prevSymbolIsLetterOrDigit = false;

            for (int i = 0; i < queries.Length; i++)
            {
                if (char.IsLetter(queries[i]) || char.IsDigit(queries[i]))
                {
                    prevSymbolIsLetterOrDigit = true;
                }
                else
                {
                    if (prevSymbolIsLetterOrDigit)
                    {
                        WordsNumber++;
                        prevSymbolIsLetterOrDigit = false;
                    }
                    else
                    {
                        prevSymbolIsLetterOrDigit = false;
                    }
                }
            }
        }

        /// <summary>
        /// Calculate number of sentences (queries) in the queries text
        /// </summary>
        /// <param name="queries">string which consist sql queries</param>
        private void CalculateSentencesNumber(string queries)
        {
            char[] delimiters = new char[] { ';' };
            queries = queries.Trim();
            SentencesNumber = queries.Split(delimiters,
StringSplitOptions.RemoveEmptyEntries).Length;

            // calculate CTE quieries as a sentences too
            if (queries.ToUpper().Contains("WITH"))
            {
                Regex regex = new Regex(@"AS\s*\(\s*SELECT");
                MatchCollection matches = regex.Matches(queries.ToUpper());
```

```
                SentencesNumber += matches.Count;
            }
        }

        /// <summary>
        /// Calculate Automated Readability Index
        /// </summary>
        /// <param name="queries">string which consist sql queries</param>
        /// <returns>Double variable of the ARI rounded up to 3 characters</returns>
        public double ARICalculate(string queries)
        {
            CalculateCharactersNumber(queries);
            CalculateWordsNumber(queries);
            CalculateSentencesNumber(queries);

            try
            {
                ARIScore = ((4.71 * (CharactersNumber / WordsNumber)) + (0.5 * (WordsNumber /
SentencesNumber))) - 21.43;
                if (double.IsNaN(ARIScore))
                {
                    throw new DivideByZeroException();
                }
            }
            catch (DivideByZeroException)
            {
                throw new ArgumentException("Cannot calculate ARI. Please fill the text area and
check semicolons as a sentence divider.");
            }

            return ARIScore;
        }
    }
}
```

2.3 Модуль FileIO

2.3.1 Текст програми файлу FileIO.cs

```
using System;
using System.IO;
using System.Text;

namespace FileIO
{
    /// <summary>
    /// FileIO class to read-save text files to device
    /// </summary>
    public static class FileIO
    {
        public static string Path { get; set; }

        public static bool CheckSQLFormat(string path) =>
System.IO.Path.GetExtension(path).ToLower() == ".sql";
        /// <summary>
        /// Read text file from device
        /// </summary>
        /// <param name="path">full path to the text file</param>
        /// <returns></returns>
        public static string ReadFile(string path)
        {
            Path = path;
```

```
            try
            {
                return File.ReadAllText(path);
            }
            catch
            {
                throw new IOException($"Cannot read file from {Path}");
            }
        }

        /// <summary>
        /// Save text file to the device
        /// </summary>
        /// <param name="text">string that need to be saved</param>
        /// <param name="path">full path to the new or existing file</param>
        public static void SaveAsFile(string text, string path)
        {
            Path = path;
            File.WriteAllText(path, text, Encoding.UTF8);
        }

        /// <summary>
        /// Save text to the working file
        /// </summary>
        /// <param name="text">string that need to be saved</param>
        public static void SaveFile(string text)
        {
            if (File.Exists(Path))
            {
                File.WriteAllText(Path, text, Encoding.UTF8);
            }
            else
            {
                throw new IOException($"Current path ({Path}) is invalid");
            }
        }
    }
}
```

2.4 Модуль UnitTest

2.4.1 Текст програми файлу UnitTest.cs

```
using ARIMetric;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;

namespace UnitTest
{
    [TestClass]
    public class UnitTest
    {
        [TestMethod]
        public void CheckCalculaction_ValidText_ValidResult()
        {
            //Arrange
            string text = "SELECT * FROM Table;";
            double charactersNumber = 15;
            double wordsNumber = 3;
            double sentencesNumber = 1;
```

```
            double expected = Math.Round(4.71 * (charactersNumber / wordsNumber) + 0.5 *
(wordsNumber / sentencesNumber) - 21.43, 3);
            //Act
            var actualARI = new ARI();
            actualARI.ARICalculate(text);
            //Assert
            Assert.AreEqual(expected, actualARI.ARIScore);
            Assert.AreEqual(charactersNumber, actualARI.CharactersNumber);
            Assert.AreEqual(wordsNumber, actualARI.WordsNumber);
            Assert.AreEqual(sentencesNumber, actualARI.SentencesNumber);
        }

        [TestMethod]
        public void CheckCalculaction_ValidTextManyQueries_ValidResult()
        {
            //Arrange
            string text = "SELECT * FROM Table; SELECT * FROM Table2;";
            double charactersNumber = 31;
            double wordsNumber = 6;
            double sentencesNumber = 2;
            double expected = Math.Round(4.71 * (charactersNumber / wordsNumber) + 0.5 *
(wordsNumber / sentencesNumber) - 21.43, 3);
            //Act
            var actualARI = new ARI();
            actualARI.ARICalculate(text);
            //Assert
            Assert.AreEqual(expected, actualARI.ARIScore);
            Assert.AreEqual(charactersNumber, actualARI.CharactersNumber);
            Assert.AreEqual(wordsNumber, actualARI.WordsNumber);
            Assert.AreEqual(sentencesNumber, actualARI.SentencesNumber);
        }

