Хмельницький національний університет

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Метод виявлення семантичних помилок у коді на декларативній мові програмування SQL за допомогою статичного аналізу

Назва теми

Рівень вищої освіти Другий (магістерський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.190155.01.04.ПЗ

Виконав студент 2 курсу, група ІПЗм-22-1	Підпис	Ігор НЕТРЕБА Ім'я, ПРІЗВИЩЕ
Керівник канд. пед. наук, доцент. Науковий ступінь, звання	Підпис	Наталія ПРАВОРСЬКА Ім'я, ПРІЗВИЩЕ
Нормоконтролер канд. техн. наук. доцент	Підпис	Оксана ЯШИНА Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри
інженерії програмного забезпечення — Підпис — Леонід БЕДРАТЮК (Ім'я, ПРІЗВИЩЕ)

Дата 8 грудня 2023 р

Хмельницький 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Інженерії програмного забезпечення

Рівень вищої освіти Другий (магістерський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри ІПЗ

Л. П. Бедратюк

01.09.2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Нетребі Ігорю Вікторовичу

Прізвище, ім'я, по батькові здобувача

1. Тема роботи Метод виявлення семантичних помилок у коді на декларативній мові програмування SQL за допомогою статичного аналізу

Керівник роботи Праворська Наталія Іванівна, канд. пед. наук, доцент

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 15.08.2023 р. № 30

2. Строк подання студентом роботи на кафедру 01.12.2023 р.

3. Вихідні дані до роботи Матеріали науково-дослідної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

1. Дослідження предметної області та постановка задачі

2. Помилки в SQL-запитах

3. Стратегія виявлення семантичних помилок SQL

4. Інструмент статичного аналізу для виявлення семантичних помилок

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Презентаційні матеріали (слайди), таблиця з ручним аналізом запитів

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Антиплагіат	Форкун Ю. В., доцент	28.11.23	2.12.2023
Нормоконтроль	Яшина О.М., доцент		

7. Дата видачі завдання «01» вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Вивчення предметної області; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження; формування логістичної структури кваліфікаційної роботи	01.09 –07.09.2023	
2 Робота над розділом 1 кваліфікаційної роботи – аналіз Інтернет-джерел та літератур; аналіз наявних методів за темою роботи; визначення методологічних підходів до вирішення задачі; постановка задач дослідження та висновки	08.09 –18.09.2023	
3 Робота над розділом 2 кваліфікаційної роботи – аналіз сучасних емпіричних досліджень з виявлення та тестування коду; висновки до розділу	19.09 –01.10.2023	
4 Робота над науковими статтями	01.10 –23.10.2023	
5 Робота над розділом 3 кваліфікаційної роботи – детальна розробка та характеритика евристик для аналізу; висновки до розділу	24.10 – 06.11.2023	
6 Робота над розділом 4 кваліфікаційної роботи – аналіз розробленого програмного забезпечення та методики, виділення загроз валідності та рекомендацій з удосконалення; висновки до розділу	07.11 – 16.11.2023	
7 Попередній захист кваліфікаційної роботи	17.11.2023	
8 Узгодження постановки задачі, отриманих результатів та висновків; оформлення згідно вимог пояснювальної записки; написання загальних висновків, вступу, оформлення переліку джерел посилання та додатків;	18.11 – 30.11.2023	
9 Перевірка роботи на наявність плагіату; нормоконтроль; брошурування пояснювальної записки; підготовка супровідних документів	01.12 –08.12.2023	
10 Підготовка до захисту кваліфікаційної роботи	з 01.12.2023	

Студент _____ Ігор Кострба
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Наталія Праворська
Підпис Ім'я, ПРІЗВИЩЕ

РЕФЕРАТ

Тема дипломної роботи: Метод виявлення семантичних помилок у коді на декларативній мові програмування SQL за допомогою статичного аналізу.

Автор роботи: Нетреба Ігор Вікторович.

Керівник проекту: Праворська Наталія Іванівна.

Пояснювальна записка: 155 с., 47 рис., 1 табл., 3 дод., 50 джерел.

ПОМИЛКА, СЕМАНТИЧНА ПОМИЛКА, ДЕКЛАРАТИВНА МОВА ПРОГРАМУВАННЯ, SQL, АНАЛІЗ, JAVA, SQL-ЗАПИТ, СИСТЕМА УПРАВЛІННЯ БАЗАМИ ДАНИХ.

Метою роботи є розробка методики виявлення семантичних помилок у коді на декларативній мові програмування SQL за допомогою статичного аналізу.

У дипломній роботі детально проаналізовано помилки, які виникають під час написання SQL-запитів. Досліджено вплив на вихідний результат. Проведена оцінка існуючих методик та робіт інших дослідників в даній області, на базі яких було реалізовано нова методика, яка вирішує проблему перевірки запитів на помилки. Також була проаналізована частота семантичних помилок у запитах, та кореляція між складністю запиту та кількістю семантичних помилок.

Для реалізації тестового середовища було використано мову програмування Java, бібліотека JsqlParser для парсингу запитів.

Практична значимість отриманих результатів полягає у розробці метода виявлення семантичних помилок у коді на декларативній мові програмування SQL за допомогою статичного аналізу.

1.12.2023
Дата

Підпис

ABSTRACT

Master's thesis: «Detection Method of Semantic Errors in Declarative SQL Programming Language Code Using Static Analysis».

Author: Ihor Viktorovych Netreba.

Head of work: Natalia Ivanivna Pravorska.

Master's thesis consist of: 155 p., 47 pic., 1 tab., 3 add., 50 src.

ERROR, SEMANTIC ERROR, DECLARATIVE PROGRAMMING LANGUAGE, SQL, ANALYSIS, JAVA, SQL QUERY, DATABASE MANAGEMENT SYSTEM.

The purpose of this thesis is to develop a methodology for detecting semantic errors in code written in the declarative SQL programming language using static analysis.

The thesis thoroughly analyzes errors that occur during the writing of SQL queries. The impact on the output result is investigated. An evaluation of existing techniques and works of other researchers in this field is conducted, based on which a new methodology is implemented to address the problem of query error checking. The frequency of semantic errors in queries and the correlation between query complexity and the number of semantic errors were also analyzed.

To implement the testing environment, the Java programming language and the sqlParser library for query parsing were used.

The practical significance of the obtained results lies in the development of a method for detecting semantic errors in code written in the declarative SQL programming language using static analysis.

1.12.2023
Дата

Підпис

ЗМІСТ

Перелік скорочень 8
Вступ 9
1 Дослідження предметної області та постановка задачі 12
1.1 Аналіз предметної області, останніх досліджень та джерел 12
1.2 Проблема пошуку семантичних помилок 12
1.3 Проблема написання правильних запитів 19
1.4 Постановка задачі та висновок 24
2 ПОМИЛКИ В SQL-ЗАПИТАХ 26
2.1 Семантичні помилки 26
2.2 Емпіричні дослідження в сфері навчання SQL 27
2.3 Емпіричні дослідження SQL-запитів 27
2.4 Тестування SQL-запитів 30
2.5 Збір даних 32
2.6 Висновок 37
3 СТРАТЕГІЯ ВИЯВЛЕННЯ СЕМАНТИЧНИХ ПОМИЛОК SQL 39
3.1 Семантичні помилки в запитах 39
3.2 Стратегії виявлення семантичних помилок 42
3.2.1 Несумісна змінна кортежа 43
3.2.2 Постійний вихідний стовбець 44
3.2.3 Дублювання вихідного стовпця 45
3.2.4 Непотрібна інструкція JOIN 46
3.2.5 Ідентичні змінна кортежа 47
3.2.6 Порівняння з NULL 48
3.2.7 Відсутня інструкція JOIN 49
3.2.8 Некорельовані підзапити EXISTS 50
3.2.9 Ділення на нуль 52
3.2.10 Дивна інструкція HAVING 53
3.2.11 Дивні символи підстановки без LIKE 55
3.2.12 Неспівпадіння в підзапиті SELECT 56

3.2.13 Неефективна інструкція HAVING 57
3.2.14 Непотрібна умова ORDER BY 58
3.2.15 Непотрібна інструкція GROUP BY 59
3.2.16 Непотрібна умова UNION 61
3.2.17 Непотрібна фраза GROUP BY 61
3.2.18 Непотрібне загальне порівняння 63
3.2.19 Використання LIKE без символів підстановки 64
3.2.20 Непотрібний список SELECT в EXISTS 65
3.2.21 Непотрібне сканування індексу 66
3.2.22 Непотрібний DISTINCT в агрегаціях 68
3.2.23 Непотрібний атрибут GROUP BY 69
3.2.24 Непотрібний аргумент COUNT 70
3.3 Висновок 72
4 ІНСТРУМЕНТ СТАТИЧНОГО АНАЛІЗУ ДЛЯ ВИЯВЛЕННЯ СЕМАНТИЧНИХ ПОМИЛОК 73
4.1 Реалізація 73
4.2 Перевірка 75
4.3 Результати 77
4.3.1 Поширеність семантичних помилок у запитах SQL 77
4.3.2 Частота семантичних помилок у запитах SQL 79
4.3.3 Кореляція між складністю запиту та кількістю семантичних помилок 81
4.4 Загрози для валідності 83
4.5 Рекомендації та удосконалення 85
4.6 Висновок 86
Висновок 88
Перелік джерел посилання 91
Додаток А Ручний аналіз запитів 96
Додаток Б Копії наукових публікацій 136
Додаток В Презентаційні матеріали 147

ПЕРЕЛІК СКОРОЧЕНЬ

SQL	–	structured query language
СУБД	–	система управління базами даних
IT	–	інформаційні технології
IDE	–	integrated development environment
LDA	–	latent dirichlet allocation
ISO	–	international organization for standardization
ANSI	–	american national standards institute

Вступ

В сучасному світі великої кількості даних, бази даних та мови запитів SQL відіграють ключову роль у забезпеченні ефективного зберігання, організації та взаємодії з інформацією. Однак, незважаючи на зростання обсягів знань та інструментів для роботи з SQL, розробники і аналітики нерідко стикаються з семантичними помилками, які можуть призвести до неправильних результатів запитів та складнощів у роботі з даними.

Мова структурованих запитів, також відома як SQL, являє собою предметно-орієнтовану мову програмування, яка використовується для управління і взаємодії з реляційними СУБД. Крім того, це одна з перших мов, які використовували реляційну модель, представлені Коддом [12] у його статті. Пізніше, це стало стандартом ANSI і ISO, що призвело до того, що SQL вважається найбільш поширеною мовою баз даних і більше 56% розробників використовують її.

Основним аспектом семантичних помилок є результат колиSQL-запит синтаксично вірний, але його смислове значення не відповідає очікуваному. Однією з основних причин є неправильне розуміння бізнес-логіки та вимог, які передаються через SQL-запити.

До типових семантичних помилкок можна віднести: типові семантичні помилки та їх уникнення, помилки при об'єднанні таблиць, неправильне використання агрегатних функцій, використання невірного порядку сортування, ORDER BY, неправильне використання аліасів, неправильне використання підзапитів, невірне розташування умов в JOIN, тощо.

Існує незаперечний факт, що SQL використовується не тільки в IT, але й в інших сферах, наприклад: банківські справи, бухгалтерський облік, авіація, торгівця, тощо. Це робить згадану мову однією з найпоширеніших і в той же час, це означає, що багато людей з різним рівнем підготовки і досвідом розробляють і використовують SQL-запити. В залежності від рівня розуміння SQL, деякі користувачі можуть напряму використовувати запити з форумів або інших веб-сайтів питань та відповідей, на кшталт StackOverflow. Однак існує ризик, якщо ці

запити містять помилки. Через цю причину важливо мати інструмент, який може допомогти розробникам виявляти не тільки синтаксичні помилки в SQL-запитах, але й семантичні, подібно до того, як сучасні інтеграційні середовища розробки пропонують переформування коду для різних інших мов програмування.

В наш час інструменти, які існують для SQL, в основному орієнтовані на виявлення тільки синтаксичних помилок у запитах, котрі пишуть розробники. Хоча, це ще важливий аспект, запити також варто перевіряти і на наявність семантичних проблем, оскільки такі типи проблем можуть мати великий вплив на продуктивність системи, як буде пояснено пізніше. Крім того, для більшості існуючих інструментів потрібна повна схема бази даних для аналізу запитів і виявлення потенційних семантичних помилок. Ще одним важливим фактором є те, що більшість сучасних інструментів також не можуть виявляти проблеми в динамічно генерованих SQL-запитах, що дуже обмежує, адже додатки можуть створювати запити під час виконання в залежності від різних вхідних даних.

В рамках цієї роботи було виявлено відсутність інструментів з відкритим програмним кодом, направлених на виявлення семантичних помилок в запитах SQL. Відсутність такого інструментарію, в свою чергу ускладнює виявлення таких проблем розробниками.

Запити, які містять такого типу проблеми вимагають більше знань і більш глибшого розуміння основних принципів SQL для їх вирішення. Крім того в більшості випадків навіть після того, як формулювання запиту вірне, розробники, як правило, пропускають етап перевірки потенційних семантичних помилок. Тому, наявність автоматизованих інструментів для перевірки, може сильно допомогти в розв’язанні такого роду проблем.

Мета представленої роботи двояка. В першу чергу запропоновано 24 перевірених евристик для виявлення найбільш поширених семантичних помилок, які зустрічаються в SQL-запитах, на основі даних попередніх розробників. Після чого було впроваджено інструмент, використовуючи запропоновану евристику, для виявлення семантичних проблем в SQL-запитах і виміряна точність інструменту, виконуючи чималий ручний аналіз великої кількості запитів. По-

друге, було досліджено поширеність цих семантичних помилок у двох різних наборах даних SQL, один з яких надано в статті Кастейна [3], який містить запити трьох різних проектів з відритим кодом, які відслідковуються на GitHub, а другий був створений для представленої роботи, які були зібрані з повідомлень на StackOverflow.

Інструмент статичного аналізу, який розроблено на основі правил в процесі написання роботи, спроможний виявляти семантичні помилки в запитах з точністю до 95%. Крім того було виявлено, що зі всіх запитів більше ніж 19% містили хоча б одну семантичну помилку в своєму формулюванні. Також було з’ясовано, що складність запитів напряму впливає на кількість семантичних помилок.

1. Дослідження предметної області та постановка задачі

1.1 Аналіз предметної області, останніх досліджень та джерел

У роботі Брасс та Голберг [9] та Брасс та ін. [10] були першими, хто зосередився на класі SQL-запитів, які правильні з синтаксичної точки зору, але містять семантичні проблеми, незалежно від завдання запиту. Автори класифікують помилки SQL на дві групи, одна з яких пов'язана із синтаксичними проблемами, а інша – семантичними. У першій групі система управління базою даних виявить помилку, оскільки запит взагалі може бути невиконаний. Ці помилки зазвичай легше виявити та виправити, оскільки розробник одразу отримує повідомлення про помилку або попередження при спробі виконати запит. У другій групі запити коректні, але не дають очікуваних результатів. Ці типи помилок важче виявити та виправити, оскільки проблема не очевидна відразу.

У своїй роботі автори додатково класифікують семантичні помилки на дві групи, для однієї з яких завдання має бути відоме заздалегідь, а для іншого достатньо одного SQL-запиту для виявлення семантичних проблем. Потім основна увага приділяється другій групі, і це також стосується даної тези. Основний внесок їх роботи є таксономією семантичних помилок, які часто з'являються в SQL запитах, зібраних з домашніх завдань та екзаменаційних матеріалів для одного з курсів баз даних в Університеті Галле.

Крім цього списку, Брасс і Голдберг [8] провели ще одне дослідження для аналізу двох інших конкретних типів семантичних проблем, що виникають у SQL запитах, а саме неузгоджених умов і запитів, які можуть генерувати помилки часу виконання. Однією з потенційних загроз достовірності їхньої роботи є обмежений розмір набору даних, на якому проводилося дослідження, зокрема, помилки було вилучено із шести запитів SQL, що виявились на двох іспитах із приблизно 150 учасниками. Варто зазначити, що Голдберг [20] провів дослідження з використанням такої самої таксономії семантичних помилок на більшому наборі даних, проте запити все ще були вилучені з попередніх досліджень. У цій роботі,

окрім реалізації інструменту для виявлення цих семантичних помилок, також проводиться дослідження великого набору даних запитів, вилучених із реальних додатків із StackOverflow, що дозволяє краще зрозуміти, як виникають ці проблеми.

Велика таксономія помилок в коді SQL також представлена Карвіном [23]. У своїй роботі автор представляє антипаттерни і підводні камені SQL, які найчастіше зустрічаються. Наведений список проблем також враховує найбільш поширені проблеми SQL, з якими автор стикався, відповідаючи на різні питання на он-лайн форумах, у списках розсилки та групах новин за більш ніж 15-річний період.

Роботи по дослідженню семантичних помилок у SQL є досить актуальною сферою, тому що правильна робота запитів є ключовою для ефективної роботи баз даних.

Крім зазначених досліджень, також існують інші підходи до вивчення семантичних помилок у SQL та інших мовах програмування. Наприклад, деякі науковці аналізують конкретні типи семантичних помилок, сюди входять: неузгоджені умови, помилки в запитах, оскільки вони що можуть призводити до помилок часу виконання, або проблеми з виборкою даних.

Ще однією областю досліджень є аналіз семантичних помилок у реальних додатках. Наприклад, великі набори даних запитів, вилучених з реальних проектів або від користувачів, можуть служити об'єктом досліджень для виявлення і аналізу семантичних помилок.

Деякі дослідники також розробляють інструменти автоматичного виявлення семантичних помилок. Такі інструменти можна використовувати для аналізу SQL-запитів щоб надавати розробникам підказки та виявляти потенційні проблеми.

Усі ці дослідження спрямовані на поліпшення розуміння семантичних проблем у SQL-запитах, розробку ефективних методів їх виявлення та вирішення, а також на підвищення якості роботи з базами даних.

Деякі дослідження також можуть акцентувати увагу на аспектах, пов'язаних із навчанням баз даних. Автори робіт аналізували типові помилки для запитів, які робили студенти при написанні SQL-запитів, і розробляли методи навчання, спрямовані на уникнення цих помилок. Це допомогло вдосконалити якість базової освіти по аналізу та роботі з базами даних і в підготовці майбутніх розробників до ефективної роботи з мовою програмування SQL.

Слід відзначити, що багато з цих досліджень використовували методи аналізу великих обсягів даних та машинного навчання для виявлення шаблонів та аномалій у написаних SQL-запитах. Це дозволяє автоматизувати процес виявлення семантичних помилок і розширює можливості вдосконалення якості роботи з базами даних.

Важливим аспектом досліджень є також розробка інструментів, які можуть допомагати розробникам виявляти та виправляти семантичні помилки у їх SQL-коді. Це може включати в себе розробку плагінів для інтегрованих середовищ розробки, онлайн-сервісів аналізу SQL-коду та інші інструменти, спрямовані на полегшення роботи з базами даних та вдосконалення якості програмного забезпечення. Загальною метою цих досліджень було забезпечення надійності та ефективності в роботі з базами даних за допомогою виявлення та вирішення семантичних помилок, а також підготовка майбутніх фахівців до впевненої роботи з SQL.

Ще однією актуальною темою досліджень є розробка методів автоматичного аналізу та виправлення семантичних помилок в SQL-коді. Автоматизація цього процесу може сприяти не лише підвищенню продуктивності розробників, але і забезпеченню вищого рівня безпеки та точності у роботі з базами даних.

Також є дослідження які розглядають вплив семантичних помилок на безпеку баз даних, тому що, невірно сформульовані запити можуть привести до витоку конфіденційної інформації або інших серйозних проблем безпеки. Розуміння цього аспекту роботи з базами даних може сприяти розробці більш

безпечних систем управління базами даних та інструментів для їх обслуговування.

Інші напрямки досліджень включали аналіз впливу різних фреймворків та архітектурні підходи на виникнення семантичних помилок, вивчення взаємодії баз даних з іншими компонентами програмних систем, а також розробку нових методів навчання для розробників щодо уникнення семантичних помилок у SQL-коді.

В цілому, область досліджень семантичних помилок у SQL-коді досить актуальна та має досить важливе значення для підвищення ефективності та безпеки роботи з базами даних. Проведення додаткових експериментів та розробка нових методів можуть сприяти подальшому вдосконаленню роботі у цій області.