        [TestMethod]
        public void CheckCalculaction_ValidTextCTE_ValidResult()
        {
            //Arrange
            string text = "WITH [testCTE] AS( SELECT * FROM Table) SELECT * FROM [testCTE];";
            double charactersNumber = 45;
            double wordsNumber = 9;
            double sentencesNumber = 2;
            double expected = Math.Round(4.71 * (charactersNumber / wordsNumber) + 0.5 *
(wordsNumber / sentencesNumber) - 21.43, 3);
            //Act
            var actualARI = new ARI();
            actualARI.ARICalculate(text);
            //Assert
            Assert.AreEqual(expected, actualARI.ARIScore);
            Assert.AreEqual(charactersNumber, actualARI.CharactersNumber);
            Assert.AreEqual(wordsNumber, actualARI.WordsNumber);
            Assert.AreEqual(sentencesNumber, actualARI.SentencesNumber);
        }

            [TestMethod]
        public void CheckCalculaction_TextWithoutSemicolons_ThrowArgumentException()
        {
            //Arrange
            string text = "";
            //Act
            //Assert
            Assert.ThrowsException<ArgumentException>(() => { new ARI().ARICalculate(text);});
        }
    }
}
```

Міністерство освіти і науки України

Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна

ТЕЗИ

Всеукраїнської науково-технічної конференції молодих учених, магістрантів та студентів «Науково-технічний прогрес на транспорті»

(29 березня 2021 року)

Дніпро
2021

A. A. Ryzhkova
Research supervisor: A. P. Ivanov, Cand. of Engineering Sciences, Associate Professor
Language supervisor: A. O. Muntian, Cand. of Philological Sciences, Associate Professor
Dnipro National University of Railway Transport named after Academician V. Lazarian

REFACTORING SQL QUERIES

Often, code that was hastily written or hasn't been refactored for a long time is difficult for the reader to understand. Common knowledge is that on the project programs are read much more often than they are written, so do not forget about such an important component of the software development process. For a fast and quality development process, both new and old team members need to understand what is happening in the program they are developing. This knowledge will later help you write code faster, look for errors in the program, and possibly even improve system performance.

So what is refactoring? Refactoring consists of improving the internal structure of an existing program's source code, while preserving its external behavior. Usually, refactoring is done after adding some functionality to the program. Programmers do not neglect this technique, but use it either when urgently needed, or refactor regularly.

Since this technique is important for development, it makes sense to apply it not only to functional and object-oriented programming languages, but also to other programming paradigms. It is proposed to consider the query language SQL. This language is standard formost DBMS in use that use a relational approach. Relational databases occupy a leading position in the global market.

Refactoring of SQL queries can be carried out with two main goals: improving the readability of a query by changing its structure, and improving performance. With such a change, the database schema should not be changed, because these techniques will relate to refactoring the database architecture.

This topic is relevant now, it is being considered by experts. Leading SQL formatters are starting to add techniques that will help improve not only the database architecture, but also the previously written queries. As a result of the search, we were able to find two software products that provide refactoring methods, but a small number of the proposed methods can be attributed specifically to refactoring SQL queries.

To implement the development of methods for refactoring SQL queries, you can use the methods that are intended for object-oriented programming languages as a basis. You can also try to replace some language constructs in favor of those that will improve the performance of the query.

For the best application of the found methods, it is proposed to create software that will automate the process of refactoring queries. Such a solution will allow you to avoid mistakes when applying methods yourself, reduce the time for changing a query and make it more convenient. Since queries are usually used in the text of programs that are written in an object- oriented language and queries themselves do not always need refactoring, it makes little sense to develop an extension for the development environment, but it would be more comfortable, because there is no need

to use others. means. Nevertheless, it is proposed to develop a separate program that will contain the necessary tools for refactoring SQL queries.

As a result, it can be noted that this topic is relevant and still insufficiently studied. Some software products already include refactoring techniques in their toolbox. It is necessary to develop in this direction to improve the work of software products with databases.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИАТ

«УКРАЇНСЬКА ЗАЛІЗНИЦЯ»

ДНІПРОВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ЗАЛІЗНИЧНОГОТРАНСПОРТУ ІМЕНІ АКАДЕМІКА В. ЛАЗАРЯНА

УКРАЇНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЗАЛІЗНИЧНОГО ТРАНСПОРТУ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФРАСТРУКТУРИ ТА ТЕХНОЛОГІЙ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ АВТОМОБІЛЬНО-ДОРОЖНІЙ УНІВЕРСИТЕТ

ІНФОРМАЦІЙНО-ТЕЛЕКОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ ТА КОМП'ЮТЕРНЕ МОДЕЛЮВАННЯ

ЗБІРНИК ТЕЗ ДОПОВІДЕЙ

81 Всеукраїнської науково-технічної конференції молодих учених, магістрантів та студентів

«НАУКА І СТАЛИЙ РОЗВИТОК ТРАНСПОРТУ»

28 жовтня 2021 року

CONFERENCE PROCEEDINGS

81th all Ukrainian Scientific and Technical Conferenceof young scientists, masters and students

"SCIENCE AND SUSTAINABLE DEVELOPMENTOF TRANSPORT"

October 28, 2021

ДНІПРО2021

РЕФАКТОРИНГ SQL ЗАПИТІВ

Автор: Рижкова А. А., студентка групи
ПЗ2021Науковий керівник: к.т.н., доцент
Іванов О.П.
Дніпровський національний університет залізничного транспортуімені
академіка В. Лазаряна

Зазвичай, текст програми, який був написаний на швидку руку або ж не піддавався рефакторингу дуже давно є для читача чимось складним для розуміння. Як відомо, на проекті тексти програм читають набагато частіше, ніж пишуть, тому не варто забувати про таку важливу складову процесу розробки програмного забезпечення. Для швидкого і якісного процесу розробки як нові, так і старі члени команди повинні розуміти, що відбувається в програмі, яку вони розробляють. Такі знання надалі допоможуть швидше писати код, шукати помилки в програмі і можливо навіть поліпшити продуктивність системи.

Так що ж таке рефакторинг? Рефакторинг полягає в поліпшенні внутрішньої структури вихідного коду існуючої програми при збереженні її початкової поведінки. Зазвичай рефакторинг проводять після додавання будь-якого функціоналу в програму. Програмісти не зневажають цю техніку, а користуються цим або за гострої необхідності, або проводять рефакторинг регулярно.

Так як дана техніка важлива для розробки, то має сенс застосувати її не тільки для функціональних і об'єктно-орієнтованих мов програмування, але і для інших парадигм програмування. Пропонується розглянути мову запитів SQL. Ця мова стандартна для більшості популярних СКБД, які застосовують реляційних підхід. Реляційних бази даних займають лідуючі місця на світовому ринку.

Рефакторинг SQL запитів можна проводити з двома головними цілями: поліпшення читабельності запиту за рахунок зміни його структури та поліпшення продуктивності.При такій зміні схема бази даних не повинна бути змінена, тому що дана техніка будє відноситься до рефакторингу архітектури бази даних.

Дана тема є актуальною зараз, вона розглядається фахівцями. Ведучі програми щодо форматування SQL запитів починають додавати методи, які допоможуть поліпшити не тільки архітектуру бази даних, а й написані запити. В результаті пошуку вдалося знайти деякі програмні продукти, які пропонують методи рефакторингу, однак мала кількість із запропонованих методів можна віднести саме до рефакторингу SQL запитів, а не схеми бази даних.

Щоб здійснити розробку методів рефакторинга SQL запитів, можна скористатися методами, які призначені для об'єктно-орієнтованих мов програмування в якості основи. Так само можна спробувати замінити деякі мовні конструкції на користь тих, які підвищать продуктивність виконання запиту.

Для кращого застосування знайдених методів пропонується створити програмне забезпечення, яке дозволить автоматизувати процес рефакторинга запитів. Таке рішення дозволить уникнути помилок при самостійному застосуванні методів, зменшить час зміни запиту і зробить це зручніше. Так як запити зазвичай використовуються в тексті програм, які написані на об'єктно-орієнтованої мови і самі по собі запити не завжди потребують рефакторінгу, то розробляти розширення для

середовища розробки не має великого сенсу,однак це було б комфортніше, тому що немає необхідності користуватися іншими засобами. Все ж пропонується розробити окрему програму, яка буде містити необхідні інструменти рефакторинга SQL запитів.

В результаті, можна відзначити, що дана тема актуальна і все ще недостатньо вивчена. Деякі програмні продукти вже вводять методи рефакторинга в свій набір інструментів. Необхідно розвиватися в даному напрямку, для поліпшення роботи програмних продуктівз базами даних.

Підготовлено до друку

РЕФАКТОРИНГ SQL ЗАПИТІВ

Мета. Створити та описати методи рефакторингу SQL запитів, які не будуть впливати на схему бази даних. **Методика.** Розробити методів засновуючись на вже існуючих підходах рефакторингу для об'єктно орієнтованих мов програмування. Розробити методи перетворення запитів SQL для збільшення їх читабельності. **Наукова новизна.** Наразі ще немає робіт присвячених редагуванню запитів, які б не впливали на зміну схеми бази даних та не були б прив'язані до спеціалізованої СКБД. **Практична значимість.** Застосування методів рефакторингу сприяє покращенню читабельності коду. Це зазвичай робиться після написання нового функціоналу системи. **Результати.** Розроблені методи, які поращують читаємість та розуміння тексту запиту. Також деякі з запропонованих методів можуть збільшити ефективність виконання запитів.

Ключові слова: рефакторинг, SQL, СКБД, запити, читабельність, ефективність.

Вступ

Рефакторинг - це процес зміни програмного проекту, в ході якого зовнішня поведінка тексту програми залишається незмінною при удосконаленні її внутрішньої структури. Це систематизований спосіб очищення коду, що мінімізує можливість появи нових помилок. По суті, рефакторинг коду є поліпшення проекту вже після того, як цей код написаний[1]. Рефакторинг є важливою складовою розробки програмного продукту, тому що наразі текст програм здебільшого читають більше, аніж пишуть. Набагато легше розбиратися у тексті програми, якщо був пройдений етап рефакторингу. Наразі велика кількість розробників користується методами рефакторингу: регулярно або по гострій необхідності.

Метою роботи є розробка методів рефакторингу SQL запитів. Системи керування базами даних(СКБД) використовують мову SQL для маніпулювання даними. Отже, має сенс розробити методи рефакторингу для підтримки покращення читабельності SQL запитів.

Предметом дослідження є питання: чи можна проводити рефакторинг SQL запитів, не втручаючись при цьому в структуру бази даних і якщо так, то якими методами. Буде розглянуто SQL загального стандарту без прив'язки до певних СКБД, що допоможе розширити область застосувань даної роботи, але й обмежить у створенні методів, тому що здебільшого додаткові об'єкти, наприклад такі як збережені процедури, є частиною структури бази даних. Методи, що пов'язані з форматуванням тексту запиту розглядатися не будуть, так як це не є рефакторингом, текст запиту не змінюється, а лише корегується стилістика відображення.

Мета

Мета дослідження полягає в тому, щоб розробити методи рефакторингу запитів та створити програмне забезпечення для їх автоматизованого здійснення, що не вплине на існуючу схему бази даних. Мета самих методів рефакторинга — полегшити розумінням коду, зробити його більш читабельним.

Методика

Для розробки нових методів використана книга «Рефакторинг. Улучшение проекта существующего кода»[1], з метою адаптувати вже існуючі методи рефакторингу для об'єктно орієнтовних мов програмування на методи, які стануть до нагоди у мові SQL. Мартін Фаулер пропонує понад 20 прикладів поганої практики програмування для ООП мов і понад 70 методів рефакторинга. Робота побудована наступним чином: розглядається метод для функціональної мови програмування, передбачається метод для мови запитів, окреслюється його мотивація, техніка виконання та формуються приклади "до" та "після" застосування методу.

Для демонстрації підходу адаптації можна розглянути метод «Перетворення умови IN на вираз», який буде детальніше описано далі. Даний метод був адаптований до мови SQL з методу рефакторингу «Консолідація умовного виразу» для ООП мов. Останній метод пропонує замінити ряд умовних виразів одним складним або ж виносити його в окремий метод. Для нового методу

рефаторингу SQL запиту пропонується об’єднати складну умову, яка перевіряє вміст ряду значень в одному з рядків обраної таблиці в вираз мови запитів IN.

Також метою роботи є дослідження мови запитів та пошук мовних конструкції, які зможуть замінити одна одну для спрощення читання та розуміння текстів запитів. Прикладом подібного методу, передбачає використання більш вдалої мовної конструкції, є «Уточнення імені об’єкту». Цей метод пропонує доповнити імена об’єктів їх джерелом, наприклад, для імені атрибуту уточнити ім’я таблиці з якої він взятий.

Для підтвердження покращення читабельності тексту запитів буде використана метрика оцінки читаємості текстів програм. Метрика Automated Readability Index»[3] створена для реалізації на обчислювальних машинах для апроксимації складності читання до номеру класу учня в американській системі освіти. Сама метрика обчислюється за наступною формулою:

$$ARI = 4.71 * \frac{\text{Кількість символів}}{\text{Кількість слів}} + 0.5 * \frac{\text{Кількість слів}}{\text{Кількість речень}} - 21.43$$

Кількість символів розраховується тільки для літер та цифр, інші символи пропускаються. Для адаптації метрики кількість речень буде рівною кількості запитів, які розділені між собою точкою з комою.

Наукова новизна та практична значимість

Для впровадження методів рефакторингу був проведений аналіз існуючих програмних продуктів, в результаті чого можна сказати, що більшість заявлених методів рефакторингу покликані реконструювати саму схему бази даних. Так само в розгляді аналогів була знайдена книга «Refactoring Legacy T-SQL for improved performance»[2], яка розповідає про засоби мови SQL та як вони впливають на продуктивність, однак акцент робиться так само на зміну схеми бази даних, тому що розглядається конкретна система керування баз даних.

Структури запропонованих методів рефакторингу

Створені методи, для покращення тексту запитів, будуть мати наступну структуру:

1. Назва методу.
2. Адаптація з методу – назва методу рефакторингу для ООП мови, який був адаптований під розглянутий метод (необов’язково).
3. Опис методу, що розглядається – уточнюється, що конкретно буде змінюватися та за яких можливих умов.
4. Мотивація до застосування – описується чому спеціаліст повинен застосувати даний метод для свого запиту.
5. Приклади роботи рефакторингу до застосування методу та після – уточнюються мовні конструкції, які будуть замінені на більш вдалі та наводиться приклад виправлення.
6. Результати розрахунку метрики Automated Readability Index.

Методи рефакторингу розділені на наступні групи:

1. Складання запитів:

a. перейменування псевдонімів та змінних;
b. підстановка розширенням;
c. перелічення параметрів для вставки;
d. уникнення сортування даних.

2. Уточнення даних:

a. уточнення імені об’єкту;
b. додавання або видалення квадратних дужок для ідентифікаторів;
c. додавання точки з комою;
d. додавання або видалення псевдоніму (as);
e. надання операторам повних імен.

3. Спрощення складних запитів:

a. виділення частин запиту у Common Table Expression (CTE);

b. заміна алгоритму об'єднання таблиць.

3. Спрощення умовних виразів:

a. використання WHERE до/замість HAVING;
b. заміна LIKE на оператор рівності;
c. перетворення умови на IN вираз;
d. використання EXISTS або JOIN замість IN.

3. Робота з коментарями:

a. додавання або видалення коментарів.

Результати створених методів рефакторингу

Складання запитів

a. **Перейменування псевдонімів та змінних**

Адаптовано з методу: «Перейменування методу»
*Опис***:** Метод дозволяє перейменовувати змінні або псевдоніми без порушення взаємозв'язків.
Мотивація: Щоб чітко розуміти з якою сутністю або даними розробнику доведеться працювати або отримати на виході, об'єкт повинен бути названий відповідно до того що він означає у даному контексті. Для покращення розуміння перейменуйте об'єкт на більш зрозумілу назву.
Приклад «до застосування методу»: не інформативні псевдоніми.