У даній роботі антипатерни віднесено до однієї з наступних категорій: проектування логічної бази даних, проектування фізичної бази даних, питання, пов'язані із запитами, і, нарешті, розробки додатків. У категорії, пов'язаній із запитами, можна побачити найпоширеніші типи семантичних проблем, які у SQL-запитах (також визначених як помилки), зокрема, список із шести антипатернів запитів: страх перед невідомим, пов'язані з особливою природою NULL в SQL, неоднозначні групи, що пов'язано з використанням інструкції GROUP BY, антипаттерн випадкового вибору, пов'язаний з проблемою, коли з бази даних вибирається випадковий рядок, і, нарешті, проблема з неявними стовпцями, яка відноситься до використання, виберіть всі зірочки (*) в операторах SELECT замість надання явного переліку вибору. Велика кількість подальших досліджень ґрунтується на таксономії, представленій у цій роботі, з метою як аналізу, так і кращого розуміння того, як і чому ці проблеми виникають у SQL-запитах.

1.2 Проблема пошуку семантичних помилок

Пошук семантичних помилок потрібно робити, коли запит містить правильні синтаксичні конструкції, але неправильно виражає зміст. Сюди може

входити: невірне використання слів, або конструкцій, нечітке висловлення або неправильні асоціації. Щоб вирішувати такі проблеми потрібно ретельно зрозуміти контекст для якого був написаний той чи інший запит.

Відслідковування семантичних помилок це досить складне завдання через те, що це вимагає не лише знань мови, а й розуміння контексту запиту. Семантичні помилки можуть виникати через неправильний вибір слів, конструкцій або недостатню ясність у висловленні, що робить їх важкими для виявлення та відслідковування. Також враховується те, що семантика залежить від контексту, який може бути неоднозначним у тому чи іншому випадку.

Додатково, семантичні помилки можуть виникати через різноманітні стилі написання коду, діалекти, термінологію або інші чинники, які ускладнюють процес правильного розуміння того чи іншого запиту. Особливо в інтернет-спілкувані може бути багато нюансів, таких як використання скорочень, які важко враховувати при виявленні семантичних помилок.

Пошук і виправлення семантичних помилок вимагає від розробника великої уваги до деталей, глибокого розуміння теми та здатності виражати думки точно та чітко. Також, семантичні помилки можуть бути важко виявити через особливості написання коду та різницю в сприйнятті тексту між тим хто написав код і тим хто його буде читати. Однаковий запит може сприйматись різними людьми по-різному залежно від їхнього досвіду та інших факторів.

Семантична помилка є складним явищем, і її виявлення вимагає здатності до аналізу тексту на різних рівнях розуміння, також значний внесок вніс розвиток технологій у сфері розуміння та аналізу мов програмування і штучного інтелекту який допомагає у виявленні семантичних помилок. Однак, навіть з великими досягненнями в цих областях, додаткам та програмам іноді складно виявити нюанси семантики.

Загальною тенденцією є постійне вдосконалення інструментів для виявлення та виправлення семантичних помилок, але повністю автоматизованого підходу зараз немає, тому що є складність семантики мови та індивідуальні особливості при написані коду. Крім того, існує виклик у виявленні семантичних

помилок в мові тому що з часом вона змінюється і вдосконалюється. Мова постійно еволюціонує, розвивається, і нові вирази чи синтаксичні конструкції можуть виникати, а це в свою чергу створює додаткові труднощі для аналізу та виправлення коду.

Здатність виявляти та коригувати семантичні помилки вимагає не лише розуміння мови, але і актуальних знань від розробника у конкретному контексті та конкретній мові програмування.

Одним із факторів, що робить виявлення семантичних помилок важким, є індивідуальність стилів написання коду. Люди використовують мову різними способами, і те, що для одного може бути сприйматись нормально, для іншого може виглядати як помилка. Таким чином, важливо враховувати індивідуальні особливості автора при аналізі коду.

Узагальнюючи, семантичні помилки виникають з різноманітних причин, і їх виявлення є викликом через комплексність мовного вираження, змінність мови та індивідуальність стилів авторів.

У майбутньому подальше вдосконалення інструментів штучного інтелекту та технологій обробки мови допоможе полегшити виявлення семантичних помилок, але важливо залишати простір для людського емоційного і творчого розуміння мови.

При пошуку семантичних помилок можуть виникати різні проблеми, і важливо враховувати різні аспекти тексту. Нижче наведено список можливих проблем, які можуть виникнути при аналізі семантики тексту:

1. Неоднозначні висловлення або ж логічна хиба – в тексті можуть бути фрази або конструкції, які мають кілька можливих тлумачень.

2. Протиріччя в інформації – різні частини запитів можуть містити протиріччя або несумісну інформацію.

3. Використання невірних термінів або понять – може неправильно використовувати терміни, що призводить до спотворення істини.

4. Неповна або невірна інформація – недостатньо повна або хибна інформація може призвести до неправильних висновків розробника.

5. Відсутність зв'язку між ідеями – може виникнути проблема, якщо ідеї в запиті не пов'язані чи не виражені послідовно.

6. Некоректне використання стилістичних засобів – невірне використання стилістичних засобів може впливати на емоційний тон або загальний сенс запиту.

7. Недостатня деталізація або перевищення деталей – запит може бути або занадто загальним, не надаючи необхідних деталей, або занадто деталізованим, ускладнюючи сприйняття.

8. Порушення логічної структури – логічна структура запиту може бути порушена, що робить важчим розуміння послідовності.

При аналізі семантичних помилок важливо враховувати контекст та спробувати зрозуміти інтенції автора, а також взаємозв'язок між різними частинами тексту.

Таким чином рішенням цих проблем може бути:

1. Неоднозначність висловлення – уточнити фрази або конструкції, які можуть бути розуміні різними способами. Використовувати більше конкретних термінів або надайте додатковий контекст для уникнення непорозумінь.

2. Протиріччя в інформації – перевірити джерела інформації та виправити протиріччя. Якщо можливо, скористатися додатковими джерелами для підтвердження точності інформації.

3. Використання невірних термінів або понять – означити терміни або поняття, щоб уникнути спотворення сенсу. Перевірити словники або проконсультуватись з експертами для правильного використання термінології.

4. Неповна або невірна інформація – перевірити достовірність інформації перед включенням її у текст. Надавати джерела або додаткові дані для підтвердження правильності інформації.

5. Відсутність зв'язку між ідеями – використовути логічні зв'язки та переходи між ідеями для створення послідовності. Додавати введення та висновок для кращого розуміння контексту.

6. Некоректне використання стилістичних засобів – обережно вибирати стилістичні елементи, що відповідають тону тексту. Переконайтися, що їхнє використання відображає ваші інтенції та не спотворює сенсу.

7. Недостатня деталізація або перевищення деталей – потрібно забезпечити необхідний рівень деталізації для зрозуміння, уникаючи при цьому перевищення зайвих деталей. Зосередьтитись на ключових аспектах.

8. Порушення логічної структури – переглянути текст і переконайтися, що ідеї викладені послідовно та логічно. Використовувати заголовки та абзаци для організації ідей.

Загальний огляд тексту вказує на кілька ключових аспектів, які вимагають уваги та корекцій. Розглянуті проблеми, такі як неоднозначність висловлення, протиріччя в інформації, використання невірних термінів, недостатня деталізація і порушення логічної структури, свідчать про необхідність удосконалення тексту з точки зору ясності, точності та послідовності.

Розв'язання цих проблем передбачає уточнення та узгодження семантичного змісту, перевірку і підтвердження точності інформації, правильне використання термінології, баланс між деталізацією та конкретністю, створення логічної структури, а також уважне вибір стилістичних засобів. Вирішення цих питань допоможе підвищити якість тексту, забезпечити його легше сприйняття та зрозуміння розробником.

1.3 Проблема написання правильних запитів

Звичайно, загалом важко стверджувати, що синтаксично правильний запит є семантично неправильним, якщо не знати завдання, для якого був написаний запит. Однак запити можна вважати «імовірно не вірними», якщо вони надто складні. Припустімо, що користувач написав запит Q, і існує еквівалентний запит Q(t), який є значно простішим і може бути отриманий із Q шляхом видалення певних частин. Можуть бути наступні причини, чому користувач не написав Q(t):

– користувач знав, що Q(t) не є правильним формулюванням поставленого завдання. У цьому випадку Q, звичайно, також не є правильним, але помилка може бути прихована в більш складному запиті, так що користувач цього не усвідомлює. Попередження, звичайно, було б корисним у цьому випадку;

– користувач не знав, що Q(t) еквівалентне. Оскільки Q(t) не є зовсім іншим запитом, а є результатом Q шляхом видалення певних частин, це свідчить про те, що користувач, ще не володіє SQL. Знову ж таки, попередження було б корисним. Часто простіший запит справді виконується швидше (наприклад, оптимізатор запитів Oracle не видаляє непотрібні об'єднання);

– користувач знав, що Q(t) еквівалентний, але він або вона вірив, що Q працюватиме швидше. Оскільки SQL є декларативною мовою, це має бути лише крайній засіб. З сучасними оптимізаторами, це не повинно траплятися часто на практиці. Якщо це необхідно, можливо, має бути якийсь коментар, і це також можна використати для вимкнення попередження. Хоча відомо принаймні один випадок, коли більш складний запит справді виконується швидше на Oracle 9i, SQL не дає жодних гарантій щодо того, як запит буде оцінено. Таким чином, у наступній версії Oracle або при використанні іншої СУБД може статися, що відносна швидкість Q і Q(t) різко зміниться;

– користувач знав, що Q(t) еквівалентний, але він або вона думав, що Q буде зрозумілішим для людини, яка читає, і простіше підтримувати. Потрібно бути обережним, щоб визначити можливі перетворення від Q до Q(t), щоб цього не сталося.Наприклад, було б зрозуміліше використовувати явні змінні кортежу в посиланнях на атрибути, навіть якщо назва атрибута унікальна. Звичайно, видалення змінної кортежу в цьому випадку, не може вважатися створенням іншого, коротшого запиту. Ще більш очевидно, що ми не вимагатимемо, щоб значущі імена змінних кортежу були скорочені або мали коментарі. Також певні додаткові ключові слова (наприклад, «AS») є справою смаку. На жаль, це означає, що кожне можливе «скорочувальне перетворення» SQL-запитів має розглядатися окремо (як це зроблено нижче на рис. 1.1).

Розглянемо детальніше кожну з цих причин:

– Неусвідомлення помилок: користувач може не усвідомлювати наявність помилок у своєму SQL-запиті, особливо якщо він надто складний. Попередження та навіть автоматична перевірка наявності альтернативних, більш простих формулювань може допомогти уникнути таких ситуацій.

– Недостатні навички в SQL: користувач може не володіти достатніми навичками в SQL, тому він створює складний запит, навіть якщо існують більш прості аналоги. Оптимізація та спрощення SQL-запитів може вимагати освіти та практики.

– Віра в оптимізатор: користувач може вірити, що його складний запит буде виконуватися швидше, ніж простіший еквівалент. Однак, з огляду на декларативний характер SQL та ефективність сучасних оптимізаторів, це переконання може бути недоречним.

– Читабельність та обслуговуваність: користувач може обирати складний запит для поліпшення читабельності та обслуговуваності коду, хоча існують простіші альтернативи. Важливо виявляти такі ситуації та надавати розробникам інструменти для зрозумілості та легкості обслуговування SQL-коду.

– Недостатня увага до ефективності коду: ористувач може недооцінювати важливість ефективності свого SQL-коду, обираючи складніше формулювання. У змаганні за продуктивність важливо сприяти усвідомленості щодо можливих оптимізацій.

– Прозорість та розуміння: користувач може вибирати більш складний запит у надії забезпечити кращу прозорість та розуміння запиту для інших членів команди. Однак це може призвести до непотрібної складності та ускладнити розуміння коду. У таких випадках, коментарі та документація можуть допомогти роз'яснити логіку складних запитів.

– Вплив на підтримку та розвиток коду: вибір складного запиту може вплинути на підтримку та розвиток коду у майбутньому. Якщо розробник не залишає достатньо коментарів та пояснень, то у майбутньому інші члени команди можуть зіткнутися з труднощами підтримки та розширення такого коду.

– Сталість оптимізації: користувач може сподіватися, що його складний запит завжди буде оптимізований ефективніше, але це може не завжди відбуватися в майбутніх версіях системи управління базами даних. Завданням є надання розробникам інформації щодо того, що оптимізатори можуть змінюватися, і розроблення ефективних стратегій оптимізації коду.

Цей контекст підкреслює важливість надання розробникам інструментів та попереджень, які допомагають уникнути непродуктивного та потенційно помилкового написання SQL-запитів, особливо у випадку, коли існують еквівалентні, простіші формулювання.

Загалом, надання попереджень та інструментів для аналізу та оптимізації SQL-коду може великою мірою полегшити роботу розробників та забезпечити більш ефективну та зрозумілу роботу з базами даних. Урахування вищезазначених аспектів може допомогти створювати більш ефективні та зручні SQL-запити, сприяючи покращенню якості роботи з базами даних у реальних проектах.

Насправді «еквівалентність» у сенсі вимоги, до точно однакового результату запиту в усіх станах бази даних, зробить умову все ще надто суворою. По-перше, є необхідність мінімізувати не лише запит, але й результат запиту. Розглянемо наступний запит, досить типовий для початківців SQL-програмістів:

```
SELECT EMPNO, ENAME, JOB
FROM EMP
WHERE JOB = 'MANAGER'
```

Рисунок 1.1 – Приклад типового запиту

Останній стовпець у результатах запиту зайвий, оскільки відомо, що він завжди має бути «MANAGER». Таким чином, жодна інформація не втрачається, коли цей стовпець видаляється. Насправді добре видно, що коли доведеться вибирати між коротким запитом і коротким результатом запиту, цілком ймовірно, що перевага буде віддана короткому результату. Звичайно, спроба мінімізувати

результат запиту без втрати інформації не означає, що застосовуються алгоритми стиснення чи складне кодування. Важливою вимогою є те, що з коротшого результату запиту користувач може реконструювати оригінальний результат запиту з «дуже невеликими інтелектуальними зусиллями» — менше, ніж те, що потрібно для читання довгої версії. Це твердження трохи розпливчасте, але його можна зробити точним, перерахувавши операції, які дозволені для реконструкції вихідного результату запиту. У цьому документі користувачу потрібні лише постійні співвідношення (у випадку суперечливих умов) і прогнози.

Крім того, краще виключити певні незвичні стани, коли вимагається, щоб результат обох запитів (Q і Q(t)) був однаковим. Наприклад, іноді буває, що студенти оголошують змінну кортежу, а потім не використовують її та забувають видалити як показано на рис. 1.2.

```
SELECT DISTINCT DNAME
FROM DEPT, EMP
```

Рисунок 1.2 – Приклад лишньої змінної кортежу

«DISTINCT» також є типовим прикладом, коли неправильну інструкцію застосовано до проблеми. Наведений вище запит завжди повертає той самий результат, що й наступний на рис. 1.3 , за винятком випадків, коли EMP порожній.

```
SELECT DISTINCT DNAME
FROM DEPT
```

Рисунок 1.3 – Приклад лишньої змінної кортежу

Тому будемо вимагати еквівалентності лише для станів, у яких усі відношення не порожні. Можна навіть припустити, що всі стовпці містять принаймні два різних значення.

Деякі типи помилок створюють багато дублікатів. Можна використовувати потужніші спрощення запитів, якщо ці дублікати не вважаються важливими для еквівалентності (принаймні, якщо простіший запит Q(t) створює менше дублікатів, ніж складніший запит Q). наприклад у наведеному вище прикладі необхідно було б видалити невикористану змінну кортежу, навіть якщо DISTINCT не було вказано.

У наступних розділах буде наведено список усіх випадків, у яких запит може бути явно спрощений відповідно до цього трохи ослабленого поняття еквівалентності. У кожному з цих випадків користувач має бути попереджений.

1.4 Постановка задачі та висновок

Існує широкий клас SQL-запитів, які мають правильний синтаксис, але явно не призначені для використання, незалежно від конкретної мети запиту. Можна було б очікувати, що добре розроблені системи управління базами даних автоматично виведуть попередження для цих запитів. Однак, на превеликий жаль, наразі жодна з таких систем цього не робить. В даному контексті працюватимемо над інструментом, який буде призначений для виявлення семантичних помилок у SQL-запитах. Перераховані у цій роботі типи помилок виступають основою для специфікації цього інструменту.

Дослідження, проведені в ході аналізу предметної області, призводять до висновків про ефективність методів та покращення коду, в разі виявлення семантичних помилок. Наявні на теперішній час матеріали занадто мало виділяють уваги для оптимізації та аналізу семантичних помилок в процесі написання та виконання коду.

Об’єктом дослідження є методи виявлення семантичних помилок у коді на декларативній мові програмування SQL за допомогою статичного аналізу.

Метою дослідження є реалізація методу виявлення семантичних помилок у коді на декларативній мові програмування SQL.

Заради досягнення мети потрібно успішно виконати наступні завдання:

– визначити, список помилок які найчастіше всього зустрічаються в запитах;

– проаналізувати існуючі рішення;

– проаналізувати роботу бази даних при виявлені семантичних помилок;

– дослідити бібліотеки для роботи та парсингу SQL-запитів;

– дослідити вплив інших факторів;

– дослідити, як краще аналізувати запити без зв’язку та інформації про базу даних;

– проаналізувати, наскільки якісними є представлена методика.

2 ПОМИЛКИ В SQL-ЗАПИТАХ

2.1 Емпіричні дослідження в сфері навчання SQL

Інші подібні дослідження, представлені Мітровичем [29, 30, 31] та Байдером і Роджерсом [7], зосереджені на створенні систем навчання SQL, що базуються на зібраних даних. Автори в цих дослідженнях представляють дві дуже схожі системи, призначені для використання, як керовані навчальні середовища для SQL. Мотивація створення таких систем полягає в тому, що, хоча SQL є відносно простою і добре структурованою мовою, учні, як і раніше, стикаються з багатьма труднощами при першому її вивченні, тому такі інтелектуальні навчальні системи спрямовані на подолання цих труднощів і служать додатковим середовищем для практики вивчають SQL.

Крім того, обидві представлені системи зосереджені на виявленні семантичних помилок у SQL-запитах, оскільки синтаксичні проблеми можуть бути виявлені механізмами баз даних.

Більше того, два підходи, представлені в цих статтях, також вимагають знання про завдання, яке необхідно вирішити, як правильний запит для заданого питання SQL. Навпаки, підхід, прийнятий у цій роботі, не передбачає наявності таких знань, завдяки яким інструменти та алгоритми працюють для запитів, написаних загалом, а не лише у навчальному середовищі. Система, створена Митровичем [29], працює лише з операторами SELECT SQL, тоді як система, представлена Митровичем [31], працює з операторами SELECT та CREATE VIEW. Підхід, прийнятий у даній роботі, полягає в тому, що інструмент виявлення семантичних помилок на основі правил, повинен працювати з усіма операторами SQL. Нарешті, в обох раніше згаданих дослідженнях представлені інструменти для виявлення семантичних проблем у SQL були протестовані на досить невеликих наборах даних, які містили лише оператори SQL, отримані із завдань. Проте дослідження показало великий потенціал цих інструментів у

навчальному середовищі, що допомагає більшості учнів краще зрозуміти синтаксис SQL.

В роботі Тайпалуса та Сепьонен [43] та Тайпалус та Перьоло [42] також представлені подальші практики викладання SQL у вищих навчальних закладах, а також огляд освітніх тем досліджень SQL та найчастіших проблем, що виникають у SQL-запитах, написаних студентами. Як і у випадку з іншими попередніми дослідженнями, автори зазначають, що майбутні дослідження повинні проводитися на різноманітніших наборах даних із запитами з реальних проектів, щоб зрозуміти, чи поширюються їхні результати на програмне забезпечення в цілому, аспект, який буде з'ясовано у даній роботі.

2.3 Емпіричні дослідження SQL-запитів

Інше цікаве дослідження, присвячене проведенню першого великомасштабного кількісного аналізу синтаксичних помилок, що виникають у SQL-запитах, написаних студентами, представлене Ахаді [1]. Крім розгляду найбільш поширених помилок, які припускаються учні, у дослідженні також описується, як можна використовувати автоматичний класифікатор у прогнозування продуктивності учнів при написанні операторів SQL. Хоча стаття не має прямого відношення до аналізу семантичних помилок у SQL, автори зазначають, що у більшості випадків студенти відмовляються відповідати на питання, пов'язані з SQL через синтаксичну помилку. Їх результати показують, що синтаксичні помилки в різних інструкціях SELECT, помилки стовпця з невизначеними посиланнями та помилки групування є найбільш поширеними типами помилок, які допускають програмісти-початківці SQL при написанні своїх інструкцій.

Тому цікаво відзначити, що, хоча для вирішення семантичних помилок потрібні додаткові знання і більш глибоке розуміння принципів SQL, що лежать в основі, учні напевно не впораються з ними, поки формулювання запиту містить синтаксичні помилки, які ставлять автори дослідження.

Крім того, у подальшому дослідженні Ахаді[2], також провели перший доволі розгорнутий аналіз семантичних помилок у запитах SQL SELECT, написаних студентами. Набір даних, використаний у цій статті, містив запити, що стосуються семи різних ключових концепцій SQL: GROUP BY без HAVING, self JOIN, GROUP BY, звичайний JOIN, простий підзапит, простий запит лише з однією таблицею та, нарешті, корельовані підзапити. Знову ж таки, у дослідженні також використовувалися знання, пов'язані з питанням, для якого був написаний оператор SQL, тому запит вважався семантично невірним, коли набір результатів, що повертається їм, відрізнявся від набору, що повертається прийнятим запитом рішення.

Автори виявляють, що більшість семантичних помилок пов'язані з пропуском типу. Це свідчить про те, що учням важко вибрати правильний тип запиту, і навіть не вистачає систематичного підходу до формулювання запиту. Запити, що вимагають JOIN, підзапит або GROUP BY, найбільш схильні до цих типів помилок пропуску. Крім того, результати також показують, що більшість семантичних помилок виникає в інструкції WHERE оператора SELECT, що, як вказують автори, може бути результатом навантаження пам'яті при роботі зі складними запитами.

Інше великомасштабне дослідження помилкового коду SQL, в якому вивчається поширеність, вплив, еволюція та збіг коду SQL з традиційними помилками коду, було проведено Мюзом та іншими [34]. У своїй роботі вони проаналізували 150 проектів програмного забезпечення з відкритим вихідним кодом, у яких використовувалися популярні API-інтерфейси доступу до баз даних, такі як JDBC, JPA та Hibernate, у спробі з'ясувати, як часто у цих системах з'являються помилки коду SQL.

Хоча традиційне виявлення помилок в коді вже кілька років є важливою темою досліджень, останнім часом зростає інтерес до аналізу та виявлення такого коду SQL. Автори дослідження використовують інструмент SQLInspect і фокусуються на аналізі чотирьох помилок коду SQL. Ці помилки були вибрані з великого каталогу, опублікованого Карвіном [23], яка є першою книгою, де

представлено вичерпний список SQL помилок. Зокрема, автори дослідження розглянули неявні стовпці, де використовується оператор вибору всіх (*) замість скороченого списку, що має наслідки втрати пропускної спроможності мережі, неправильної обробки значень NULL, де використовуються оператори порівняння. Замість виразів IS NULL або IS NOT NULL використано двозначний код угруповання, де терміни інструкції GROUP BY використовуються неправильно, і, нарешті, тут проявляється випадковий відбір, коли розробники запитують один випадковий рядок, що бореться при повному скануванні таблиці.

Результати дослідження показують, що помилки коду SQL дійсно існують у системах з інтенсивним використанням даних, проте вони не залежать від інших традиційних помилок коду. Найбільш поширеним помилковим кодом було використання оператора SELECT ALL, помилка використання неявних стовпців. Також дуже цікавим відкриттям є те, що помилки коду SQL мають слабший зв'язок з помилками, ніж помилки традиційного коду, а також той факт, що помилки в SQL частіше з'являються на початку проекту і не усуваються так швидко, як інші. традиційні помилки, що також може бути пов'язано з тим, що інструменти для виявлення та інформування розробників про ці проблеми, ще не створені або не набули достатнього поширення.

У недавньому дослідженні Тайпалус [41] представляє докладний аналіз типів помилок SQL, які найчастіше зустрічаються в SQL-запитах, сформульованих початківцями. Крім того, автор також намагається пояснити причини найпоширеніших помилок. У дослідженні використовується таксономія різних синтаксичних та семантичних помилок SQL, раніше визначена Тайпалус та ін. [44] у роботі над помилками і складнощами при формулюванні SQL-запитів, і навіть набір даних запитів, зібраних із чотирьох ітерацій курсу SQL. Представлені результати показують, що трьома найпоширенішими помилками були відсутні вирази, сторонні чи опущені стовпці угруповання та відсутні оператори з'єднання.

Більше того, автори зауважили, що причина помилки тісно пов'язана з концепцією запиту, а не з типом помилки. Нарешті, у дослідженні рекомендується спосіб пом'якшення трьох помилок, що найчастіше зустрічаються в SQL-запитах

шляхом навчання студентів методам розпізнавання шаблонів мови програмування та їх еквівалентів SQL, перш ніж приступати до написання більш складних запитів.

2.4 Тестування SQL-запитів

Цікаве дослідження створення тестових даних для SQL-запитів представлено Castelein та ін. [11]. Як і у випадку з будь-яким іншим фрагментом коду, тестування має першорядне значення, щоб гарантувати, що програмне забезпечення працює правильно, і це також має стосуватися програм, які використовують SQL-запити. У своїй роботі автори моделюють проблему генерації тестових даних для SQL-запитів як завдання, що базується на пошуку, і представляють три нові підходи, засновані на пошукових алгоритмах, для вирішення цієї проблеми. Зокрема, у документі пропонується використання алгоритмів, випадкового пошуку та упередженого випадкового пошуку, а також аналізується кожен підхід у трьох проектах з відкритим вихідним кодом та однією промисловою програмною системою.

Алгоритми приймають, як вхідні дані, деякий SQL-запит для тестування, а також схему бази даних і використовують три різні методи для заповнення таблиць тестовими даними, які повинні відповідати певним критеріям тестування. Результати показують, що підхід із алгоритмом здатний покрити 98% всіх запитів у наборі даних. А також має ту перевагу, що він може правильно тестувати запити з операторами JOIN, кількома підзапитами, а також функціями маніпулювання рядками. Одним із основних вкладів у дослідження стала реалізація інструменту EvoSQL для генерації тестових даних. для SQL-запитів, а також емпіричне дослідження ефективності та продуктивності інструменту на наборі даних запитів, вилучених із чотирьох різних програмних проектів.

Крім того, дослідження показує, що повне покриття для одного запиту може бути досягнуто майже у всіх випадках протягом 2-15 секунд, що робить інструмент придатним для практичного використання. Набір даних, зібраний для

даного дослідження, буде використовуватись для тестування інструменту статичного аналізу на основі правил, які надаються в цій роботі, для подальшого аналізу поширеності семантичних помилок у запитах, які витягуються з різних програмних проектів.

У інших дослідженнях Наги і Клеве[36, 37] знайдено першу спробу реалізації інструменту виявлення семантичних помилок в коді в SQL-запитах, вбудованих у код Java. Інструмент використовує комбінацію методів статичного аналізу разом із схемами бази даних, а також даними, присутніми в базі даних, для виявлення потенційних семантичних проблем SQL-запитах, вилучених з вихідного коду.

Крім того, залежно від контексту помилки, інструмент також може визначити його серйозність. Що цікаво для цього дослідження, так це те, що інструмент надається як у вигляді інтерфейсу командного рядка, так і у вигляді модуля Eclipse, який підключається. Він повинен допомогти в забезпеченні кращої підтримки для розробників, а також прискорити впровадження інструменту, що найбільш важливо.

Варто зазначити, що в роботі Наги і Клеве [37] автори дають широку оцінку свого інструменту ряду проектів Java з відкритим вихідним кодом різної складності і розміру, з результатами, що показують потенціал. Також вони мають такі інструменти для перевірки SQL-запитів. SQL-код з помилками, реалізований авторами цих досліджень, знову взятий з книги Карвіна [23], в якій наводиться список із шести поширених антипатернів SQL.

Цікаво, що автори вирішили не реалізовувати правило виявлення для проблеми з непотрібним скануванням індексу, коли на початку рядка для вираження LIKE використовується знак підстановки, що призводить до відсутності предикату доступу Проте інструмент запропонований в даному дослідженні робить це і спроможний виявити і цю проблему. Детектор помилки коду SQL, представлений авторами, приймає як вхідні дані деякі оператори SQL, вилучені з файлу вихідного коду, і, можливо, схему бази даних разом з будь-яким вмістом бази даних, який може бути присутнім.

Після цього будуються абстрактне синтаксичне дерево (AST), так і абстрактний семантичний граф (ASG), на яких виконуються реалізовані правила для виявлення різних помилок. У майбутньому планується розширити список підтримуваних помилок коду для інструмента виявлення, а також додати додаткові функції в модуль Eclipse.

2.5 Збір даних

StackOverflow – станом на 2023 сайт з більше ніж 20 мільйонами зареєстрованих користувачів і більше ніж 10 мільйонами відвідань в день став одним з найбільш впливових веб-сайтів з питаннями та відповідями, пов’язаних з програмуванням, який існує в Інтернеті. Він був запущений в 2008 році і швидко набрав оберти і тепер містить більше 26 мільйонів питань і близько 40 мільйонів відповідей з коефіцієнтом 70% і в середньому 7 тисяч нових питань щодня ставлять на платформі кожен день. Ця величезна популярність означає, що його він являє собою велику базу знань для багатьох мов програмування і, відповідно, є цінним ресурсом для дослідників. Існує багато публікацій, та статей по збору різного типу інформації з StackOverflow, однак лише з 2020 року з’явився інтерес по вилучення інформації пов’язаної з SQL з даної платформи.

Першу спробу по вилученню інформації зв’язаної з SQL, точніше запитів зробили в 2015 році Чаба Надь та Ентоні Клів [35]. Методи, представлені у їх роботах, найбільш близькі до запропонованого в досліджені підходу, проте автори використовують дампи даних StackOverflow, що надаються Stack Exchange у форматі XML, на відміну від запропонованого підходу, який використовує дані платформи у реальному часі, використовуючи переваги API Stack Exchange. Крім того, алгоритм вилучення SQL-запитів із розділів блоку коду, який буде використано в даній роботі для побудови нашого набору даних також відрізняється від представленого в статті. Нарешті, дослідження перевіряє, чи схильні отримані запити містити помилки, шляхом ручного вивчення запитів і складання звітів за деякою загальною описовою статистикою, такою як: кількість

з'єднань на запит або використовувані функції. Як вважають автори статті, запити повинні бути додатково досліджені за допомогою автоматизованих інструментів для правильного виявлення проблем, що охоплює наше дослідження. Тим не менш, це показує величезний потенціал платформи StackOverflow для отримання великої кількості SQL-запитів з різних тем дослідження. Більше того, це має допомогти подолати поточну прогалину, яка існує в цій галузі досліджень, у тому сенсі, що більшість досліджень з виявлення семантичних проблем у SQL-запитах проводяться на наборах даних, які надходять з курсів SQL, де запити пишуться студентами в рамках домашнього завдання чи іспитів. Використання цих нових методів інтелектуального аналізу даних для вилучення запитів із StackOverflow тепер має дозволити створювати якісніші набори даних та отримувати нове уявлення про те, які типи проблем можуть виникати у запитах, виявлених у реальних додатках.

В іншому дослідженні Алламаніса та Саттона [3] використовується інформація, витягнута з питань StackOverflow, у спробі зрозуміти, які концепції програмування є найбільш заплутаними і чи є якісь подібності між різними мовами. Для цього дослідження було розглянуто низку мов програмування, серед яких Java, Python та SQL. Подивившись на слова, які найчастіше використовуються, автори змогли витягти концепції повідомлення з питаннями і створити дві категорії, одна з яких пов'язана з концепціями програмування, а інша пов'язана з типом інформації, яку шукали користувачі. Потім кожне питання було поміщене в одну із двох категорій. Більше того, питання також були згруповані з використанням тематичних моделей, і одним із основних висновків статті було те, що розподіл за типами питань не відрізняється між мовами програмування. Це означає, що типи запитань не різняться залежно від мови програмування. Цікаво, що також був представлений метод визначення того, які типи питань найбільше пов'язані з певною мовою програмування, наприклад, SQL.

Як вважають автори дослідження, цей тип інсайту може бути корисним для майбутніх IDE або інтелектуальних систем документації для надання рекомендацій розробникам залежно від контексту.

Чанг з іншими дослідниками представив дослідження уразливостей коду C/C++, виявлене у повідомленнях StackOverflow [47]. Метою статті є аналіз поширених помилок перерахування слабкостей (CWE) [27, 28] у фрагментах коду, знайдених на StackOverflow. Це особливо важливо, оскільки такий код може, наприклад, містити вразливість у системі безпеки, і його недбале використання може призвести до виникнення цих проблем в інших системах. Автори наводять, що виявляється 36% типів проблем CWE, зокрема неправильне обмеження роботи в межах буфера пам'яті. Що ще тривожніше, дослідження показує, що частка коду, що містить проблеми в StackOverflow, подвоїлася з 2008 по 2018 рік. Для створення набору даних автори спочатку беруть фрагменти коду з відповідей StackOverflow, позначені тегом C/C++. Крім того, видаляються фрагменти коду з менш ніж 5 рядками коду, щоб зменшити упередженість для частого коду. Нарешті, у документі використовується інструмент із відкритим вихідним кодом Gueslang2 для визначення того, які із вилучених фрагментів коду насправді містять код C/C++. Показано, що цей інструмент з точністю 90% визначає мову програмування зі списку 20 зумовлених мов, включаючи C/C++. Автори також проаналізували історію змін повідомлень з відповідями, щоб визначити, чи виправляються проблеми також при виявленні іншими користувачами, що не завжди так чи в деяких ситуаціях це відбувається з великою затримкою. Таким чином, у цьому дослідженні обговорюється проблема спільного використання коду, викликаного помилками, у StackOverflow, який недосвідчені розробники можуть безпосередньо використовувати у своїх додатках, не усвідомлюючи наслідків потенційного внесення помилок чи семантичних проблем до своїх систем. Попередні дослідження Ву та ін. [45] і Фішер та співавт. [19] показують, що фрагменти коду на StackOverflow дійсно поширені і використовуються розробниками, і вивчають наслідки, що випливають з цього. При цьому, Балтес та ін. [4] створили відкритий набір даних, що містить код, витягнутий зі StackOverflow, щоб надати засоби для розуміння того, як ці фрагменти підтримуються та розвиваються з часом.

Дітріх та ін [14] представляють цікаве дослідження з визначення мови програмування, що використовується у фрагментах коду StackOverflow. Автори розглядають два різних методи класифікації мови, яка використовується. Перший використовує метадані, надані користувачами, такі як теги та інші фільтри. Другий підхід використовує лінгвістичний інструмент GitHub3, який, за словами розробників, є потужною галузевою бібліотекою для автоматичної класифікації коду та визначення основної мови програмування, яка використовується. Автори дослідження зазначають, що обидва підходи виявилися досить масштабованими, щоб їх можна було використовувати в різних репозиторіях різного розміру коду. Однак, результати статті показують, що два підходи дають результати, які часто несумісні, що вказує на те, що ці інструменти слід використовувати з обережністю. Це також вказує на те, що в майбутніх дослідженнях слід вивчити гібридний підхід, який поєднує в собі сильні сторони обох методів, метадані, які надаються користувачами, а також інструменти автоматичного виявлення.

У Понцанелі та ін. [40] автори створюють повну граматику, здатну моделювати пости StackOverflow із тегами Java. Метою дослідження було побудувати гетерогенне абстрактне синтаксичне дерево (H-AST) для кожного типу повідомлень у StackOverflow, тобто питань, відповідей та коментарів, яке потім використовується для побудови структурованого набору даних для інформації про повідомлення, щоб бути використані в майбутніх дослідженнях.

Це вирішує одну з основних проблем інтелектуального аналізу даних StackOverflow, саме різнорідний характер даних, які можна знайти в повідомленнях. Використовуючи даний набір даних, можна переміщатися за вмістом повідомлення, розрізняючи код, яким у цьому випадку є Java, та фрагменти мови. Крім того, автори також надали іншу метаінформацію для кожного посту у наборі даних, таку як: вектори частоти термінів та згадані терміни. У попередньому дослідженні Понцанелі та ін. [39] представляють цікавий модуль Eclipse, що підключається, який використовує інформацію з повідомлень StackOverflow. Він також надає допомогу розробникам, що використовують активний контекст в середовищі IDE. Зокрема, модуль, що

підключається, формує запити до StackOverflow, використовуючи інформацію, доступну в активному контексті IDE, а потім надає ранжований список результатів розробнику, який потім може безпосередньо вставляти отриманий код з повідомлень.

У попередніх дослідженнях також використовувалися дані StackOverflow, щоб отримати уявлення про тенденції та найпоширеніші теми, які становлять інтерес у спільноті розробників. Баруа та ін [5] та Лінарес-Васкез та ін. [26] використовують методи моделювання LDA, щоб виявити найчастіші теми, присутні у постобговореннях. Дослідження також аналізують ці теми, а також їх еволюцію з часом і показують дедалі більшу популярність дискусій, пов'язаних з веб-розробкою та мобільними додатками. Так само Байєр і Пінчер [6] використовують ручні методи для категоризації повідомлень, пов'язаних з Android, у потоці StackOver і знаходять залежності між типами питань та певними проблемами, забезпечуючи краще розуміння найчастіших проблем, що виникають при розробці мобільних додатків. Це ще раз показує великий потенціал використання величезної бази знань, якою став StackOverflow, у видобуванні цікавої інформації для різних досліджень.

Хоча цей аналіз не наводиться у нашому дослідженні, було б дуже цікаво використовувати запропонований зібраний набір даних і виконати його аналіз LDA, щоб визначити основні теми, які становлять інтерес для питань, пов'язаних із SQL, представлених у повідомленнях StackOverflow. Крім того, було б дуже корисно виконати ручний аналіз деяких повідомлень, для яких запропонований інструмент повідомляє про семантичні проблеми, виявлені у вилучених запитах, щоб зрозуміти, чи розробники помічають наявність цих типів проблем SQL, а також якщо помилки усуваються з часом. Нарешті, Хатун та співавт. [24] надають повний огляд та оцінку сучасних інструментів та методів вилучення вихідного коду, в основному зосереджених на таких джерелах, як веб-сайти відстеження коду або інші API, проте доречні в поданні різних методів вилучення вихідного коду. Інші дослідження з видобутку вихідного коду різних API також представлені Лямбда та ін. [25], Хьйо та Пей [46] та Кагді [22].

2.6 Висновок

В даному розділі було проведено аналіз і порівняльний огляд різних підходів до викладання мови SQL в навчальних середовищах. Дослідження, проведені Мітровичем, Байдером, Роджерсом, Тайпалусом і Сепьонен, розглядали створення інтелектуальних навчальних систем, спрямованих на подолання труднощів, з якими стикаються студенти під час вивчення SQL. Порівняно із попередніми дослідженнями, які обмежувалися деякими конкретними аспектами SQL, ця робота виходить за межі і розглядає всі аспекти мови SQL.

Важливо відзначити, що основним відмінностями представленого дослідження є те, що воно використовує інструмент виявлення семантичних помилок на основі правил, який спроможний працювати з усіма можливими операторами SQL. Це розширює можливості навчання і дозволяє краще розуміти не лише синтаксис SQL, але й його семантику.

Додатково, дослідження Тайпалуса, Сепьонен та Перьоло вказує на необхідність подальших вивчень з використання реальних проектів із запитами SQL. Це може допомогти з'ясувати, наскільки результати вищезгаданих досліджень можуть бути застосовані до реальних програмних забезпечень та реальних завдань, що виникають у практичних проектах.

У ряді досліджень, проведено великомасштабний аналіз синтаксичних та семантичних помилок у SQL-запитах, які формулюються студентами та іншими початківцями. Аналіз синтаксичних помилок, представлений у роботі Ахаді, показує, що студенти найчастіше допускають помилки в операторах SELECT, такі як помилки в області WHERE, використання невизначених посилань і пропуск стовпців. Ці помилки є загальними та можуть бути важко виявлені через відсутність систематичного підходу до формулювання запитів.

Семантичний аналіз, представлений у дослідженні Ахаді та Тайпалуса, розкриває, що багато семантичних помилок пов'язані з вибором неправильного типу запиту та відсутністю систематичності в формулюванні запитів. Це свідчить про те, що студенти мають недостатнє розуміння основних принципів SQL. Ці

результати вказують на важливість навчання студентів систематичному підходу до формулювання запитів та надання їм знань про правильний вибір типу запиту у конкретних ситуаціях.

Дослідження помилок коду SQL, яке вивчало різноманітні аспекти цих помилок у відкритих проектах з використанням популярних API-інтерфейсів, вказує на те, що ці помилки виникають навіть у реальних програмних проектах, що підкреслює важливість усунення їх на ранніх етапах розробки. Результати цього дослідження показують, що помилки, такі як використання оператора SELECT ALL та неправильна обробка значень NULL, є серйозними проблемами в SQL-коді, які можуть впливати на продуктивність і надійність програмного забезпечення.

Загалом, ці дослідження свідчать про важливість систематичного підходу до навчання SQL, включаючи навчання синтаксичних та семантичних аспектів, а також усунення помилок в коді SQL на ранніх етапах розробки. Навчання студентів правильному формулюванню запитів та усунення їхніх помилок може значно покращити якість SQL-коду, що пишуть студенти та розробники.