```
SELECT Table.name, AVG(Table.Count)
FROM(
      SELECT project.name, project_employee.employee_position, COUNT(task.id)
AS Count
      FROM project_employee
      JOIN project ON project.id = project_employee.project_id
      JOIN task ON task.project_employee_id = project_employee.id
      GROUP BY project_employee.employee_position, project.name) AS Table
GROUP BY Table.name;
```

Приклад «після застосування методу»: псевдоніми, які описують відповідні дані, відображають сутність об'єкту.

```
SELECT tasks_number_per_employee.name, AVG(tasks_number_per_employee.
tasks_number)
FROM(
      SELECT project.name, project_employee.employee_position, COUNT(task.id)
as tasks_number
      FROM project_employee
      JOIN project ON project.id = project_employee.project_id
      JOIN task ON task.project_employee_id = project_employee.id
      GROUP BY project_employee.employee_position, project.name) AS
tasks_number_per_employee
GROUP BY tasks_number_per_employee.name;
```

Результати метрики:

Запит	Кількість символів	Кількість слів	ARI
«До»	283	55	30.305
«Після»	363	69	37.849

b. **Підстановка розширенням**

Адаптовано з методу: «Додавання параметру», «Видалення параметру»

Опис: Не використовувати знак * (підстановку розширенням) в операторі SELECT.
Мотивація: Важливо уникати повертати з запиту непотрібні стовбці, це погіршує розуміння запиту, його специфікацію. Не завжди очевидно, що поверне подібний запит. Повернення непотрібних рядків і стовпців таблиці витрачає зайвий час процесора, пам'ять, диск і пропускну здатність мережі.
Приклад «до застосування методу»: не замінена підстановка розширенням.