3. СТРАТЕГІЯ ВИЯВЛЕННЯ СЕМАНТИЧНИХ ПОМИЛОК SQL

3.1 Семантичні помилки в запитах

У цьому розділі запропоновано кілька евристик, які використовуються для виявлення семантичних помилок у SQL-запитах. Кожну з яких, пізніше буде реалізовано у вигляді правил у інструменті статичного аналізу, який розроблявся для цього проекту. Помилки, які виявляються за допомогою запропонованої в даному дослідженні евристики, були спочатку представлені та класифіковані в роботі Брасса та Голдберга [1], яка є повним списком семантичних помилок, що зустрічаються в SQL. Невелика підмножина семантичних помилок була зібрана за допомогою інструмента SQL Enlight. Це інструмент статичного аналізу із закритим вихідним кодом, розроблений Yubitsoft Eood, який також спрямований на виявлення низки проблем у SQL-запитах. Іншими інструментами для аналізу SQL і виявлення помилок коду є TOAD і SQL Prompt, обидва з яких мають закритий вихідний код і вимагають набору вхідних запитів, а також схем бази даних для виявлення будь-яких проблем. Крім того, жоден із раніше згаданих інструментів не може аналізувати запити, вбудовані у вихідний код.

Інструмент статичного аналізу на основі правил, розроблений для цього проекту, використовує жорстку реалізацію, а це означає, що завжди передбачається найгірше, оскільки немає додаткової інформації, пов'язаної із завданнями чи схемами бази даних. Це означає, що для певних запитів інструмент може генерувати неправдиві спрацьовування попереджень, як буде обговорюватися пізніше, однак це було кращим, оскільки він пропонує гарний компроміс для налаштування, коли відомо мало інформації про запити.

У наступних підрозділах надано опис кожної помилки, приклади SQL-запитів, що містять семантичну помилку, та представлено реалізацію кожної стратегії виявлення разом з будь-якими припущеннями, зробленими евристиками, які використовуються у запропонованому інструменті. Зведення всіх семантичних помилок, виявлених нашим інструментом, також наведено в таблиці 3.1.

Таблиця 3.1 – Список помилок

Семантична помилка	**Опис**
Несумісна змінна кортежа	Кортежі, зіставлені за ключовими атрибутами, отримують різні значення деяких інших атрибутів.
Постійний вихідний стовбець	Значення вихідного стовпця є постійним і може бути отримано із запиту.
Дублювання вихідного стовпця	Той самий атрибут присутній в SELECT декілька раз
Непотрібна інструкція JOIN	Кортеж можна замінити іншим вже існуючим (видалення непотрібного JOIN)
Ідентичні змінна кортежа	Кортежі, зіставлені за ключовими атрибутами, ідентичні.
Порівняння з NULL	Використання звичайних операторів порівняння замість виразу IS NULL
Відсутня інструкція JOIN	Відсутня посилання для об’єднання таблиць
Некорельовані підзапити EXISTS	Підзапити не містять посилань на таблиці або кортежі, визначені у частинах запиту вищого рівня.
Ділення на нуль	Можливе ділення на нуль через порядок виконання операції
Дивна інструкція HAVING	Запит з використанням інструкції HAVING без інструкції GROUP BY
Дивні символи підстановки без LIKE	Використання підстановочних знаків без LIKE
Неспівпадіння в підзапиті SELECT	Для підзапитів IN список SELECT із внутрішнього запиту повинен відповідати атрибутам, що використовуються у зовнішньому запиті
Дивні умови підзапита	Умова всередині підзапиту SQL, що звертається тільки до змінних кортежу із зовнішнього запиту, є дивним
Неефективна інструкція HAVING	Умова всередині інструкції HAVING, яку можна перемістити всередину інструкції WHERE, що робить

Продовження таблиці 3.1

	запит ефективнішим
Непотрібна умова ORDER BY	Постійний термін не обов'язково включати в інструкцію ORDER BY
Непотрібна інструкція GROUP BY	Постійний термін не обов'язково включати в інструкцію GROUP BY
Непотрібна умова UNION	Дві інструкції UNION з однаковими умовами SELECT та FROM та взаємовиключненнями де умови можуть бути об'єднані
Непотрібна фраза GROUP BY	Якщо функції агрегації не використовуються, GROUP BY можна замінити на SELECT DISTINCT
Непотрібне загальне порівняння	Використання операторів «більше чи рівно» або «менше чи рівно» замість оператора «рівно»
Використання LIKE без символів підстановки	LIKE можна замінити оператором EQUALS
Непотрібний список SELECT в EXISTS	Немає необхідності додавати складний список SELECT всередині підзапитів EXISTS
Непотрібне сканування індексу	Використання знаку підстановки на початку рядка призводить до відсутності предикату доступу
Непотрібний DISTINCT в агрегаціях	Для деяких функцій агрегації ключове слово DISTINCT не впливає на результат.
Непотрібний атрибут GROUP BY	Непотрібний атрибут у групі GROUP BY, який не відображається в групі SELECT або HAVING поза агрегаціями
Непотрібний аргумент COUNT	У деяких випадках аргумент COUNT не є обов'язковим і може бути замінений чимось простим

3.2 Стратегії виявлення семантичних помилок

Виявлення семантичних помилок в SQL-запитах може бути непростою задачею через їх складність і можливість виникнення помилок на різних етапах виконання запитів. Однак існують різні стратегії, які можна використовувати для виявлення семантичних помилок в SQL:

1. Перевірка синтаксису. Перш за все, необхідно переконатися, що SQL-запит синтаксично вірний. Це можна зробити за допомогою SQL-аналізаторів або інструментів для редагування SQL-коду, які підкреслюють синтаксичні помилки.

2. Перевірка назв таблиць та колонок. Переконатись, що всі назви таблиць і колонок вказані вірно. Використання імен таблиць та колонок, які існують у базі даних.

3. Перевірка правильності запиту. Переконатись у тому, що типи даних, умови і операції вказані правильно. Наприклад, чи порівнюються правильні типи даних, чи вірно використовуються оператори порівняння тощо.

4. Перевірка зв'язків між таблицями. Якщо є SQL-запит з JOIN операціями, потрібно перевірити, що зв'язки між таблицями вказані правильно. Переконатись у тому, що вказано вірні ключі для з'єднання таблиць.

5. Перевірка унікальності та обмежень. Потрібно переконатись у тому, що унікальність та інші обмеження бази даних враховані в SQL-запиті. Наприклад, переконайтеся, що не спробують вставити запис із значенням ключового поля, яке вже існує.

6. Використання тестових даних для виконання SQL-запитів. Тестові дані можуть допомогти виявити неправильні результати або семантичні помилки у запитах.

7. Використання логічних перевірок. Використання логічних перевірок, щоб переконатися у тому, що результати запитів відповідають очікуваним, та порівняння отриманих даних з очікуваними результатами.

8. Запобігання SQL-ін'єкціям. Потрібно звертати увагу на вразливості SQL-ін'єкцій. Використовувати параметризовані запити або підготовлені інструкції, щоб запобігти SQL-ін'єкціям та іншим видам атак на базу даних.

9. Використання SQL-інструментів для від лагодження. Використання інструменту для відлагодження SQL-запитів, які дозволяють виконувати запити по кроку, спостерігати за змінними, що змінюються в процесі виконання запиту, та відстежувати логіку виконання.

10. Аудит запитів. Ведення журналу запитів та помилок для виявлення семантичних або логічних помилок у запитах.

Ці стратегії можуть бути корисними для виявлення семантичних помилок в SQL-запитах і покращення якості ваших баз даних та додатків.

3.2.1 Несумісна змінна кортежа

Проблема: ця помилка присутня щоразу, коли ключові атрибути двох різних змінних кортежу в одному відношенні зіставляються, а в той же час для якогось іншого атрибута дві змінні кортежу співставляються з різними значеннями. Оскільки наш інструмент не потребує інформації про схему бази даних, ми не можемо визначити ключові атрибути, тому умова для цієї семантичної помилки пом’якшується шляхом усунення цього обмеження. Точніше, щоразу, коли дві різні кортежні змінні в одному відношенні мають набір атрибутів, за якими вони зіставляються, а також принаймні один атрибут, для якого існують різні вимоги, тоді це означає, що запит непослідовний, оскільки умови не можуть бути задоволені одночасно, отже, набір результатів буде порожнім. У наступному прикладі на рис. 3.1 слід виявити семантичну помилку.

```
SELECT *
FROM table t1
WHERE NOT EXIST
(
      SELECT *
      FROM table t2
      WHERE t2.field = 'value1' AND t1.field = 'value2' AND t2.id = t1.id
)
```

Рисунок 3.1 – Несумісна змінна кортежа

У цьому прикладі дві змінні кортежі t1 і t2 в одному і тому ж відношенні таблиці містяться всередині підзапиту EXISTS за атрибутом id , однак запит не погоджений, оскільки t1.field = ‘value1’ і t2.field = ‘value2’ не може мати два різних значення одночасно, тому набір результатів для цього запиту завжди буде пустим. Не знаючи намірів розробника, ми не можемо запропонувати рефакторингову версію запиту, однак еквівалентний запит виглядатиме так як зображено на рисунку 3.2.

```
SELECT * FROM table t1
```

Рисунок 3.2 – Відрефакториний код

Стратегія виявлення: знаходження цієї семантичної помилки здійснюється шляхом побудови для кожної використаної змінної кортежу набору атрибутів, які прирівнюються до інших змінних кортежу через те саме відношення. Крім того, створюється другий набір, який зберігає атрибути, які прирівнюються до інших постійних значень. Після аналізу запиту для всіх змінних кортежу в одному відношенні треба перевірити набір відповідних атрибутів, а також набір атрибутів, які прирівнюються до різних значень. Якщо існує повна відповідність між атрибутами в першому наборі, тоді перевіряється, чи є спільні атрибути з другого набору, які можуть бути прирівняні до різних значень. Якщо це так, тоді запит містить неузгоджені умови, і наш інструмент генерує попередження про цю семантичну помилку.

3.2.2 Постійний вихідний стовбець

Проблема: ця семантична помилка з'являється кожного разу, коли значення вихідного стовпця є постійним і може бути отримано із запиту без будь-якої додаткової інформації щодо стану бази даних. Зокрема, кожного разу, коли атрибут у інструкції WHERE прирівнюється до деякого постійного значення, якщо він також присутній у списку SELECT , то він завжди матиме те саме значення. Тому

цей атрибут можна видалити із інструкції SELECT. У наступному прикладі на рис. 3.3 має бути виявлена семантична помилка.

```
SELECT * FROM tabl1 WHERE field = ‘value’;
```

Рисунок 3.3 – Постійний вихідний стовбець

У цьому прикладі є постійний стовпець, який є у виведенні запиту, точніше, атрибут імені прирівнюється до деякого постійного значення, тому його можна видалити з інструкції SELECT . Ми не можемо безпосередньо надати потенційне виправлення для цього запиту, не маючи додаткової інформації про структуру схеми бази даних, проте запит можна легко виправити, вказавши лише необхідні окремі стовпці.

Стратегія виявлення: ця помилка виявляється шляхом пошуку атрибутів у інструкції WHERE, які прирівнюються до деякого постійного значення. Якщо цей атрибут також присутній у інструкції SELECT або використовується оператор вибору всіх (*), то це означає, що він завжди матиме постійне значення, тому його можна видалити з виводу, і буде згенеровано попередження, що вказує на це, нашим інструментом.

3.2.3 Дублювання вихідного стовпця

Проблема: ця помилка з'являється, коли список SELECT запит або підзапит містить один і той же атрибут кілька разів. В даному випадку це означає, що в результуючому наборі будуть присутні вихідні стовпці, що повторюються, тому ці стовпці можна видалити з інструкції SELECT, щоб спростити розуміння запиту. У наступному прикладі на рис. 3.4 має бути виявлена семантична помилка.

```
SELECT t1.field1 , t1.field2 , t2.field3 AS text, t2.field3 AS text2
FROM tabble1 AS t1
INNER JOIN table2 as t2;
```

Рисунок 3.4 – Помилка дублювання вихідного стовпця

У цьому прикладі є вихідний стовпець, який повторюється, оскільки атрибут t2.field3 вказаний двічі в інструкції SELECT запиту. Можливе виправлення для прикладу запиту наведено нижче на рис. 3.5.

```
SELECT t1.field1 , t1.field2 , t2.field3 AS text1_or_text2
FROM tabble1 AS t1
INNER JOIN table2 as t2;
```

Рисунок 3.5 – Виправлення помилка дублювання вихідного стовпця

3.2.4 Непотрібна інструкція JOIN

Проблема: ця помилка з'являється в запитах, що містять кортежну змінну A, для якої звертаються тільки до атрибутів ключа, а потім цей ключ прирівнюється до зовнішнього ключа іншої кортежної змінної B. У цьому випадку кортежна змінна A не потрібна, оскільки це призводить до непотрібного використання JOIN. Таким чином, ця семантична помилка може вплинути на продуктивність запиту та має бути виправлена. Для нашого інструменту відпадає вимога до ключових атрибутів, оскільки вхідні дані містять тільки представлений запит, і немає способу визначити, які атрибути представляють ключі для змінної кортежу. Тому ми перевіряємо, чи не прирівнюються для деякої змінної кортежу A атрибути, до яких здійснюється доступ для неї, з тими ж атрибутами іншої змінної кортежу B, і в цьому випадку A може не знадобитися, що призводить до непотрібного JOIN. У наступному прикладі на рис. 3.6 має бути виявлена семантична помилка.

```
SELECT t.*
FROM terms t JOIN term_relationships tr ON tr.term_id = t.term_id
JOIN posts p ON p.post_id = tr.post_id WHERE p.post_id = 1;
```

Рисунок 3.6 – Запит з непотрібна інструкцією JOIN

У цьому прикладі є непотрібне JOIN для повідомлень з post_id , оскільки умова всередині інструкції WHERE, p.post_id = 1 можна замінити на tr.post_id = 1 і перемістити вгору в другому реченні JOIN. Це не тільки допомагає спростити розуміння запиту, а й підвищує продуктивність запиту, оскільки необхідно виконати одне з'єднання менше. Можливе виправлення для прикладу запиту наведено нижче на рис. 3.7 .

```
SELECT t.*
FROM terms t JOIN term_relationships tr ON tr.term_id = t.term_id
WHERE tr.post_id = 1;
```

Рисунок 3.7 – Запит без зайвого JOIN

Стратегія виявлення. Виявлення цієї семантичної помилки здійснюється шляхом відстеження для кожної змінної кортежу порівнянь, в яких вони беруть участь протягом усього запиту. Крім того, якщо є оператор вибору всіх (*), то також немає необхідності вибирати інші атрибути. Після аналізу запиту кожної змінної кортежу A перевіряються всі вирази порівняння, у яких вона використовувалася. Якщо всі атрибути, до яких здійснюється доступ, також прирівнюються до іншої змінної кортежу B, у припущенні, що ці атрибути є зовнішнім ключем цієї змінної кортежу B, видається попередження про непотрібну інструкцію JOIN .

3.2.5 Ідентичні змінна кортежа

Проблема: якщо запит містить дві різні змінні кортежу над тим самим відношенням, для яких прирівнюються ключові атрибути, то ці дві змінні завжди повинні вказувати на той самий кортеж. Оскільки запропонований інструмент не приймає на вхід ніякої інформації про ключові атрибути схеми, для якої був написаний запит, правило виявлення перевіряє змінні кортежу за тим самим відношенням, для якого прирівнюється хоча б один атрибут, припущення, що цей атрибут представляє ключ. У деяких випадках це може не виконуватися і

згенероване попередження слід розглядати як помилкове спрацьовування. У наступному прикладі на рис. 3.8 має бути виявлена семантична помилка.

```
SELECT *
FROM employees X, employees Y
WHERE X.employee_id=Y.employee_id;
```

Рисунок 3.8 – Запит з помилкою дублюванням змінної кортежа

У цьому прикладі дві змінні кортежу X і Y для одного і того ж відношення employee зіставляються по employee_id, тому в припущенні, що це унікальний ключ, дві змінні кортежу ідентичні, тобто вони завжди будуть посилатися на ті самі рядки. У цьому випадку не можлво безпосередньо надати потенційне виправлення для цього запиту, не маючи додаткової інформації про структуру схеми співробітників та основне завдання, для якого було написано запит.

Стратегія виявлення. Реалізація для виявлення цієї семантичної помилки спочатку зберігає всі змінні кортежу, які використовуються у запиті, а також таблиці, на які посилаються змінні. Щоразу, коли виявляється оператор рівності, буде перевірятися, чи вказують змінні кортежу, що використовуються при порівнянні, на ту саму базову таблицю, і той самий атрибут використовується на обох сторонах порівняння. Якщо це так, у припущенні, що порівнюваний атрибут являє собою унікальний ключ у таблиці, дві змінні кортежу завжди повинні вказувати на той самий кортеж, тому вони ідентичні, і генерується попередження, що вказує на наявність цієї семантичної помилки.

3.2.6 Порівняння з NULL

Проблема: у деяких системах управління базами даних синтаксично допустимо використовувати A = NULL, проте цей вираз завжди має постійне істинне значення або NULL, або невідоме. Щоб уникнути таких ситуацій, натомість завжди слід використовувати IS NULL або IS NOT NULL. У наступному прикладі на рис. 3.9 має бути виявлена семантична помилка.

```
SELECR r.username FROM root r r.id, r authtoken.instagram, WHERE r.abc <> NULL
```

Рисунок 3.9 – Помилка порівняння стовпця з NULL

У цьому вся прикладі r.abc порівнюється з нулем. Для деяких ядер баз даних це може не вважатися синтаксичною помилкою, проте умова повертає значення NULL або UNKNOWN. Щоб уникнути таких ситуацій, коли використовується NULL, завжди слід використовувати вирази IS NULL або IS NOT NULL замість операторів порівняння. Можливе виправлення для прикладу запиту наведено нижче на рис. 3.10.

```
SELECT r.id, r.authtoken.instagram , r.username
FROM root r
WHERE r.abc IS NOT null;
```

Рисунок 3.10 – Запит з вірним порівнням з NULL

Стратегія виявлення. Для виявлення цієї семантичної помилки щоразу, коли всередині запиту використовуються оператори «рівно» чи «не рівно», перевіряється, чи є одне з умов ключовим словом NULL . Якщо це так, оператор порівняння слід замінити виразом IS NULL або IS NOT NULL відповідно.

3.2.7 Відсутня інструкція JOIN

Проблема: ця помилка з'являється у запитах, які включають з'єднані таблиці, котрим у джерелах об'єднаних таблиць немає жодного стовпця, який є посиланням ні за умови JOIN , ні з інструкції WHERE. Якщо предикат JOIN відсутній, запит включатиме Декартовий добуток всіх рядків, також відомий, як перехресний добуток, напевно, призводить до зниження продуктивності запитів, а також до потенційно неправильних результатів. Тому важливо, щоб на об'єднані таблиці посилалися на інструкції JOIN ON або WHERE, щоб уникнути подібних проблем.

Однак є один виняток, зокрема коли використовується CROSS JOIN. У цих випадках немає необхідності, щоб задіяні джерела таблиць мали посилання на будь-які стовпці, оскільки для цих запитів явно намір полягає в тому, щоб отримати векторний добуток всіх рядків, тому попередження не повинно генеруватися. У наступному прикладі на рис 3.11 має бути виявлена семантична помилка.

```
SELECT station , slot, subslot , compid , compname
FROM devicetrace AS dt
INNER JOIN complist as cl;
```

Рисунок 3.11 – Запит без інструкції JOIN

У цьому прикладі є дві з'єднані таблиці, devicetrace і complist , у яких немає посилань на стовпці ні умови JOIN, ні WHERE. Це означає, що результат запиту включатиме перехресний добуток усіх рядків. Незалежно від того, чи це було наміром автора запиту чи ні, відсутні умови з'єднання є потенційною причиною зниження продуктивності і повинні сигналізуватися як семантичні помилки. Не маючи додаткової інформації про базове завдання, на яку було написано запит, можна надати потенційне виправлення цього запиту.

Стратегія виявлення. Виявлення цієї семантичної помилки здійснюється шляхом аналізу запиту та відстеження всіх використовуваних джерел таблиць. Якщо використовується менше двох таблиць або задіяно CROSS JOIN, немає необхідності продовжувати перевірку відсутніх умов з'єднання, тому жодних попереджень не буде. Якщо використовуються як мінімум два джерела таблиць, то перевіряються інструкції ON і WHERE задіяного підзапиту, щоб визначити, чи є в цих джерелах таблиць якісь стовпці, на які є посилання. Для будь-якої таблиці, на яку немає посилання, буде згенеровано попередження про відсутність умови з'єднання.

3.2.8 Некорельовані підзапити EXISTS

Проблема: некорельовані підзапити – це ті, які не містять жодних посилань на таблиці або змінні кортежу, визначені у частинах запиту вищого рівня. Якщо підзапит EXISTS або NOT EXISTS не містить жодних посилань на змінні кортежу із зовнішніх запитів, це означає, що підзапит некорельований, що робить його глобально істинним або глобально помилковим. Це незвичайна поведінка, оскільки обидві інструкції EXISTS та NOT EXISTS слід використовувати з корельованими запитами. При використанні в корельованих підзапитах вираз оцінюється один раз для кожного рядка батьківського запиту, і в результаті повертається значення TRUE або FALSE. Таким чином, при правильному використанні підзапитів EXISTS в корельованих підзапитах оптимізатор SQL може скористатися перевагою оцінки підзапиту без матеріалізації проміжних результатів, а також кешування результатів повторного використання раніше обчислених значень предикату для тих же значень посилання на кортеж зовнішнього запиту. Тому щоразу, коли підзапит EXISTS не посилається на будь-яку змінну кортежу із зовнішнього запиту, це є семантичною помилкою і має бути видане попередження. У наступному прикладі на рис 3.12 має бути виявлена семантична помилка.

```
SELECT *
FROM final_combined_result wfcr
WHERE NOT EXISTS (SELECT contact_id , account_id FROM temp_wfcr);
```

Рисунок 3.12 – Помилка некорельованого підзапиту EXISTS

У цьому прикладі підзапит NOT EXISTS некорельований, так як усередині підзапиту немає посилань на зовнішні змінні кортежу. Це означає, що підзапит буде або глобально істинним, або глобально помилковим, повертаючи той самий набір результатів кожного разу, коли він виконується для рядків у батьківському запиті. Крім того, цей приклад також містить іншу семантичну помилку, зокрема список SELECT усередині підзапитів EXISTS не важливий, тому його можна спростити, просто використовуючи оператор вибору всіх записів (*).

У цьому прикладі підзапит NOT EXISTS некорельований, так як усередині підзапиту немає посилань на зовнішні змінні кортежу. Це означає, що підзапит буде або глобально істинним, або глобально помилковим, повертаючи той самий набір результатів кожного разу, коли він виконується для рядків у батьківському запиті. Крім того, цей приклад також містить іншу семантичну помилку, зокрема список SELECT усередині підзапитів EXISTS не важливий, тому його можна спростити, просто використовуючи оператор вибору всіх записів (*).

Стратегія виявлення. Для виявлення цього семантичної помилки запит аналізується в пошуках підзапитів EXISTS. Список раніше виявлених таблиць і змінних кортежу зберігається, і щоразу, коли зустрічається умова всередині підзапиту EXISTS, буде перевірятися чи робиться якесь посилання зовнішні таблиці чи змінні кортежа. Якщо виявлено хоча б одне таке посилання, то підзапит EXISTS вважається корельованим і йому не видаються попередження. Щоразу, коли підзапит EXISTS не корельований, інструмент видає попередження про цю семантичну помилку.

3.2.9 Ділення на нуль

Проблема: одна з найпоширеніших проблем з операторами типів даних – ділення на нуль. Оскільки в SQL немає гарантії послідовності обчислень операторів у інструкції WHERE, розробникам важко уникнути таких ситуацій. Наступний приклад на рис. 3.13 вважається небезпечним і має бути виявлений як семантична помилка.

```
SELECT itm_num
FROM itemconfig
WHERE (pal_qty / case_qty) > 500 AND case_qty > 0;
```

Рисунок 3.13 – Помилка з потенційним діленням на нуль

У цьому прикладі, незважаючи на те, що у інструкції WHERE є умова case_qty > 0 , оскільки порядок операцій не застосовується, цей запит, як і раніше,

небезпечний. Тому попередження має бути викликане, коли у запиті є ділення, за винятком ситуацій, в яких ділення знаходиться в реченні SELECT, а стовпець дільника перевіряється на не рівність нулю в реченні WHERE, оскільки в цих випадках WHERE інструкція оцінюватиметься до SELECT будь-яким механізмом бази даних.

Стратегія виявлення. Реалізація цього правила аналізує запит і зберігає всі операції ділення, що зустрічаються. Крім цього, також створюється логічний прапор, який вказує, чи знаходилось операція ділення у реченні SELECT чи ні. Крім того, всі стовпці, які не рівні нулю в інструкції WHERE, також зберігаються. Нарешті, підзапити, які перебувають у інструкції SELECT, але для яких стовпець дільника перевіряється у інструкції WHERE, видаляються з вихідного набору, а члени ділення, що залишилися, повертаються в результаті для цього правила, яке генерує попередження для потенційних поділів на нуль.

3.2.10 Інструкція HAVING

Проблема: запит з використанням інструкції HAVING без інструкції GROUP BY вважається дивним, оскільки він може мати лише один результат або не мати його взагалі. Інструкція HAVING схожа на інструкцію WHERE, проте вона застосовується лише до груп загалом, а не до окремих рядків. У стандартному SQL спочатку виконується інструкція FROM, потім інструкція WHERE, потім GROUP BY і, нарешті, інструкції HAVING та SELECT. Це означає, що якщо інструкція GROUP BY відсутня, всі рядки, які не виключені інструкцією WHERE , будуть повернені як єдина група на інструкцію HAVING. У цьому випадку, оскільки між інструкціями WHERE і HAVING не виконується групування, останнє діятиме як інструкція WHERE, за винятком того, що вона працює з результуючим набором як із групою, а також дозволені функції агрегування. Крім того, за наявності з'єднань інструкція HAVING без GROUP BY має генерувати синтаксичну помилку у більшості механізмів баз даних. Таким чином, оскільки HAVING фільтрує групи, за відсутності інструкції GROUP BY всі рядки представляють одну групу, і в цьому випадку, якщо предикат усередині інструкції

HAVING має значення TRUE, запит поверне один рядок, інакше нічого не буде повернено. Таким чином, у цих ситуаціях запит з використанням інструкції HAVING без інструкції GROUP BY є дивним і повинен розглядатися як такий, що містить семантичну помилку. У наступному прикладі на рис 3.14 має бути виявлена семантична помилка.

```
SELECT height
FROM sashelp.class
HAVING height = max(height);
```

Рисунок 3.14 – Помилка з дивною інструкцією HAVING

У деяких механізмах баз даних цей SQL-запит може навіть розглядатися, як такий, що має синтаксичну помилку, оскільки в інструкції HAVING є посилання на стовбець висоти, але вона не міститься в агрегатній функції або інструкції GROUP BY. У цьому випадку запропонований інструмент виявить семантичну помилку, оскільки інструкція HAVING використовується без GROUP BY. Можливе виправлення для прикладу запиту наведено нижче на рис. 3.15.

```
SELECT height
FROM sashelp.class
GROUP BY height
HAVING height = max(height);
```

Рисунок 3.15 – Використання HAVING з GROUP BY

Стратегія виявлення. Реалізація цього правила буде аналізувати запит у пошуках інструкцій GROUP BY та HAVING. Два логічні прапори використовуються для відстеження двох інструкцій у запиті. Після завершення синтаксичного аналізу або всього запиту або підзапиту перевіряються два прапори. Якщо інструкція HAVING є без використання інструкції GROUP BY, то інструмент позначає це, як семантичну помилку.

3.2.11 Спеціальні символи підстановки без LIKE

Проблема: в SQL зарезервовано кілька спеціальних символів для використання як знаки підстановки. Точніше, ці символи мають особливе значення та використовуються для заміни одного або кількох символів у рядку. Підстановочні знаки використовуються у поєднанні з оператором LIKE, дозволеним у інструкції WHERE запиту, для пошуку вказаного шаблону записах стовпця. Оскільки ці спеціальні символи записуються всередині рядка, використання одного з цих символів у виразі порівняння без оператора LIKE не призведе до синтаксичної помилки, оскільки механізм бази даних не може дізнатися, чи автор запиту мав намір використовувати оператор підстановки. або його базовий символ усередині рядка. Таким чином, щоразу, коли підстановочний знак використовується без оператора LIKE, має видаватися попередження. У наступному прикладі на рис. 3.16 має бути виявлена семантична помилка.

```
SELECT *
FROM employees
WHERE name = 'chris%';
```

Рисунок 3.16 – Невірне використання символів підстановки

У цьому прикладі механізм бази даних буде розглядати знак підстановки % як звичайний літерал. Це, безумовно, не те, що насправді мав на увазі розробник при написанні цього запиту, оскільки результат, який повертається, ймовірно, буде порожнім набором, оскільки немає реальних імен, що закінчуються на %. Можливе виправлення для прикладу запиту наведено нижче на рис. 3.17.

```
SELECT *
FROM employees
WHERE name LIKE  'chris%';
```

Рисунок 3.17 – Вірне використання символів підстановки

Стратегія виявлення. Реалізація цього правила аналізує запит у пошуках виразів порівняння, що включають стовпці та рядкові значення, що містять один або кілька знаків підстановки. Щоразу, коли зустрічаються ці типи виразів, спрацьовує попередження про знаки підстановки, що використовуються без оператора LIKE . Підстановочний знак підкреслення (_) не розглядається для цієї перевірки, так як ці типи символів досить часто з'являються у значеннях стовпця, тому в цих ситуаціях неможливо визначити, чи відсутній оператор LIKE є семантичною помилкою.

3.2.12 Розбіжності в підзапиті SELECT

Проблема: при використанні оператора IN разом з виразом підзапиту список SELECT з внутрішнього запиту повинен відповідати атрибутам, що використовуються у зовнішньому запиті. використовуються у батьківському запиті. Також вірно, що можуть бути випадки, коли такі невідповідності можуть фактично не представляти проблеми із запитом, проте будемо вважати, що розробники SQL все одно повинні бути проінформовані про ці невідповідності, щоб мати можливість подальшого аналізу, чи містить запит будь-яку проблему чи ні. Тому, коли інструкція SELECT із підзапиту містить атрибути, відмінні від атрибутів, які використовуються у зовнішньому запиті, має бути видане попередження. У наступному прикладі на рис. 3.18 має бути виявлена семантична помилка.

```
SELECT t1.articleno , t1.artdescription , t2.dateyear
FROM t1 JOIN t2
WHERE NOT t1.articleno IN (SELECT t2.dateyear FROM t2);
```

Рисунок 3.18 – Помилка з розбіжностями в підзапиті SELECT

У цьому прикладі є невідповідність між t1.articleno в батьківському запиті і t2.dateyear у інструкції SELECT внутрішнього запиту. Хоча це не завжди може

означати основну проблему із запитом, запропонований інструмент видасть попередження у таких ситуаціях. Знову ж таки, не маючи інформації про завдання, для якого було написано запит, або інформації про схему бази даних, неможливо точно виявити ці типи проблем, а також неможливо надати виправлення для цих запитів, однак є змога припустити щоразу, коли є невідповідність між цими атрибутами потрібна особлива увага, тому розробники повинні бути проінформовані.

Стратегія виявлення. Реалізація цього правила переглядає лише вирази IN із запитів та порівнює список атрибутів, що використовуються у зовнішньому батьківському запиті, зі списком, який використовується інструкцією SELECT у внутрішньому підзапиті. Щоразу, коли виникає невідповідність між іменами стовпців, які у цих двох реченнях, інструмент відстежує це, і після аналізу всього запиту всіх виявлених невідповідностей відображається попередження. Щоб уникнути великої кількості потенційно неправдивих позитивних попереджень, реалізація також перевіряє збіги підрядків, і в цьому випадку попередження не буде згенероване.

3.2.13 Неефективна інструкція HAVING

Проблема: якщо умова всередині інструкції HAVING SQL-запиту використовує лише атрибути GROUP BY і ніякі функції агрегації не використовуються в інструкціях HAVING, ні в інструкціях GROUP BY, то умову можна перемістити в інструкцію WHERE. Оскільки інструкція WHERE виконується перед інструкцією HAVING, набагато дешевше перевірити умову вже в інструкції WHERE, оскільки це призведе до більш швидкого виконання, що дозволить оптимізатору SQL краще виконати запит. Не слід вважати семантичною помилкою, якщо умова всередині інструкції має використовувати будь-які функції агрегації SQL, оскільки вони не можуть бути присутніми в інструкції WHERE запиту. У наступному прикладі на рис. 3.19 має бути виявлена семантична помилка.

```
SELECT first_name
FROM students
GROUP BY class HAVING class IN('a');
```

Рисунок 3.19 – Помилка неефективна інструкція HAVING

У цьому прикладі умову IN всередині інструкції HAVING можна перемістити в інструкцію WHERE, оскільки вона не використовує будь-які функції агрегування. Це призведе до швидшого запиту, оскільки умова буде виконуватися при оцінці інструкції WHERE , тому GROUP BY виконуватиметься на меншому наборі результатів. Можливе виправлення для прикладу запиту наведено нижче на рис. 3.20.

```
SELECT first_name
FROM students
WHERE class = 'a';
```

Рисунок 3.20 – Запит без зайвого HAVING

Стратегія виявлення. Для виявлення цієї семантичної помилки спочатку перевіряються інструкції GROUP BY та HAVING, щоб визначити, чи використовуються якісь функції агрегування. Якщо це не так, інструкція HAVING потім аналізується у пошуках умов, які використовують лише атрибути у інструкції GROUP BY. Тому ці умови можна перемістити в інструкцію WHERE, що зробить запит більш ефективним

3.2.14 Непотрібна умова ORDER BY

Проблема: ця помилка стосується терміна GROUP BY, який може мати лише одне можливе значення, і в цьому випадку немає необхідності включати цей термін у інструкцію GROUP BY. Зокрема, щоразу, коли термін всередині інструкції WHERE прирівнюється до деякого постійного значення, немає сенсу,

щоб цей термін також був присутній всередині інструкції GROUP BY, оскільки це не вплине на результат запиту. У наступному прикладі на рис. 3.21 має бути виявлена семантична помилка.

```
SELECT division , count(id) AS ct
FROM test
WHERE role >=101 AND division=1
GROUP BY division;
```

Рисунок 3.21 – Помилка з непотрібною умовою ORDER BY

У цьому прикладі термін поділу з інструкції GROUP BY не потрібен, тому що він має постійне значення, що видно з умови Division = 1 всередині інструкції WHERE. Можливе виправлення для прикладу запиту наведено нижче на рис. 3.22.

```
SELECT division , count(id) AS ct
FROM test
WHERE role >=101 AND division=1;
```

Рисунок 3.22 – Запит без умови ORDER BY

Стратегія виявлення. Стратегія виявлення цієї семантичної помилки включає синтаксичний аналіз запиту і збереження стовпців констант, які прирівнюються до одного постійного значення всередині інструкції WHERE, а також не використовуються в жодних інших виразах. Потім для цих термінів також будемо перевіряти, чи є вони у інструкції GROUP BY, і в цьому випадку запропонований інструмент має видати попередження.

3.2.15 Непотрібна інструкція GROUP BY

Проблема: ця помилка стосується терміна ORDER BY, який може мати лише одне можливе значення, і в цьому випадку немає необхідності включати цей термін у інструкцію ORDER BY. Зокрема, щоразу, коли термін всередині

інструкції WHERE прирівнюється до деякого постійного значення, немає сенсу, щоб цей термін також був у інструкції ORDER BY, оскільки це не вплине на результат запиту. Більше того, це вірно у випадку, коли термін ORDER BY функціонально визначається попередніми термінами, які використовуються в реченні. Однак, оскільки ці випадки не можуть бути виявлені без додаткової інформації про схему бази даних, наш інструмент не перевірятиме такі ситуації. Крім того, щоразу, коли всередині інструкції ORDER BY присутній вираз CASE, задіяні терміни необхідні незалежно від того, чи мають вони лише одне можливе значення чи іншим чином функціонально визначаються попередніми термінами, використаними всередині інструкції. У наступному прикладі на рис. 3.22 має бути виявлена семантична помилка.

```
SELECT q.postid , a.postid , c.commentid
FROM posts q
WHERE q.postid = 1234
ORDER BY q.postid , a.postid , c.commentid;
```

Рисунок 3.22 – Помилка з використанням непотрібної інструкції GROUP BY

У цьому прикладі термін q.postid із інструкції ORDER BY не потрібен, оскільки він має постійне значення, як видно з умови q.postid = 1234 всередині інструкції WHERE . Можливе виправлення для прикладу запиту наведено на рис. 3.23.

```
SELECT q.postid , a.postid , c.commentid
FROM posts q
WHERE q.postid = 1234
ORDER BY a.postid , c.commentid;
```

Рисунок 3.23 – Запит без інструкції GROUP BY

Стратегія виявлення: для виявлення цієї семантичної помилки запит аналізується та зберігаються постійні стовпці, які прирівнюються до одного

постійного значення всередині інструкції WHERE. Потім для цих термінів будемо перевіряти, чи вони відсутні всередині інструкції ORDER BY , а також чи не включені вони в будь-який вираз CASE всередині інструкції ORDER BY. У цих випадках терміни не потрібні в інструкції ORDER BY і наш інструмент повинен видати попередження.

3.2.16 Непотрібна умова UNION

Проблема: запит, для якого два підзапити UNION використовують одні й самі вирази SELECT, одні й самі списки FROM, і навіть взаємовиключні умови WHERE, можна спростити, з'єднавши дві інструкції WHERE і з'єднавши їх з допомогою OR, повністю замінивши UNION. Те саме стосується і запитів з використанням інструкції UNION ALL, і в цьому випадку запит можна знову спростити, а дві різні інструкції WHERE з'єднати за допомогою OR. Для запропонованого інструменту умова взаємовиключних умов WHERE не може бути повністю перевірена без додаткової інформації про схему бази даних, тому це було спрощено до використання тільки різних виразів інструкції WHERE. У наступному прикладі на рис. 3.24 має бути виявлена семантична помилка.

```
SELECT id, name, age FROM student WHERE age < 15 UNION
SELECT id, name, age FROM student WHERE name LIKE '%a%' ORDER BY name;
```

Рисунок 3.24 – Запит з зайвим об’єднанням

У цьому прикладі запит містить одну інструкцію UNION, і ми можемо помітити, що вирази SELECT двох підзапитів такі ж, як і списки FROM, зокрема, в обох підзапитах використовується та сама таблиця student. Крім того, дві інструкції WHERE є взаємовиключними, тому запит можна спростити, замінивши UNION на OR і з'єднавши дві інструкції WHERE . Цей запит також страждає від іншої семантичної помилки, а саме від непотрібного сканування індексу, оскільки вираз LIKE починається з знаку підстановки. Можливе виправлення для прикладу запиту наведено нижче на рис. 3.25.

```
SELECT id, name, age
FROM student
WHERE age < 15 OR MATCH(name) AGAINST ('a' WITH QUERY EXPANSION)
ORDER BY name;
```

Рисунок 3.25 – Запит без помилки

Стратегія виявлення. Реалізація цієї стратегії спочатку визначає, чи включає запит будь-які інструкції UNION або UNION ALL. Якщо це так, то треба продовжувати зберігати вирази SELECT, а також списки FROM і вирази в інструкціях WHERE. Після аналізу всього запиту необхідно перевірити попарно зібрані списки, що включають три різні інструкції. Щоразу, коли два вирази списку SELECT, а також списки FROM з двох різних підзапитів збігаються, перевіряються інструкції WHERE, і якщо вони відрізняються, то видається попередження, що інформує користувача про те, що UNION може бути замінений умовою OR.

3.2.17 Непотрібна фраза GROUP BY

Проблема: якщо запит має такі самі атрибути у інструкції SELECT, перелічені у інструкції GROUP BY, і якщо в жодній з двох інструкцій не використовуються функції агрегування, то інструкцію GROUP BY можна замінити інструкцією SELECT DISTINCT. У більшості випадків оптимізатор SQL створюватиме однакові або схожі плани виконання для двох запитів, тому в цих випадках не завжди є виграш у продуктивності, однак, переписуючи запит, він стає коротшим і зрозумілішим. У більшості випадків використання інструкції GROUP BY фактично видаляє дублікати з результуючого набору, тому оператор DISTINCT краще підходить для використання, на відміну від ситуацій, коли використовуються функції агрегування, в яких дійсно потрібно використовувти GROUP BY. У наступному прикладі на рис. 3.26 має бути виявлена семантична помилка.

```
SELECT p.name
FROM pc p LEFT JOIN sc s ON p.color=s.color
WHERE s.color IS NOT NULL GROUP BY p.name;
```

Рисунок 3.26 – Помилка з непотрібною фразою GROUP BY

У цьому прикладі всі атрибути SELECT перераховані у інструкції GROUP BY, а функції агрегації також не використовуються, тому інструкцію GROUP BY можна опустити та замінити інструкцією SELECT DISTINCT, що зробить запит коротшим та зрозумілішим. Можливе виправлення на рис. 3.27 для прикладу запиту наведено нижче.

```
SELECT DISTINCT p.name
FROM pc p LEFT JOIN sc s ON p.color=s.color
WHERE s.color NOT NULL;
```

Рисунок 3.27 – Запит без непотрібного GROUP BY

Стратегія виявлення. Реалізація цієї стратегії спочатку перевіряє, чи використовуються будь-які функції агрегації у інструкціях SELECT чи GROUP BY. Якщо це так, то немає потреби продовжувати подальшу перевірку запиту на цю семантичну помилку. В іншому випадку треба продовжувати визначати всі терміни, що використовуються у реченні SELECT, а також ті, які використовуються у інструкції GROUP BY. Якщо є ідеальний збіг між цими двома наборами термінів, це означає, що інструкція GROUP BY може бути замінена SELECT DISTINCT і спрацьовує попередження про цю семантичну помилку.