```
SELECT *
FROM project_employee
JOIN task ON task.project_employee_id = project_employee.id
JOIN project ON project_employee.project_id = project.id
JOIN employee ON employee_id = employee.id
```

Приклад «після застосування методу»: псевдоніми, які описують відповідні дані, відображають сутність об’єкту.

```
SELECT employee.lastname, task.title, project.name
FROM project_employee
JOIN task ON task.project_employee_id = project_employee.id
JOIN project ON project_employee.project_id = project.id
JOIN employee ON employee_id = employee.id
```

Результати метрики:

Запит	Кількість символів	Кількість слів	ARI
«До»	153	30	17.591
«Після»	189	36	21.298

c. Перелічення параметрів для вставки

Адаптовано з методу: «Додавання параметру», «Видалення параметру»
Опис: При використанні оператору INSERT необхідно перелічувати атрибути для вставки.
Мотивація: Важливо перелічувати атрибути у запитах на вставку даних. Це не тільки зробить запит вставки нових даних більш зрозумілим але й при зміні структури таблиці буде легше виправляти дані, якщо компілятор не зможе виконати запит.
Приклад «до застосування методу»: не уточнені параметри для вставки.

```
INSERT task
VALUES
(1, 'Create project architecture','ASP.NET project + Angular', 4, 6),
(2, 'Create calendar matrix','ASP.NET project + Angular', 1, 1),
(3, 'Test plan','Write test plan', 4, 4);
```

Приклад «після застосування методу»: зрозумілий порядок для вставки даних та уточнені стовбці.

```
INSERT task(id, title, description, task status id, project employee id)
VALUES
(1, 'Create project architecture','ASP.NET project + Angular', 4, 6),
(2, 'Create calendar matrix','ASP.NET project + Angular', 1, 1),
(3, 'Test plan','Write test plan', 4, 4);
```

Результати метрики:

Запит	Кількість символів	Кількість слів	ARI
«До»	131	31	13.974
«Після»	178	40	19.53

d. Уникнення сортування даних

Опис: не потрібно використовувати сортування даних ORDER BY, якщо це не несе користі у даному контексті.

Мотивація: Зайві операції роблять текст запиту більш заплутаним, тому без потреби не рекомендується вживати непотрібні елементи. Операція сортування погіршує швидкодійність виконання запиту.

Приклад «до застосування методу»: сортування даних в даному випадку не потрібне, можна замінити іншим оператором.

```
SELECT TOP(1) task_status.id
FROM task_status
JOIN task ON task.task_status_id = task_status.id
JOIN project_employee ON project_employee.id = task.project_employee_id
WHERE project.id = project_employee.project_id
ORDER BY task.id
```

Приклад «після застосування методу»: сортування даних замінено іншим оператором.

```
SELECT MIN(task_status.id)
FROM task_status
JOIN task ON task.task_status_id = task_status.id
JOIN project_employee ON project_employee.id = task.project_employee_id
WHERE project.id = project_employee.project_id
```

Результати метрики:

Запит	Кількість символів	Кількість слів	ARI
«До»	184	41	20.208
«Після»	170	36	18.812

Уточнення даних

a. **Уточнення імені об’єкту**

Опис: Функція уточнення імені об'єкта дозволяє спеціалістам змінити запити SQL, для уточнення всіх імен об'єктів. Уточнення імені об'єкту можна виконати додаванням:

- імені власника об'єкту (ім’я бази даних, схеми даних) для кожного об'єкту, зазначеного в запиті SQL в форматі owner.object;
- імені об'єкту (таблиці, імені уявлення і т. п.) до імен стовпців в форматі table.column;
- ім'я псевдоніма для підзапиту до імен стовпців в форматі alias.column.

Обов’язково використовувати уточнення для об’єктів, які мають однакову назву в декількох об’єктах запиту.

Мотивація: Розробник точно буде знати до якого власника/об’єкта/псевдоніма відноситься ідентифікатор. Це підвищує читабельність та уникає протиріч, якщо деякі об’єкти названі однаково, або майже однаково у різних джерелах. Також SQL сервер не буде перевіряти, чи є поточний користувач власником об'єкту, що прискорить пошук помилок.

Приклад «до застосування методу»: деякі об’єкти не уточнені власником використаного об’єкту.

```
SELECT MIN(task_status.id)
FROM task_status
JOIN task ON task_status_id = task_status.id
JOIN project_employee ON project_employee.id = project_employee_id
WHERE project.id = project_id
```

Приклад «після застосування методу»: додана схема даних та таблиця, стовбець якої використовується.

```
SELECT MIN(dbo.task_status.id)
FROM dbo.task_status
JOIN dbo.task ON dbo.task_status_id = dbo.task_status.id
JOIN dbo.project_employee ON dbo.project_employee.id =
dbo.task.project_employee_id
WHERE dbo.project.id = dbo.project_employee.project_id
```

Результати метрики:

Запит	Кількість символів	Кількість слів	ARI
«До»	147	32	16.207
«Після»	196	44	21.585

b. **Додавання або видалення квадратних дужок для ідентифікаторів**

*Опис***:** Ім’я ідентифікатора можна включати будь які символи, якщо він записаний у квадратних дужках. У деяких СКДБ є внутрішні правила назви ідентифікаторів для особливих об’єктів, тому щоб використати будь які символи для ідентифікаторів необхідні квадратні дужки. Це працю також і для символів, які не включені в кодування UNICODE.

Мотивація: Щоб дозволити імпорт баз даних з систем із іншими правилами, Microsoft SQL Server дозволяє видіфляти ідентифікатори квадратними дужками. Додавання квадратних дужок до ідентифікаторів працює в рамках покращення зовнішнього вигляду запиту.

Приклад «до застосування методу»: помилка використання, символ «@» - у Microsoft SQL Server символізує змінну.

```
SELECT @employee_position, COUNT(employee_id)
FROM project_employee
GROUP BY @employee_position;
```

Приклад «після застосування методу»: ідентифікатор наразі символізує ім’я стовбця.

```
SELECT [@employee_position], COUNT(employee_id)
FROM project_employee
GROUP BY [@employee_position];
```

Результати метрики:

Запит	Кількість символів	Кількість слів	ARI
«До»	79	13	13.692
«Після»	79	13	13.692

c. **Додавання крапки з комою**

*Опис***:** Вставка крапки з комою в кінці кожного оператора, якщо вони були опущені.

Мотивація: Вони не є обов'язковими в стандарті SQL, але полегшують читання пакета. Також це допомагає візуально розділити кінець на початок наступного виразу.

Приклад «до застосування методу»: візуально не зрозуміло, де кінець попереднього запиту:

```
SELECT @employee_position, COUNT(employee_id)
FROM project_employee
GROUP BY @employee_position
```

```
SELECT MIN(task_status.id)

FROM task_status
JOIN task ON task_status_id = task_status.id
JOIN project_employee ON project_employee.id = project_employee_id

WHERE project.id = project_id
```

Приклад «після застосування методу»: точка з комою візуально розділяє запити на окремі:

```
SELECT @employee_position, COUNT(employee_id)
FROM project_employee
GROUP BY @employee_position;

SELECT MIN(task_status.id)

FROM task_status
JOIN task ON task_status_id = task_status.id
JOIN project_employee ON project_employee.id = project_employee_id

WHERE project.id = project_id;
```

Результати метрики:

В даному прикладі розглядається запит «До» візуально у якості одного запиту (речення), а запит «Після», який розділений точкою з комою, у якості двох.

Запит	Кількість символів	Кількість слів	ARI
«До»	226	45	24.725
«Після»	226	45	13.475

d. **Додавання псевдоніму (AS)**

Адаптовано з методу: «Введення пояснювальної змінної»
*Опис***:** Ключове слово AS, яке є частиною визначення псевдоніма таблиць.
Мотивація: Подібна конструкція не є обов'язковою але її використання робить текст запитів більш читабельним, шляхом підвищення швидкості розуміння яке значення поверне вираз з назвою псевдоніму.
Приклад «до застосування методу»: відсутність псевдонімів для зовнішнього запиту.

```
SELECT tasks_number_per_employee.name, AVG(tasks_number_per_employee.
tasks_number)
FROM(
      SELECT project.name, project_employee.employee_position, COUNT(task.id)
as tasks_number
      FROM project_employee
      JOIN project ON project.id = project_employee.project_id
      JOIN task ON task.project_employee_id = project_employee.id
      GROUP BY project_employee.employee_position, project.name) AS
tasks_number_per_employee
GROUP BY tasks_number_per_employee.name;
```

Приклад «після застосування методу»: додавання псевдонімів до об’єктів:

```
SELECT tasks_number_per_employee.name AS project_name,
AVG(tasks_number_per_employee.tasks_number) AS
average_tasks_number_per_employee
FROM(
      SELECT project.name, project_employee.employee_position, COUNT(task.id)
as tasks_number
      FROM project_employee
      JOIN project ON project.id = project_employee.project_id
      JOIN task ON task.project_employee_id = project_employee.id
      GROUP BY project_employee.employee_position, project.name) AS
tasks_number_per_employee
GROUP BY tasks_number_per_employee.name;
```

Результати метрики:

Запит	Кількість символів	Кількість слів	ARI
«До»	363	69	37.849
«Після»	407	78	42.147

e. Надання операторам повних імен.

Опис: використовувати повні імена операторів, а не їх псевдоніми.

Мотивація: Повне ім'я зробить текст запиту більш зрозумілим, адже деталі додають конкретики. Найбільш корисне це для спеціалістів, які давно не працювали з запитами SQL або тільки навчаються, бо повне ім'я дає додаткову інформацію про конкретику оператору.

Приклад «до застосування методу»: використані псевдоніми операторів.

```
SELECT employee.lastname, task.title, project.name project_name
FROM project_employee
JOIN task ON task.project_employee_id = project_employee.id
JOIN project ON project_employee.project_id = project.id
LEFT employee ON employee_id = employee.id
```

Приклад «після застосування методу»: використані повні імена операторів.

```
SELECT employee.lastname, task.title, project.name AS project_name
FROM project_employee
INNER JOIN task ON task.project_employee_id = project_employee.id
INNER JOIN project ON project_employee.project_id = project.id
OUTER LEFT JOIN employee ON employee_id = employee.id
```

Результати метрики:

Запит	Кількість символів	Кількість слів	ARI
«До»	200	38	22.359
«Після»	221	43	24.277

Спрощення складних запитів

a. **Виділення Common Table Expression (CTE)**

Адаптовано з методу: «Вилучення методу»

Опис: Розбийте складний запит на декілька частин за допомогою CTE.

Мотивація: CTE роблять код більш читабельним, полегшують налагодження запитів. Такі вирази дозволяють посилатися на їх результати кілька разів протягом усього запиту. Зберігаючи результати підзапиту, є можливість повторно використовувати їх у більших запитах. CTE можуть допомогти виконати багаторівневе агрегування, використання CTE для збереження результату агрегації, яку потім можна узагальнити в основному запиті.

Приклад «до застосування методу»: запит без використання CTE.

```
SELECT tasks_number_per_employee.name,
AVG(tasks_number_per_employee.tasks_number)
FROM(
     SELECT project.name, project_employee.employee_position, COUNT(task.id)
as tasks_number
     FROM project_employee
     JOIN project ON project.id = project_employee.project_id
     JOIN task ON task.project_employee_id = project_employee.id
     GROUP BY project_employee.employee_position, project.name) AS
tasks_number_per_employee
GROUP BY tasks_number_per_employee.name;
```

Приклад «після застосування методу»: запит з інкапсульованим підзапитом у якості CTE.