3.2.18 Непотрібне загальне порівняння

Проблема: ця семантична помилка стосується операторів більше чи одно, а також операторів менше чи рівно. Коли всередині інструкції WHERE використовується оператор «більше або рівно», а найправіший термін оператора порівняння містить підзапит, в якому список SELECT використовує тільки функцію MAX з тим же атрибутом, що і крайній лівий термін порівняння, тоді

оператор «більше або рівне» може замінити оператором рівності. Аналогічно, те саме вірно для оператора менше або рівно і функції MIN . У наступному прикладі на рис. 3.28 має бути виявлена семантична помилка

```
SELECT name
FROM world
WHERE gdp >= (SELECT max(gdp) FROM world WHERE continent = 'europe')
```

Рисунок 3.28 – Помилка з непотрібним загальним порівнянням

У цьому прикладі оператор «більше або рівно» у інструкції WHERE можна замінити оператором «рівно» через вираз SELECT max(gdp) з підзапиту, що спростить розуміння запиту. Можливе виправлення для прикладу запиту наведено нижче на рис. 3.29.

```
SELECT name
FROM world
WHERE gdp = (SELECT max(gdp) FROM world WHERE continent = 'europe')
```

Рисунок 3.29 – Запит без порівняння

Стратегія виявлення. Ця семантична помилка виявляється шляхом перевірки операторів більше або рівне, або менше, або всередині інструкцій WHERE запитів. Щоразу, коли виявляються такі оператори, перевіряється інструкція, що йде за оператором, щоб визначити, чи використовується SELECT MAX або SELECT MIN, де функції MAX і MIN використовують той самий атрибут, що й крайній лівий термін оператора порівняння. У таких випадках порівняння можна замінити оператором рівності, що спрощує розуміння запиту.

3.2.19 Використання LIKE без символів підстановки

Проблема: якщо в запиті використовується вираз LIKE, що не містить підстановочних знаків, то LIKE можна і потрібно замінити оператором рівності. У наступному прикладі на рис. 3.30 має бути виявлена семантична помилка.

```
SELECT description
FROM tproduct
WHERE description LIKE 'diamond';
```

Рисунок 3.30 – Запит з не правильним використання LIKE

У цьому прикладі використовується вираз LIKE, проте він не містить знаків підстановки. У цьому випадку вираз LIKE можна замінити оператором рівності. Можливе виправлення для прикладу запиту наведено нижче на рис. 3.31.

```
SELECT description
FROM tproduct
WHERE description = 'diamond';
```

Рисунок 3.31 – Запит з LIKE з порівнянням

Стратегія виявлення. Для виявлення цієї помилки перевіряються вирази LIKE з проаналізованого запиту, щоб побачити, чи використовуються будь-які знаки підстановки. Якщо це не так, видається попередження про наявність цієї семантичної помилки.

3.2.20 Непотрібний список SELECT в EXISTS

Проблема: при використанні підзапиту EXISTS список SELECT із підзапиту не важливий, оскільки EXISTS перевіряє лише наявність рядка, незалежно від вибраних стовпців. Коли запит виконується, як тільки запит EXISTS повертає один рядок, його виконання зупиняється і батьківському запиту повертається TRUE. Отже, у підзапитах EXISTS список SELECT не важливий і слід використовувати щось просте, наприклад оператор вибору всіх записів (*). Однак це не стосується підзапитів IN, де дійсно важливий список SELECT всередині підзапиту. У наступному прикладі на рис. 3.32 має бути виявлена семантична помилка.

```
SELECT id, name
FROM tbl_bkp t1
WHERE NOT EXISTS ( SELECT id, name FROM tbl_namecode WHERE id=t1.id AND name =
t1.name);
```

Рисунок 3.32 – Помилка непотрібного списку SELECT в EXISTS

У цьому прикладі всередині підзапиту EXISTS список SELECT містить декілька атрибутів. Це надмірно, оскільки батьківському запиту буде повернено лише значення TRUE або FALSE, тому список SELECT можна замінити більш простим, наприклад, оператором SELECT ALL. Можливе виправлення для прикладу запиту наведено нижче на рис. 3.33.

```
SELECT id, name
FROM tbl_bkp t1
WHERE NOT EXISTS ( SELECT * FROM tbl_namecode WHERE id=t1.id AND name = t1.name);
```

Рисунок 3.33 – Запит без помилки непотрібного списку SELECT в EXISTS

Стратегія виявлення. Виявлення цієї семантичної помилки здійснюється шляхом аналізу запиту та пошуку підзапитів EXISTS. При їх виявленні перевіряється список SELECT, і якщо в ньому не використовується простий оператор, такий як оператор вибору всіх або один атрибут, видається попередження про цю помилку.

3.2.21 Непотрібне сканування індексу

Проблема: ця семантична помилка стосується використання підстановочних знаків на початку слова при пошуку за допомогою оператора LIKE. У операторі SQL використання LIKE часто є причиною непередбачуваних проблем із продуктивністю через неефективне індексування. Положення підстановочних знаків усередині співпадаючого слова може призвести до значного падіння продуктивності. При використанні фільтра LIKE тільки символи, які з'являються перед першим підстановочним знаком, можуть використовуватися в якості

індексів для прискореного співставлення із зразком. Інші символи є просто предикатами фільтра, які не допомагають зменшити діапазон сканованого індексу. Таким чином, оператор LIKE може містити предикат доступу, частину перед першим встановленим знаком, і предикат фільтра, що залишилися символи. Відсканований діапазон індексів стає меншим, якщо попередній доступ більш вибірковий, що призводить до підвищення продуктивності запитів, оскільки пошук по індексу виконується швидше. Таким чином, якщо оператор LIKE починається з підстановочного знака, то предикат доступу відсутній, що суперечить цілим індексам, що призводить до сканування всієї таблиці бази даних. У наступному прикладі на рис. 3.34 повинна бути виявлена семантична помилка.

```
SELECT * FROM street WHERE street_name LIKE '%park%ave%10%';
```

Рисунок 3.34 – Помилка з непотрібним скануванням індексу

В цьому прикладі умова пошуку для виявлення LIKE починається з підстановочного знака (відповідного нулю або більше символів), тому предикат доступу відсутній, тому запит буде виконувати повне сканування індексу таблиці. Для оптимізації запиту було б краще або надати предикат доступу, або, якщо це неможливо, використати функцію пошуку, яка приведе до перегляду повнотекстового індексу. У цьому випадку запит можна відправити, використовуючи синтаксис MATCH і AGAINST. Оператор MATCH приймає в якості вхідних даних список, розділений зап'ятими, який називає стовбці для пошуку, в той час як інструкція AGAINST приймає в якості вхідних даних рядок для пошуку разом з необов'язковим модифікатором, який включає тип пошуку. Це може бути логічний пошук або пошук із розширенням запиту.

Стратегія виявлення. Реалізація для виявлення цієї семантичної помилки аналізує запит на пошук операторів LIKE. Кожного разу, коли такий вираз знайдено, цей оператор перевіряється, щоб визначити, починається чи він з

підстановочного знака. Мотивація: функцію агрегування COUNT можна використовувати, як з аргументом, так і без нього, і в цьому випадку частіше всього використовується символ зірочки (*). Всякий раз, коли у функції COUNT не використовується DISTINCT і аргумент не може бути нульовим, ці дві версії функції COUNT є рівноцінними, і перевага віддається варіанту без аргументів, оскільки він спрощує розуміння і читання запиту. У запропонованій реалізації цієї стратегії виявлено, що не можливо перевірити умови. В цьому випадку генерується попередження про відсутність попереднього доступу, тому буде виконано повне сканування індексу.

3.2.22 Непотрібний DISTINCT в агрегаціях

Проблема: для функцій агрегації MIN і MAX ключове слово DISTINCT ніколи не потрібне, оскільки воно не впливає на результати основного запиту. Крім того, у більшості випадків наявність дублікатів, ймовірно, має велике значення при вирахуванні результатів функцій агрегування SUM і AVG, тому не слід виключати їх, якщо для цього немає важких причин. Тому наявність ключового слова DISTINCT всередині цих функцій агрегування є стороннім і має викликати попередження. Для інших функцій агрегації неможливо визначити, необхідний DISTINCT або немає всередині інструкції, не має додаткової інформації про завдання запиту, тому інші функції агрегації виключаються з цієї перевірки. У наступному прикладі на рис. 3.35 повинна бути виявлена семантична помилка.

```
SELECT SUM(DISTINCT money) AS sum_money , m.created_at
FROM receipt r
JOIN material m
GROUP BY m.created_at;
```

Рисунок 3.35 – Помилка непотрібного DISTINCT в агрегаціях

У цьому прикладі ключове слово DISTINCT використовується всередині функції SUM. Якщо для цього немає ніяких причин, то DISTINCT тут виявлено

дивним, так як унікальні значення, швидше за все, суттєві в цьому випадку, однак, немає додаткової інформації про завдання, для якого був написаний запит, запропонований інструмент видасть попередження про наявну семантичну помилку. Не маючи додаткової інформації щодо базової задачі, для якої може бути написаний запит, не можливо показати виправлення цього запиту.

Стратегія виявлення. Стратегія виявлення перевіряє, чи присутня в запиті будь-яка з функцій агрегації MIN, MAX, SUM або AVG. Потім для кожної з них визначається, чи використовувалося ключове слово DISTINCT всередині інструкції функцій, і якщо це так, видається попередження.

3.2.23 Непотрібний атрибут GROUP BY

Проблема: щоразу, коли атрибут, який з'являється в інструкції GROUP BY, функціонально визначається іншими атрибутами, і якщо вона не з'являється в інструкціях SELECT або HAVING поза функціями агрегації, тоді її можна повністю видалити з GROUP BY. Для нашого інструменту ця умова ослаблена, оскільки увага звертається лише на запит, не знаючи схеми бази даних, тому неможливо визначити, чи атрибут визначається функціонально іншими атрибутами чи ні. Тому є потреба перевірити тільки атрибут, який з'являється всередині інструкції GROUP BY, але не використовується ні в інструкціях SELECT, ні в інструкціях HAVING поза функціями агрегування, і в цьому випадку буде активовано попередження про цю семантичну помилку. У наступному прикладі на рис. 3.36 має бути виявлена семантична помилка.

```
SELECT count(*) AS countbyid
FROM items
WHERE fkid = 2003799
GROUP BY fkid
HAVING countbyid >1
ORDER BY countbyid;
```

Рисунок 3.36 – Помилка непотрібного атрибута GROUP BY

У цьому прикладі атрибут fkid є у інструкції GROUP BY, але не використовується ні в інструкціях SELECT, ні в інструкціях HAVING поза функціями агрегування, тому його можна видалити з GROUP BY. Це ще більше спростить запит, оскільки fkid є єдиним атрибутом, присутнім у GROUP BY. Крім того, цей запит містить ще одну семантичну помилку, так як fkid може мати тільки одне значення, тому що прирівнюється до певної константи у інструкції WHERE, тому групування не вплине на результат. Можливе виправлення для прикладу запиту наведено нижче на рис. 3.37.

```
SELECT count(*) AS countbyid
FROM items
WHERE fkid = 2003799
HAVING countbyid >1
ORDER BY countbyid;
```

Рисунок 3.37 – Запит без непотрібного атрибута GROUP BY

Стратегія виявлення. Стратегія виявлення цієї помилки аналізує запит і відстежує атрибути, що з’являються в інструкціях SELECT і HAVING поза функціями агрегації. Крім того, атрибути, які з’являються в інструкції GROUP BY, також зберігаються. Після завершення аналізу запиту перевіряються атрибути, знайдені в інструкції GROUP BY, яких немає в жодному з двох списків з атрибутами, виявленими в інструкціях SELECT або HAVING. Для кожного з цих атрибутів виникає попередження, яке вказує на те, що в запиті виявлено семантичну помилку.

3.2.24 Непотрібний аргумент COUNT

Проблема: функцію агрегування COUNT можна використовувати як з аргументом, так і без нього, і в цьому випадку найчастіше використовується символ зірочки (*). Щоразу, коли в функції COUNT не використовується DISTINCT і аргумент не може бути нульовим, ці дві версії функції COUNT еквівалентні, і перевага надається варіанту без аргументів, оскільки він спрощує

розуміння і читання запиту. У запропонованій реалізації цієї стратегії виявлення не можливо перевірити умову, пов'язану з аргументами функції COUNT, які не дорівнюють нулю, оскільки для цього буде потрібна додаткова інформація про схему бази даних. Тому було зроблене припущення, що аргументи, які використовуються всередині функцій COUNT, не дорівнюють нулю. У наступному прикладі на рис. 3.38 має бути виявлена семантична помилка.

```
SELECT m.time , COUNT(r.seconds)
FROM dbo.minutes m
GROUP BY m.time;
```

Рисунок 3.38 – Помилка непотрібного аргументу COUNT

У цьому прикладі в функції COUNT не використовується DISTINCT, і використовується лише один аргумент. Таким чином, вираз всередині COUNT можна спростити, і замість цього можна використовувати оператор (*). Можливе виправлення для прикладу запиту наведено нижче на рис. 3.39.

```
SELECT m.time , COUNT(*)
FROM dbo.minutes m
GROUP BY m.time;
```

Рисунок 3.39 – Оптимізований запит

Стратегія виявлення. Для виявлення цієї семантичної помилки проводився аналіз запиту і перевірялося, чи використовуються будь-які функції COUNT. Потім для кожного з них треба перевіряти, чи DISTINCT використовується всередині інструкції COUNT. Якщо це не так, і інструкція містить лише аргумент, то видається попередження, що вказує, що інструкцію COUNT можна спростити, натомість використовуючи оператор (*).

3.3 Висновок

У цьому розділі було розглянуто ряд евристик для виявлення семантичних помилок у SQL-запитах. Ці евристики використовувалися для створення набору правил у запропонованому інструменті статичного аналізу. Помилки, які виявляються за допомогою цих евристик, взяті з повного списку семантичних помилок, що був скомпільований у роботі Брасса та Голдберга. Для збирання підмножини семантичних помилок, ми скористалися інструментом SQL Enlight, який дозволив нам виявити різні проблеми у SQL-запитах.

У порівнянні з іншими інструментами, які використовують закритий вихідний код, інструмент статичного аналізу розроблений на основі правил, має вигоду у гнучкості та можливості налаштування. Хоча він використовує жорстку реалізацію без додаткової інформації про завдання чи схеми бази даних, це забезпечує стабільність та надійність виявлення помилок, навіть якщо деякі попередження можуть бути хибними позитивами. Ця компромісна модель налаштування дозволяє інструменту працювати ефективно при обмежених знаннях про конкретні запити.

У наступних розділах наведено опис кожної семантичної помилки, супроводжується прикладами SQL-запитів, що містять ці помилки, і надається реалізація кожної стратегії виявлення, разом із відповідними евристиками, використовуваними у нашому інструменті. Цей підхід дозволив в ході дослідження не лише ідентифікувати семантичні помилки, але й надати розробникам необхідні засоби для їх виправлення та покращення якості SQL-коду.

4. ІНСТРУМЕНТ СТАТИЧНОГО АНАЛІЗУ ДЛЯ ВИЯВЛЕННЯ СЕМАНТИЧНИХ ПОМИЛОК

4.1 Реалізація

Для виявлення раніше описаних семантичних помилок в SQL-запитах було запропоновано реалізувати інструмент статичного аналізу, де кожна стратегія виявлення є одним правилом для вилову іншої семантичної помилки. Інструмент був розроблений на Java, і JSQLParser використовувався для аналізу запитів SQL. JSQLParser – це бібліотека з відкритим вихідним кодом, яка перетворює SQL на дерево класів Java, що переглядається, для якого потім можна використовувати шаблон проектування відвідувача для перевірки всього запиту. Бібліотека була обрана для цього проекту через її широку підтримку різних синтаксисів, таких як Oracle, SqlServer, MySQL та PostgreSQL, а також через її відкритий вихідний код. Загальний потік починається з деякого запиту вхідного, представленого інструменту. Потім для аналізу запиту використовується JSQLParser. Кожне правило виявлення, реалізоване як JSQLParserVisitor, потім виконується для вхідного запиту, і кожного разу, коли правило порушується, запит позначається як проблемний, а попередження про виявлену семантичну помилку видається інструментом та відображається користувачеві. Огляд загального потоку інструмента представлено рисунку 4.1.

Стратегії виявлення для кожної з семантичних помилок, описаних у попередніх розділах, реалізовані у вигляді окремих правил, які використовують шаблон проектування відвідувача для аналізу SQL-запитів у пошуках різних семантичних помилок. Кожне правило фокусується на виявленні одного типу помилок. Крім того, інструмент легко відкривається для розширення, що дозволяє реалізувати безліч інших правил виявлення інших типів проблем. Оскільки для більшості правил існує ряд подібностей, таких як аналіз запитів під час пошуку джерел таблиць або імен змінних кортежів, існує загальний інтерфейс, який використовується всіма правилами.

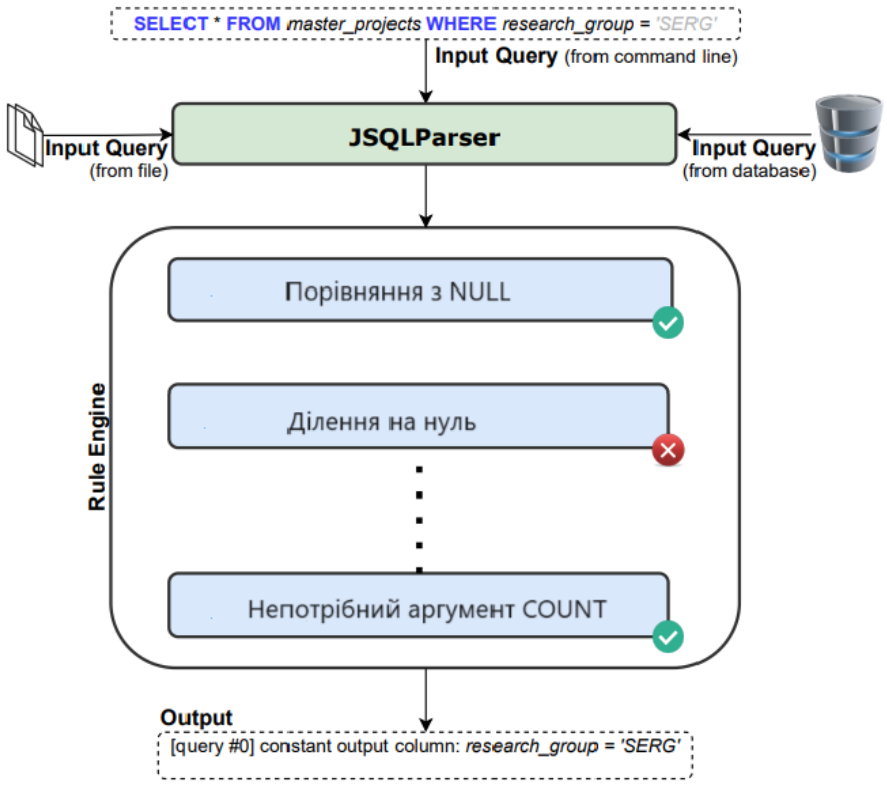


Рисунок 4.1 – Потік роботи інструменту

Одним із ключових аспектів, який необхідно було мати для запропонованого інструменту, була простота додавання інших правил у майбутньому. У нашій поточній структурі це легко зробити, створивши новий клас Java і реалізувавши основний інтерфейс, який, серед іншого, вже займається розбором запитів SQL. Розробникам тоді потрібно лише перевизначити методи, котрим потрібна індивідуальна функціональність для стратегії виявлення. Це, наприклад, дає можливість писати складні евристики для правил без необхідності переписувати логіку синтаксичного аналізу, що економить час і потенційні помилки. Ще одним важливим аспектом є те, що правила можна краще тестувати, зосередивши увагу лише на новій функціональності, яка спеціально адаптована для стратегії виявлення, оскільки логіка синтаксичного аналізу вже протестована нашою структурою. Інструмент має три різні режими роботи. У першому вводиться запит через командний рядок та перевірятимуться всі правила на наявність семантичних помилок у вхідних даних. Це можна використовувати кожного разу, коли необхідно виконати швидку перевірку для одного запиту, щоб перевірити, чи є він

семантично правильним чи ні. Другий режим роботи дозволяє вказати вхідний файл. Потім інструмент прочитає всі запити з файлу, які мають бути розділені крапкою з комою (;), і відобразить усі помилки, виявлені для кожного з запитів SQL, присутніх у файлі. Цей параметр, можливо, корисніший для проведення аналізу на більшому вхідному наборі даних. Нарешті, у третьому режимі роботи інструмент використовує платформу Spring25 для підключення до бази MySQL для отримання вхідних запитів і запуску правил для них. Цей режим використовувався під час дослідження для аналізу великого набору даних SQL запитів, зібраних із повідомлень StackOverflow. Потім інструмент також зберігає результати у цій базі даних, які згодом можна використовувати для подальшого аналізу.