```
WITH tasks_number_per_employee AS
(SELECT project.name, project_employee.employee_position, COUNT(task.id) as
tasks_number
FROM project_employee
JOIN project ON project.id = project_employee.project_id
JOIN task ON task.project_employee_id = project_employee.id
GROUP BY project_employee.employee_position, project.name)

SELECT tasks_number_per_employee.name,
AVG(tasks_number_per_employee.tasks_number)
FROM  tasks_number_per_employee
GROUP BY tasks_number_per_employee.name;
```

Результати метрики:

Запит	Кількість символів	Кількість слів	ARI
«До»	363	69	37.849
«Після»	389	74	21.829

b. **Заміна алгоритму об’єднання таблиць**

Адаптовано з методу: «Заміна алгоритму»
Опис: Заміна умовного виразу WHERE table1.table2Id = table2.Id на конструкцію з використанням INNER JOIN для об’єднання таблиць.
*Мотивація***:** Даний метод дозволяє зробити запит простішим до розуміння за рахунок спеціального виразу, який створено задля цієї операції. Таким чином можна розділити умову та вираз за яким виконується поєднання таблиць. Також це сприяє уникненню застосування неявного CROSS JOIN. У Декартовому приєднанні створюються всі можливі комбінації змінних. Це неефективне використання ресурсів бази даних, оскільки база даних виконала роботу в N разів більше, ніж потрібно. Декартові об’єднання є особливо проблематичними у великомасштабних базах даних,

оскільки декартове об’єднання двох великих таблиць може створити мільярди чи трильйони результатів.

Приклад «до застосування методу»: об’єднання таблиць за ключом у фільтрі.

```
SELECT project.name, project_employee.employee_position, COUNT(employee_id)
FROM project_employee, project
WHERE employee_position = ‘developer’ AND project.id =
project_employee.project_id
GROUP BY project_employee.employee_position, project.name;
```

Приклад «після застосування методу»: об’єднання таблиць за ключом за допомогою JOIN.

```
SELECT project.name, project_employee.employee_position, COUNT(employee_id)
FROM project_employee
JOIN project ON project.id = project_employee.project_id
WHERE employee_position = ‘developer’
GROUP BY project_employee.employee_position, project.name;
```

Результати метрики:

Запит	Кількість символів	Кількість слів	ARI
«До»	204	33	24.186
«Після»	207	34	24.246

Спрощення умовних виразів

a. Перетворення умови на IN вираз

Адаптовано з методу: «Консолідація умовного виразу»

Опис: Якщо обрана умова на одне або декілька значень одного атрибуту, її краще представити у вигляді виразу IN.

Мотивація: легше представити складний умовний вираз у вигляді списку значень, які складають умову.

Приклад «до застосування методу»: складна умова, яка може бути спрощена за допомогою IN.

```
SELECT project.name, project_employee.employee_position, COUNT(employee_id)
FROM project_employee
JOIN project ON project.id = project_employee.project_id
WHERE employee_position = ‘developer’ OR employee_position = ‘QA’ OR
employee_position = ‘manager’
GROUP BY project_employee.employee_position, project.name;
```

Приклад «після застосування методу»: використання IN оператора.

```
SELECT project.name, project_employee.employee_position, COUNT(employee_id)
FROM project_employee
JOIN project ON project.id = project_employee.project_id
WHERE employee_position IN (‘developer’, ‘QA’, ‘manager’)
GROUP BY project_employee.employee_position, project.name;
```

Результати метрики:

Запит	Кількість символів	Кількість слів	ARI
«До»	254	42	27.83
«Після»	218	37	24.821

b. Використання WHERE до/замість HAVING

Адаптовано з методу: «Консолідація умовних виразів»

Опис: Можливість використання підрозділу WHERE для фільтрації зайвих рядків. Лише після видалення нерелевантних рядків, а також після об'єднання цих рядків та їх групування слід включити підрозділ HAVING для фільтрації результату агрегатів.

Мотивація: добра практика використовувати оператори за їх зазначенням, так текст запитів залишається зрозумілим для всіх.

Приклад «до застосування методу»: використання HAVING не за призначенням

```
SELECT employee.firstname +' '+ employee.lastname as name, COUNT(task.id) AS
Count
FROM project_employee
JOIN task ON task.project_employee_id = project_employee.id
JOIN employee ON employee_id = employee.id
WHERE task.task_status_id <= 3
GROUP BY employee.firstname +' '+ employee.lastname
HAVING employee.firstname +' '+ employee.lastname LIKE(%M);
```

Приклад «після застосування методу»: фільтрування рядків перед групуванням

```
SELECT employee.firstname +' '+ employee.lastname as name, COUNT(task.id) AS
Count
FROM project_employee
JOIN task ON task.project_employee_id = project_employee.id
JOIN employee ON employee_id = employee.id
WHERE task.task_status_id <= 3 AND employee.firstname +' '+ employee.lastname
LIKE(%M)
GROUP BY employee.firstname +' '+ employee.lastname;
```

Результати метрики:

Запит	Кількість символів	Кількість слів	ARI
«До»	270	51	29.005
«Після»	259	44	28.295

c. Заміна LIKE на оператор рівності

Опис: LIKE порівнює символи та може бути об'єднаний у пару з символами шаблонами, такими як «%», але оператор «=» порівнює рядки та числа до точних збігів. Якщо є потреба точно порівняти значення, не потрібно використовувати «=».

Мотивація: LIKE не досить очевидна операція, стандартна практика порівняння значень - використання операції «=». Оператор порівняння може скористатися індексованими стовпцями, що робить виконання запиту більш ефективним.

Приклад «до застосування методу»: використання регулярного виразу для перевірки на рівність

```
SELECT project.name, project_employee.employee_position, COUNT(employee_id)
FROM project_employee
```

```
JOIN project ON project.id = project_employee.project_id
WHERE employee_position IN ('developer', 'QA', 'manager') AND
employee.firstname +' '+ employee.lastname LIKE(Mike Miller)
GROUP BY project_employee.employee_position, project.name
```

Приклад «після застосування методу»: без використання регулярного виразу

```
SELECT project.name, project_employee.employee_position, COUNT(employee_id)
FROM project_employee
JOIN project ON project.id = project_employee.project_id
WHERE employee_position IN ('developer', 'QA', 'manager') AND
employee.firstname +' '+ employee.lastname = 'Mike Miller'
GROUP BY project_employee.employee_position, project.name;
```

Результати метрики:

Запит	Кількість символів	Кількість слів	ARI
«До»	268	45	29.121
«Після»	264	44	28.83

d. Використання EXISTS або JOIN замість IN

Опис: Якщо є потреба перевірити наявність значень у таблиці, краще використовувати EXISTS замість IN. Це також справедливо для виразів NOT EXIST. Оператор JOIN, також надає можливість виконати фільтрацію даних при умові існування значень в іншій таблиці.

Мотивація: Використання операторів за їх призначенням дозволяє розробникам швидше розуміти текст запиту. Виконання EXISTS припиниться, як тільки знайдено значення пошуку, умова IN буде сканувати всю значення в таблицях.

Приклад «до застосування методу»: запит з використанням IN

```
SELECT *
FROM Customers
WHERE ID IN (
      SELECT CustomerID
      FROM Orders)
```

Приклад «після застосування методу»: запит з використанням EXIST

```
SELECT *
FROM Customers
WHERE EXISTS (
      SELECT *
      FROM Orders
      WHERE Orders.CustomerID = Customers.ID)
```

Приклад «після застосування методу»: запит з використанням INNER JOIN

```
SELECT Customers.*
FROM Customers
      JOIN Orders ON Customers.ID = Orders.CustomerID
```

Результати метрики:

Запит	Кількість символів	Кількість слів	ARI
«До»	54	10	9.004

«Після з EXIST»	78	13	13.33
«Після з JOIN»	67	11	12.758

Робота з коментарями

a. Додавання або видалення коментарів

Опис: Додавати опис функціоналу у коментарі за необхідністю. Видалення коментарів, якщо в них немає потреби.

Мотивація: Якщо запит є складним для розуміння, частини запиту не є очевидними, вставка коментарів надасть розробнику розуміння того що відбувається у запиті. У разі якщо запит маленький та простий видалення коментарів спростить текст запиту.

Приклад «до застосування методу»: невдале використання коментарів.

```
-- to do: use one code style
SELECT CountTasks.name, AVG(CountTasks.Count) -- add alias
FROM(
-- to do: extract as CTE
     SELECT project.name, project_employee.employee_position,
          COUNT(task.id) as Count -- change alias name
     FROM project_employee
     JOIN project ON project.id = project_employee.project_id
     JOIN task ON task.project_employee_id = project_employee.id
     GROUP BY project_employee.employee_position, project.name) AS
CountTasks
GROUP BY CountTasks.name;
```

Приклад «після застосування методу»: додано опис запиту, видалені нагадування про реалізацію

```
-- Description: Calculate the average number of tasks on each project per each
employee
WITH tasks_number_per_employee AS
     (SELECT project.name, project_employee.employee_position,
COUNT(task.id) AS tasks_number
     FROM project_employee
          JOIN project ON project.id = project_employee.project_id
          JOIN task ON task.project_employee_id = project_employee.id
     GROUP BY project_employee.employee_position, project.name)

SELECT tasks_number_per_employee.name AS project_name,
     AVG(tasks_number_per_employee.tasks_number) AS
average_tasks_number_per_employee
FROM tasks_number_per_employee
GROUP BY tasks_number_per_employee.name;
```

Результати метрики:

Запит	Кількість символів	Кількість слів	ARI
«До»	361	71	38.018
«Після»	504	96	27.298

Висновки

Отже, в результаті був створений перелік методів рефакторингу, які вже готові до використання розробниками, яким необхідно покращити якість текстів SQL запитів. Методи розділені на групи в залежності до цілі їх використання. Загалом методи створені для того, щоб покращити читабельність, пришвидшити розуміння коду іншими розробниками, але й деякі з них ще позитивно пливають на ефективність виконання, що не було ціллю даної роботи.

Результати які були отримані за допомогою Automated Readability Index показали, що здебільшого методи рефакторингу додають інформацію до запиту, що може негативно вплинути на формальну читаємість тексту, але додаткова інформація сприяє поліпшенню розуміння тесту запиту. Для підтримки балансу зрозумілості специфікації запиту та його чатабельності, можна комбінувати методи форматування та рефакторингу запитів.

Список використаної літератури

1. Фаулер М., Бек К., Брант Д., Опайдак У., Робертс Д. Рефакторинг: Улучшение проекта существующего кода. – Санкт-Петербург: ООО «Диалектика», 2019 – 448 с.
2. Bohm L., Refactoring legacy T-SQL for improved performance: Modern practices for SQL Server applications – Apress, Berkeley, CA, 2020 – 236 c.
3. Muhammad U.T., Muhammad B.B., Muhammad B., Adnan S., Code Readability Management of High-level Programming Languages: A Comparative Study, International Journal of Advanced Computer Science and Applications, 2020 – 8 c.
4. Buse, R.PL., Westley R.W., A metric for software readability, Proceedings of the 2008 international symposium on Software testing and analysis, pp. 121-130, Seattle, WA, USA, July 20-24, 2008.
5. Donald K. B., Oracle Silver Bullets: Real-world Oracle Permormance Secrets – Rampant Techpress, 2005 – 200 c.
6. Senter, R.J.; Smith, E.A. (November 1967). ”Automated Readability Index”. Wright-Patterson Air Force Base: iii. AMRL-TR-6620. Retrieved March 18, 2012.
7. S. Fakhoury, D. Roy, A. Hassan and V. Arnaoudova, ”Improving Source Code Readability: Theory and Practice,” 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC), pp. 2-12, Montreal, QC, Canada, 2019.
8. Pahal, Ankit, and Rajender S. Chillar. ”Code Readability: A Review of Metrics for Software Quality.”International Journal of Computer Trends and Technology (IJCTT) – Volume 46 Number 1- April 2017