Кожне правило значною мірою покрито модульними тестами, які намагаються враховувати як загальні, так і часткові випадки. Загалом для запропонованого інструменту виявлення реалізовано понад 150 модульних тестів для 24 правил. Інші тести включають автоматизований сценарій, який використовує колекцію з більш ніж 400 перевірених вручну запитів, які виконуються за допомогою інструмента та порівнюють результат із очікуваним результатом. Ще 50 запитів були перевірені вручну, щоб переконатися, що вони не містять семантичних помилок, після чого вони були представлені як вхідні дані для інструменту, щоб переконатися, що попередження не з’являються, а пізніше також були додані в автоматичні тести. Загальна якість коду для цього інструмента відстежувалася в Better Code Hub, в результаті чого була отримана загальна оцінка якості 8 з 10, при цьому єдиною областю, де не було набрано максимальну кількість балів, був поділ проблем у модулях через загальний інтерфейс, який використовує всі наші правила

4.2 Перевірка

Крім різних модульних тестів, реалізованих для перевірки кожного правила, інструмент було перевірено з використанням ручного процесу дослідження понад 400 SQL-запитів та перевірки правильності вихідних даних інструменту. Для

кожного з 24 правил виявлення семантичних помилок, реалізованих у запропонованому інструменті, спочатку було обрано випадковий набір із 20 запитів, зібраних із повідомлень StackOverflow, для яких інструмент повідомив про семантичні помилки для подальшого аналізу вручну. Інструмент був налаштований таким чином, що кожне правило виконувалося ізольовано, що означає, що кожне правило докладно аналізувалося окремо від інших, щоб підтвердити правильність як реалізації, так і базової евристики, що використовується. Щоразу, коли інструмент помилково повідомляв про семантичну помилку для певного запиту, код аналізувався та виправлявся.

Крім того, модульні тести також були покращені, щоб гарантувати, що проблеми не виникнуть на пізнішому етапі, коли можуть бути внесені подальші зміни. Якщо для певного правила було виявлено кілька проблем, після перевірки вихідного набору із 20 запитів процес повторювався шляхом вибору нового випадкового набору з 20 запитів із набору даних. Коли у випадковій вибірці більше помилок не виявлено або жодна з помилок, що повідомляються, могли бути усунені через те, що запит не міг бути правильно проаналізований бібліотекою JSQLParser або іншим чином, правило було вважати перевіреним. Було створено набір даних приблизно з 400 запитів, де для кожного правила реєструється остаточний набір із 20 випадково вибраних запитів разом із семантичними помилками, про які повідомив інструмент. Крім того, був реалізований сценарій, який використовує цей набір даних разом з очікуваними результатами для виконання автоматичних тестів інструменту для покращення тестування правил. Остаточна реалізація нашого інструменту правильно виявила 387 запитів із семантичними помилками, а також 40 запитів без будь-яких помилок із тестового набору даних із 550 запитів, довівши точність інструменту до 97%. Результати цього ручного аналізу для нашого інструменту представлені на табл. 4.1.

Таблиця 4.1 – зведена таблиця результатів інструменту

	Передбачений (-)	Передбачений (+)
Дійсний (-)	TN= 50	FT = 13
Дійсний (+)	FN = 0	TP = 487

4.3 Результати

4.3.1 Поширеність семантичних помилок у запитах SQL

Ми запустили інструмент статичного аналізу на основі правил, що містить 24 правила виявлення різних семантичних помилок SQL, як описано в Розділі 3.2, для всіх зібраних запитів. Спостерігалося, що з усіх 191 994 запитів було виявлено 36 818 запитів, які містять хоча б одну семантичну помилку, а це означає, що 19,17% запитів містили якусь семантичну проблему у своєму формулюванні. Знову ж таки, це показує необхідність надання розробникам інструментів виявлення семантичних помилок, щоб покращити загальну якість SQL-запитів, що використовуються в реальних додатках.

На рисунку 4.2 показано поширеність виявлених семантичних помилок у наборі даних Stack Overflow. З усіх цих помилок найчастішою є відсутня семантична помилка предикатів сполуки (0019) із середньою поширеністю 5,98%. Другою за частотою семантичної помилкою є стовпець постійного виведення (0002), який має медіанну поширеність 3,62%, за якою слідує помилка непотрібної кількості аргументів (0012) з медіанною поширеністю 3,35%. Не було виявлено семантичних помилок, пов'язаних з використанням ідентичних змінних кортежу (0005) у зібраному наборі даних. Для більшості інших правил медіана поширеності була нижче 0,5%, а це означає, що з 200 запитів один буде торкнутися семантичної помилки. Далі було проаналізовано правило виявлення відсутніх предикатів з'єднання, вибравши 20 випадкових запитів, для яких повідомлялося про цю помилку, та вручну перевірено результати нашого

інструменту. У всіх випадках дійшли висновку, що неправдивих спрацьовувань не було і всі помилки були виявлені коректно. Той самий ручний аналіз був проведений і для всіх інших правил, і ми включили результати до Додатку А.

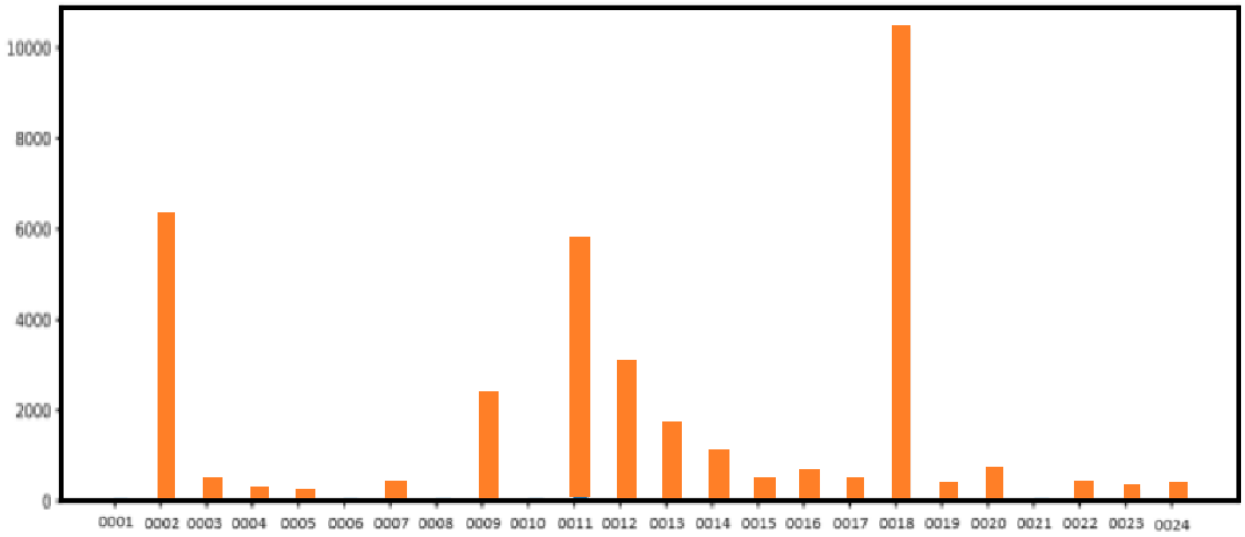


Рисунок 4.2 – Поширеність семантичних помилок у запитах SQL для набору даних взятого з StackOverflow

З усіх зібраних запитів, 19,17% містили хоча б одну семантичну помилку. Найбільш поширеними помилками є відсутні предикати з'єднання (0018), за якими йдуть постійні помилки стовпця виведення (0002) та непотрібні помилки аргументу лічильника (E011).

Інші попередні дослідження, такі як проведені Брасом і Голдбергом [9] і Голдбергом [20], також дійшли висновку, що семантична помилка відсутніх предикатів сполуки часто зустрічається в SQL-запитах. Одна помітна різниця між попередніми дослідженнями та нашою роботою пов'язана з наборами даних, що використовуються. Більшість минулих робіт ґрунтується на запитах, зібраних з курсів SQL, що означає, що запити пишуться студентами на іспитах та інших домашніх завданнях. У нашому наборі даних використовувалися запити з різних проектів з відкритим кодом, тому середня поширеність трохи нижче. Цікаво, що Голдберг [20] повідомляє про 3,6% поширеності семантичної помилки постійного вихідного стовпця, знову ж таки в наборі даних запитів, написаних студентами на

різних іспитах, що дуже близько до медіанної поширеності, яку ми виявили в нашому дослідженні для набору даних StackOverflow 3,62%

На малюнку 4.3 представлено поширеність виявлених семантичних помилок набору даних EvoSQL3. Цей набір даних містить лише запити SQL SELECT, знайдені у трьох проектах з відкритим вихідним кодом, які відстежуються на GitHub. При подальшій ручній перевірці було помічено, більшість запитів страждають від проблемного семантичного коду SQL неявних стовпців, як описано у роботах Muse et al. [34] та Карвін [23]. У нашій реалізації ця семантична помилка SQL відповідає помилці постійного вихідного стовпця (0002), і, як видно з результатів нашого інструменту, ця помилка дійсно є найчастішою в наборі EvoSQL даних із середньою поширеністю 43,23%. Для інших семантичних помилок наш інструмент виявив медіанну поширеність менше 0,4% для цього набору даних.

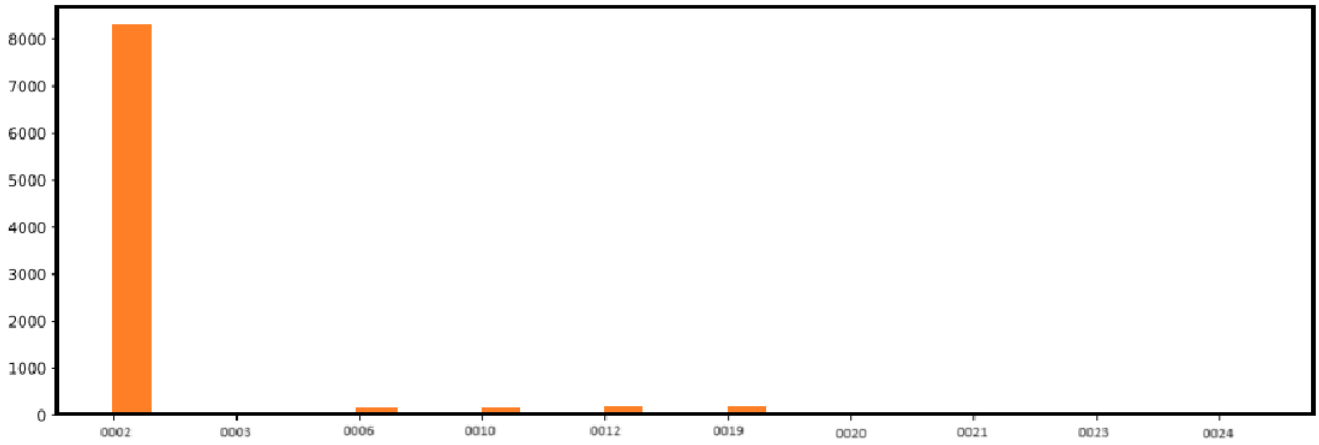


Рисунок 4.3 – Поширеність семантичних помилок у запитах SQL для набору даних взятого з EvoSQL.

4.3.2 Частота семантичних помилок у запитах SQL

На рисунку 4.4 показано матрицю спільного входження для всього нашого набору даних, де кожен запис у матриці представляє коефіцієнт подібності Жаккара, як описано Еком і Волтманом [15] (порожні клітинки мають нульове значення). З цього добре видно, що більшість значень подібності є досить

низькими, однак існує 20% подібність між помилками 0012 та 0013, а також 15% подібність між помилками 0012 та 0019, а потім 15% подібність для помилок 0002 та 0019. Індекс Жаккара – це міра подібності між двома наборами даних у діапазоні від 0% до 100%, що означає, що чим вищий відсоток, тим більше подібні два набори один до одного. У нашому випадку для двох помилок, скажімо, A та B, якщо вони мають міру схожості на 30%, це означає, що якщо помилку A виявлено в одному запиті, то існує 30% ймовірність того, що помилка B також присутня в цьому запиті.

Цей вид інформації може бути корисним при розробці майбутніх інструментів виявлення, а також систем прогнозування, які можна інтегрувати в середовище розробки (IDE), щоб сповістити розробників про певні помилки у їхніх запитах, які вже були виявлені. Подальший ручний аналіз запитів, в яких відбувається одночасне виникнення помилок непотрібного підрахунку аргументів (0012) та непотрібного групування за атрибутом (0013), показує, що, справді, ці два типи семантичних помилок частіше спостерігаються разом, оскільки зазвичай у конструкції SELECT інструкція GROUP BY використовується з метою агрегації певних даних.

Ще одним цікавим відкриттям є те, що найбільш ймовірною є одночасна поява помилки відсутності об'єднуючого предиката (0019) з іншими типами помилок. Конкретно, існує 15% схожість між запитами, які містять помилку 0019, і або помилкою 0002, або помилкою 0012, і 10% схожість між запитами, які містять помилку 0019, і або помилкою 0013, або помилкою 0015. Це також відповідає результатам першого дослідження (RQ1), яке показало, що помилка 0019 є найпоширенішою в нашому повному наборі даних, і, відповідно, вона спостерігається одночасно з 4 іншими типами помилок.

Одночасна поява семантичних помилок у SQL-запитах для всього нашого набору даних досить низька, що вказує на те, що запити рідко містять більше семантичної помилки. Найбільша схожість між двома помилками становить 20% для непотрібного аргументу кількості (0012) та непотрібного угруповання за атрибутом (0013).

4.3.3 Кореляція між складністю запиту та кількістю семантичних помилок

Для аналізу кореляції між складністю запиту та кількістю семантичних помилок на рис. 4.5 приведено блок-діаграму. На рис. 4.6 також показано розподіл оцінок складності запитів. Дані, представлені на цих малюнках, включають всю запропоновану колекцію SQL-запитів як наборів даних StackOverflow, так і наборів даних EvoSQL.

Щоб додатково визначити, чи існує кореляція між складністю запиту та кількістю семантичних помилок, було виконано тест одностороннього дисперсійного аналізу (ANOVA). Для нашого однофакторного тесту ANOVA (α = 0,05) було визначено наступну нульову та альтернативну гіпотези:

– H0 (нульова гіпотеза): всі групи мають те саме середнє, μ1 = μ2 = ⋯ = μ5

– H1 (альтернативна гіпотеза): принаймні один із середніх відрізняється

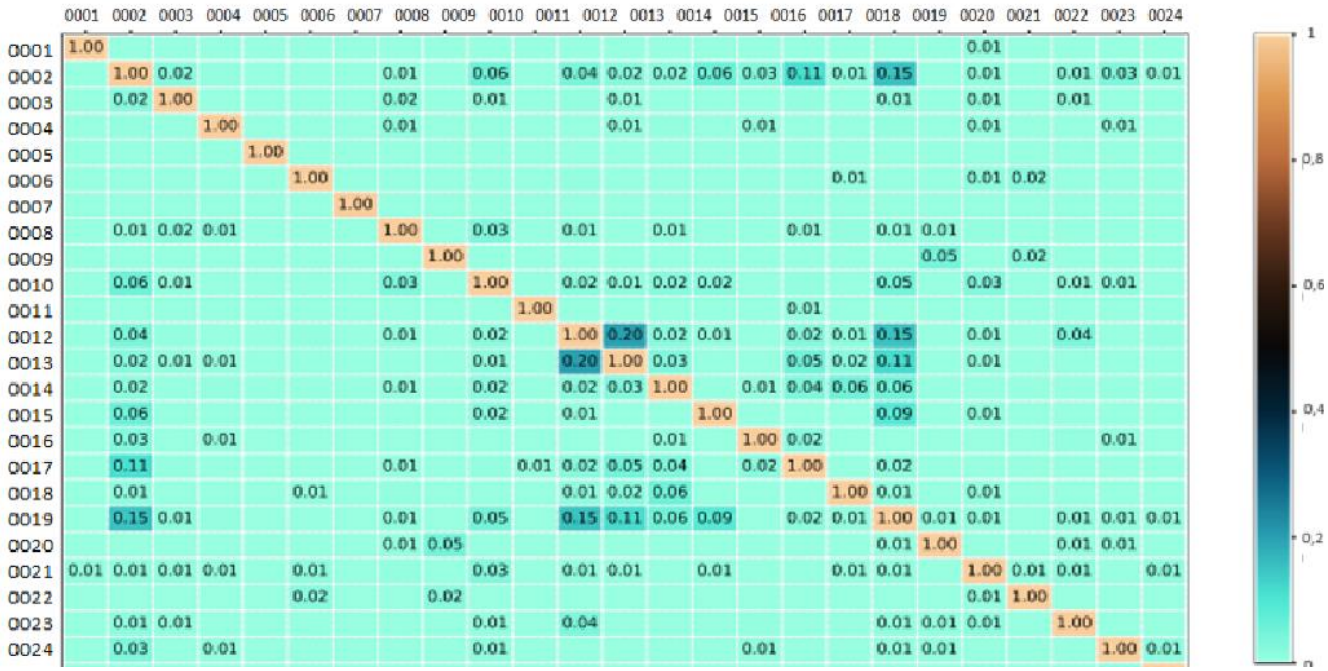


Рисунок 4.4 – Матриця збігів семантичних помилок

В ході роботи отримано, що відповідне значення p дорівнює 1,14e-168, тому, оскільки це значення менше обраного нами α = 0,05, було відкинуто нульову

гіпотезу і, отже, можемо сказати, що між групами, що розглядаються, існує значна різниця.

Обчислюючи середню оцінку складності для кожної категорії, отримано, що запити з однією семантичною помилкою мають середню складність 7, запити з двома помилками мають середню складність 10, запити з 3 помилками мають середню складність 12, запити з 4 помилками мають середню складність 13 і, нарешті, запити з 5 помилками мають середню складність 26.

Виконання тесту ANOVA показує, що є значна різниця між складністю запитів. Є також переконливі докази того, що складні запити більш схильні до семантичних помилок SQL.

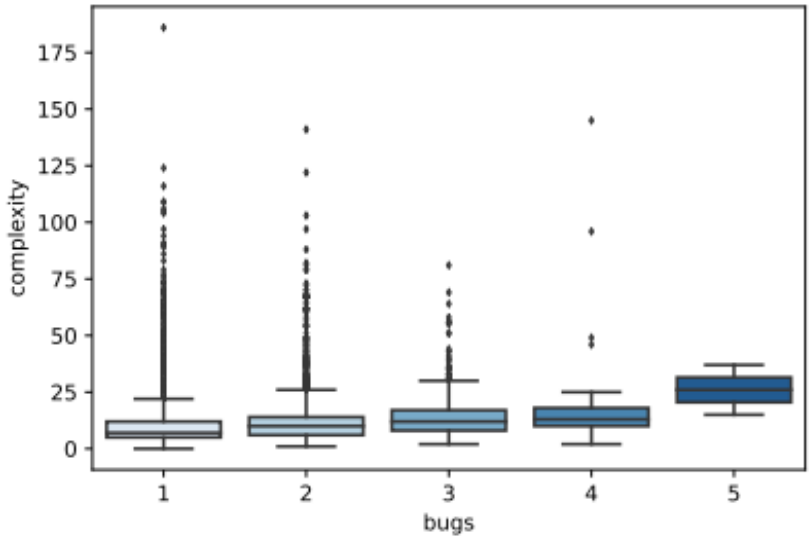


Рисунок 4.5 – Кількість семантичних помилок при різній складності

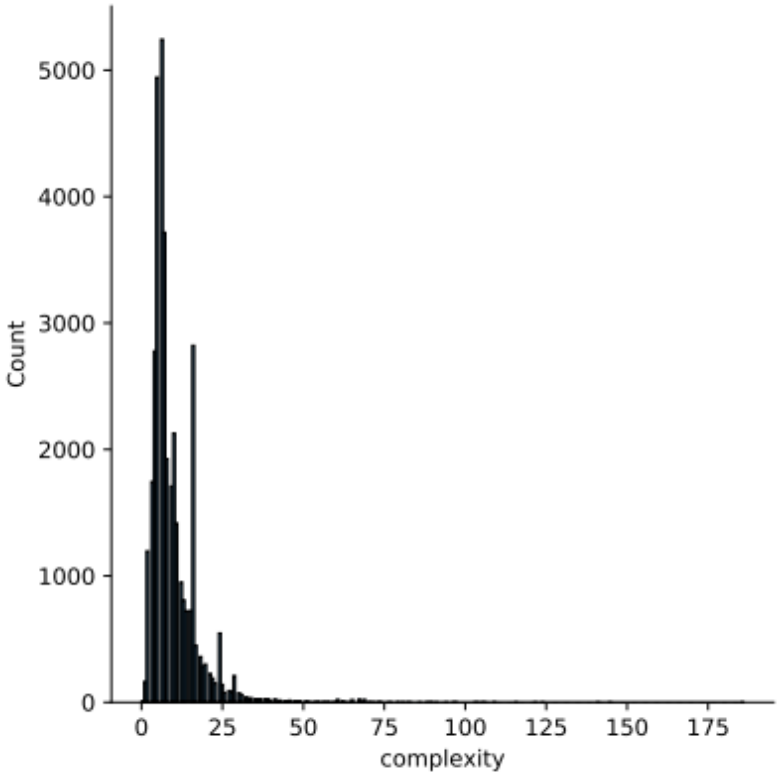


Рисунок 4.6 – Розподілення складності запитів

4.4 Загрози для валідності

У цьому розділі проведено обговорення загрози для валідності цього дослідження та заходи, які було вжито для їхнього зменшення.

Внутрішня валідність. Реалізація евристик для запропонованого інструменту статичного аналізу на основі правил може піддаватись загрозам внутрішньої валідності. Щоб допомогти в подоланні цього, кожне з 24 правил перевірено за допомогою обширного комплекту юніт-тестів як для загальних, так і для спеціальних випадкових сценаріїв. Крім того, проводився ручний аналіз понад 400 запитів, вибравши для кожного з 24 правил випадковий піднабір з 20 запитів, які були ретельно перевірені, щоб переконатися, що результати, повернені евристиками, є вірними. Нарешті, також було намагання максимально можливо узгодити реалізацію наших евристик з різними деталями реалізації, знайденими в попередніх дослідженнях.

Зовнішня валідність. Незважаючи на те, що зібраний набір даних містить різноманітний набір SQL-запитів, ми виявили, що SELECT-запитів значно більше, ніж інших типів, таких як INSERT, UPDATE або DELETE-запити, тому ці останні

можуть бути недостатньо представлені в цьому дослідженні. Це особливо помітно в наборі даних EvoSQL, який містить лише SELECT-запити. Однак, витягуючи запити з повідомлень на StackOverflow, з'ясувалося, що це може стати в пригоді для подолання деяких з цих проблем, створить достатньо різноманітну колекцію запитів і в той же час зробить представлені результати загальними для SQL-запитів в будь-якій програмній системі.

Також у цьому розділі ми обговорюємо можливі вдосконалення нашого інструменту виявлення, а також алгоритму аналізу запитів, і надаємо деякі ідеї для майбутніх досліджень, які можуть використовувати наш обширний набір даних SQL-запитів.

1. Інтегрувати інструмент статичного аналізу на основі правил для виявлення семантичних проблем у SQL-запитах у плагін для PHPStrom або Visual Studio Code. Надаючи підтримку для нашого інструменту у вигляді плагіна для IDE, розробники можуть легше користуватися ним. Інші переваги можуть полягати в виявленні проблем у реальному часі, коли запити вводяться в IDE, а також під час виконання програми, використовуючи інтеграцію із фреймворком SpringBoot.

2. Покращити алгоритм пошуку запитів. Для отримання запитів із StackOverflow було б цікаво дослідити, чи можна досягти кращих результатів (знаходити більше запитів), використовуючи інструменти, такі як GitHub's linguist або Guesslang, для визначення мови програмування у фрагменті коду.

3. Використовуйте цей набір даних для вивчення того, чи відзначають розробники семантичні проблеми SQL у коментарях на StackOverflow. Було б цікаво проаналізувати в майбутніх дослідженнях, чи помічають розробники різні типи семантичних проблем при заданні або відповіді на питання на StackOverflow. Ще одним потенційним напрямком досліджень може бути вивчення еволюції публікацій, які можуть страждати від семантичних проблем з часом і визначення того, чи вирішуються ці проблеми.

4. Покращити інструмент статичного аналізу на основі правил, розширивши кількість семантичних проблем, які можна виявити. Навіть якщо інструмент вже

може виявляти найбільш поширені типи семантичних проблем у SQL-запитах, було б дуже цікаво продовжувати розширювати цей список, включаючи інші проблеми, такі як ті, що присутні в закритому інструменті SQLEnlight. Існує вичерпний список з деякими семантичними правилами, які підтримує цей інструмент, тож у майбутньому можна було б їх включити в наш інструмент виявлення.

4.5 Рекомендації

Рекомендації для розробників. З такою кількістю додатків, які використовують SQL для запитів, розробникам зараз ніколи не важливо отримувати результати їх запитів, які страждають від різних семантичних проблем. Як ми показуємо в нашому емпіричному дослідженні, в Інтернеті вже існує досить багато запитів, які вже страждають від проблеми такого типу. Тому розробники повинні спробувати познайомитися не тільки з цими проявами семантичних проблем і тими, які вони виявляють під час роботи з SQL, але й звернути більше уваги на виявлення цих проблем, а також на їх усунення відразу ж після їх виявлення.

Рекомендації для виробників інструментів. Основний акцент слід робити на розробці інструментів для масового використання. Вибір цього як основного фактора при створенні нових інструментів повинен сприяти тому, щоб розробники дійсно бачили цінність у використанні інструменту. Зокрема, наші висновки показують, що наразі розробникам важко перевіряти свої SQL-запити на семантичні проблеми через одну основну проблему - відсутність адекватних інструментів і підтримки. Поточні реалізації інструментів, які інтегруються з інтегрованими середовищами розробки, дуже обмежені у плані підтримки виявлення семантичних проблем для запитів, більше того, наскільки нам відомо, немає інструментів, які інтегровані з популярними фреймворками розробки, такими як SpringBoot1, для перевірки цих типів проблем під час виконання додатка. Особливо цікавим для майбутніх інструментів може бути інтеграція

виявлення проблем на основі мір подібності, а також попередження на ранніх етапах на основі метрик, таких як складність запиту.

Рекомендації для дослідників. Основуючись на виявлених фактах під час проведення цього дослідження, можна сказати, що поточні дослідження зосереджуються на розробці технік і методів для виявлення синтаксичних проблем у SQL-запитах. Однак, як ми показуємо в цій роботі, а також як вказують інші дослідження, більше уваги слід приділяти виявленню семантичних проблем у запитах. З імплікаціями, що охоплюють проблеми продуктивності та вразливості до безпеки, наявність цих типів семантичних помилок у будь-якій системі безумовно є шкідливою. Тому дуже важливо, що найближчим часом буде проведено більше досліджень, пов'язаних із виявленням семантичних проблем у SQL.

4.6 Висновок

Для виявлення семантичних помилок в SQL-запитах був розроблений інструмент статичного аналізу. Цей інструмент базується на 24 правилах, кожне з яких визначає стратегію виявлення конкретного типу семантичної помилки. Інструмент реалізований на Java, а для аналізу запитів SQL використовується бібліотека JSQLParser.

Основний потік роботи інструмента включає аналіз запиту за допомогою JSQLParser, після чого кожне правило виявлення, реалізоване як JSQLParserVisitor, застосовується до запиту. Кожного разу, коли правило порушується, запит позначається як проблемний, і користувач отримує попередження про семантичну помилку. Інструмент може працювати у трьох режимах: введення через командний рядок, аналіз файлу з SQL-запитами та підключення до бази MySQL для отримання та аналізу запитів.

Розробка інструменту здійснювалась з урахуванням принципів простоти розширення. Кожне правило виявлення реалізоване як окремий клас Java, що легко розширюється для визначення нових стратегій виявлення помилок. Всі

правила використовують загальний інтерфейс для роботи із запитами, що спрощує процес тестування та розширення.

Загалом, інструмент проходив обширне тестування, включаючи понад 150 модульних тестів для 24 правил, а також ручний аналіз понад 400 SQL-запитів. Результати тестування та аналізу підтверджують ефективність інструменту виявлення семантичних помилок у SQL-коді та його здатність до розширення для додавання нових правил в майбутньому.

ВИСНОВОК

SQL є найбільш широко використовуваною мовою баз даних, її активно використовують понад 70% розробників, як зазначалося в попередньому дослідженні [38], проте відсутня підтримка виявлення семантичних проблем, у запитах SQL. Це особливо непокоїть, враховуючи високу популярність цієї мови та її широке поширення у багатьох галузях, тому вплив потенційних проблем із такими запитами неможливо переоцінити.

Попередні дослідження, проведені Мітровичем, Байдером, Роджерсом, Тайпалусом і Сепьонен, фокусувалися на створенні інтелектуальних навчальних систем для подолання труднощів, з якими стикаються студенти при вивченні SQL. Представлене дослідження вирізняється тим, що воно використовує інструмент виявлення семантичних помилок на основі правил, який працює з усіма можливими операторами SQL, що розширює можливості навчання та дозволяє студентам краще розуміти не лише синтаксис SQL, але й його семантику.

По-перше, ми пропонуємо набір із 24 перевірених евристик для виявлення найпоширеніших типів семантичних помилок, які з'являються в SQL-запитах, на основі даних попередніх досліджень. По-друге, ми проводимо емпіричне дослідження поширеності семантичних помилок у SQL, використовуючи два набори даних із запитами, витягнутими із трьох проектів з відкритим вихідним кодом. Правильність евристики перевірялася вручну за випадковою вибіркою із понад 400 запитів.

Додатково, дослідження Тайпалуса, Сепьонен та Перьоло вказує на необхідність подальших досліджень з використання реальних проектів із запитами SQL. Це може допомогти з'ясувати, наскільки результати попередніх досліджень можуть бути застосовані до реальних програмних проектів та завдань, що виникають у практичних сценаріях.

У нашому дослідженні був проведений великомасштабний аналіз синтаксичних та семантичних помилок у SQL-запитах, які формулюють студенти та інші початківці. Синтаксичний аналіз показав, що студенти найчастіше

допускають помилки в операторах SELECT, такі як помилки в області WHERE, використання невизначених посилань і пропуск стовпців. Ці помилки є загальними та можуть бути важко виявити через відсутність систематичного підходу до формулювання запитів.

Семантичний аналіз розкрив, що багато семантичних помилок пов'язані з вибором неправильного типу запиту та відсутністю систематичності в формулюванні запитів. Це свідчить про те, що студенти можуть мати недостатнє розуміння основних принципів SQL. Ці результати підкреслюють важливість навчання студентів систематичному підходу до формулювання запитів та надання їм знань про правильний вибір типу запиту у конкретних ситуаціях.

Дослідження помилок у SQL-коді, яке вивчало різноманітні аспекти цих помилок у відкритих проектах з використанням популярних API-інтерфейсів, показало, що ці помилки виникають навіть у реальних програмних проектах. Це підкреслює важливість усунення їх на ранніх етапах розробки. Результати цього дослідження свідчать про те, що помилки, такі як використання оператора SELECT ALL та неправильна обробка значень NULL, є серйозними проблемами в SQL-коді, які можуть впливати на продуктивність і надійність програмного забезпечення.

У розділі про запропонований в роботі інструмент статичного аналізу виявлення семантичних помилок у SQL-запитах було розглянуто ряд евристик. Ці евристики були використані для створення набору правил у представленому інструменті статичного аналізу. Помилки, виявлені за допомогою цих евристик, взяті з повного списку семантичних помилок, що був скомпільований у роботі Брасса та Голдберга. Наш інструмент має перевагу у гнучкості та можливості налаштування порівняно із закритими аналогами, що дозволяє йому працювати ефективно при обмежених знаннях про конкретні запити.

У наступних розділах розглянуті конкретні семантичні помилки разом із прикладами SQL-запитів, що містять ці помилки, та надана реалізація кожної стратегії виявлення, включаючи відповідні евристики. Цей підхід не лише дозволяє ідентифікувати семантичні помилки, але й надає розробникам засоби для

їх виправлення та покращення якості SQL-коду. Запропонований інструмент створює можливість для автоматичного виявлення семантичних помилок в SQL-запитах, що може бути корисним для розробників у підтримці та вдосконаленні їхніх баз даних. Подальші дослідження можуть розширити представлену роботу, розглядаючи інші типи запитів та реалізуючи більше правил для виявлення семантичних помилок у різних сценаріях використання.

Існує великий клас запитів SQL, які є синтаксично правильними, але, тим не менш, точно не призначені, незалежно від того, яке завдання запиту може бути. Можна очікувати, що хороша СУБД виведе попередження для таких запитів, але, наскільки нам відомо, жодна СУБД цього не робить.

Перелік типів помилок, що міститься в цьому документі, може служити специфікацією завдання цього інструменту.

Перелік джерел посилання

1. Alireza Ahadi, Vahid Behbood, Arto Vihavainen, Julia Prior, and Raymond Lister. Students' syntactic mistakes in writing seven different types of sql queries and its application to predicting students' success. In Proceedings of the 47th ACM Technical Symposium on Computing Science Education, pages 401–406, 2016.

2. Alireza Ahadi, Julia Prior, Vahid Behbood, and Raymond Lister. Students' semantic mistakes in writing seven different types of sql queries. In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, pages 272–277, 2016.

3. Miltiadis Allamanis and Charles Sutton. Why, when, and what: analyzing stack overflow questions by topic, type, and code. In 2013 10th Working Conference on Mining Software Repositories (MSR), pages 53–56. IEEE, 2013.

4. Sebastian Baltes, Christoph Treude, and Stephan Diehl. Sotorrent: Studying the origin, evolution, and usage of stack overflow code snippets. In 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), pages 191–194. IEEE, 2019.

5. Anton Barua, Stephen W Thomas, and Ahmed E Hassan. What are developers talking about? an analysis of topics and trends in stack overflow. Empirical Software Engineering, 19(3):619–654, 2014.

6. Stefanie Beyer and Martin Pinzger. A manual categorization of android app development issues on stack overflow. In 2014 IEEE International Conference on Software Maintenance and Evolution, pages 531–535. IEEE, 2014

7. Ilia Bider and David Rogers. Yasqlt–yet another sql tutor. In International Conference on Conceptual Modeling, pages 197–206. Springer, 2016.

8. Stefan Brass and Christian Goldberg. Detecting logical errors in sql queries. In Grundlagen von Datenbanken, pages 28–32. Citeseer, 2004.

9. Stefan Brass and Christian Goldberg. Semantic errors in sql queries: A quite complete list. Journal of Systems and Software, 79(5):630–644, 2006.

10. Stefan Brass, Christian Goldberg, and Alexander Hinneburg. Detecting semantic errors in sql queries. Technical report, Technical Report, University of Halle, 2003

11. Jeroen Castelein, Maurício Aniche, Mozhan Soltani, Annibale Panichella, and Arie van Deursen. Search-based test data generation for sql queries. In Proceedings of the 40th international conference on software engineering, pages 1220–1230, 2018

12. E. F. Codd. A relational model of data for large shared data banks. Commun. ACM, 13(6):377–387, June 1970. ISSN 0001-0782. doi: 10.1145/362384.362685

13. Antonio Cuevas, Manuel Febrero, and Ricardo Fraiman. An anova test for functional data. Computational statistics & data analysis, 47(1):111–122, 2004.

14. Jens Dietrich, Markus Luczak-Roesch, and Elroy Dalefield. Man vs machine–a study into language identification of stack overflow code snippets. In 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), pages 205–209. IEEE, 2019.

15. Nees Jan van Eck and Ludo Waltman. How to normalize cooccurrence data? an analysis of some well-known similarity measures. Journal of the American society for information science and technology, 60(8):1635–1651, 2009.

16. SQL Enlight. E0178: Like operator is used without wildcards. SQL Enlight, Retrieved April, 2021

17. SQL Enlight. E0007: Pattern starting with “%” in like predicate. SQL Enlight, Retrieved April,

18. SQL Enlight. E0102: Do not use distinct keyword in aggregate functions. SQL Enlight, Retrieved April, 2021.

19. Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. Stack overflow considered harmful? the impact of copy&paste on android application security. In 2017 IEEE Symposium on Security and Privacy (SP), pages 121–136. IEEE, 2017.

20. Christian Goldberg. Do you know sql? about semantic errors in database queries. In 7th Workshop on Teaching, Learning and Assessment in Databases, Birmingham, UK, HEA. Citeseer, 2009

21. Carl Gould, Zhendong Su, and Premkumar Devanbu. Jdbc checker: A static analysis tool for sql/jdbc applications. In Proceedings. 26th International Conference on Software Engineering, pages 697–698. IEEE, 2004.

22. Huzefa Kagdi, Michael L Collard, and Jonathan I Maletic. An approach to mining call-usage patternswith syntactic context. In Proceedings of the twentysecond IEEE/ACM international conference on Automated software engineering, pages 457–460, 2007.

23. Bill Karwin. SQL antipatterns: avoiding the pitfalls of database programming. Pragmatic Bookshelf, 2010

24. Shaheen Khatoon, Guohui Li, and Azhar Mahmood. Comparison and evaluation of source code mining tools and techniques: A qualitative approach. Intelligent Data Analysis, 17(3):459–484, 2013.

25. Yash Lamba, Manisha Khattar, and Ashish Sureka. Pravaaha: Mining android applications for discovering api call usage patterns and trends. In Proceedings of the 8th India Software Engineering Conference, pages 10–19, 2015

26. Mario Linares-Vásquez, Bogdan Dit, and Denys Poshyvanyk. An exploratory analysis of mobile development issues using stack overflow. In 2013 10th Working Conference on Mining Software Repositories (MSR), pages 93–96. IEEE, 2013.

27. MITRE. Weaknesses in software written in C. MITRE, Retrieved April, 2021

28. MITRE. CWE Top 25 most dangerous software errors. MITRE, Retrieved April, 2021.

29. Antonija Mitrovic. A knowledge-based teaching system for sql. In Proceedings of ED-MEDIA, volume 98, pages 1027–1032, 1998

30. Antonija Mitrovic. Learning sql with a computerized tutor. In Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education, pages 307–311, 1998

31. Antonija Mitrovic. An intelligent sql tutor on the web. International Journal of Artificial Intelligence in Education, 13(2-4):173–197, 2003

32. Leon Moonen. Generating robust parsers using island grammars. In Proceedings Eighth Working Conference on Reverse Engineering, pages 13–22. IEEE, 2001.

33. Tamara Munzner. Visualization analysis and design. CRC press, 2014.

34. Biruk Asmare Muse, Mohammad Masudur Rahman, Csaba Nagy, Anthony Cleve, Foutse Khomh, and Giuliano Antoniol. On the prevalence, impact, and evolution of sql code smells in data-intensive systems. In Proceedings of the 17th International Conference on Mining Software Repositories, pages 327–338, 2020.

35. Csaba Nagy and Anthony Cleve. Mining stack overflow for discovering error patterns in sql queries. In 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 516–520. IEEE, 2015.

36. Csaba Nagy and Anthony Cleve. A static code smell detector for sql queries embedded in java code. In 2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM), pages 147–152. IEEE, 2017

37. Csaba Nagy and Anthony Cleve. Sqlinspect: A static analyzer to inspect database usage in java applications. In Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, pages 93–96, 2018.

38. Luca Ponzanelli, Andrea Mocci, and Michele Lanza. Stormed: Stack overflow ready made data. In 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, pages 474–477. IEEE, 2015.

39. Toni Taipalus. Explaining causes behind sql query formulation errors. In 2020 IEEE Frontiers in Education Conference (FIE), pages 1–9. IEEE, 2020.

40. Toni Taipalus and Piia Perälä. What to expect and what to focus on in sql query teaching. In Proceedings of the 50th ACM Technical Symposium on Computer Science Education, pages 198–203, 2019.

41. Toni Taipalus and Ville Seppänen. Sql education: A systematic mapping study and future research agenda. ACM Transactions on Computing Education (TOCE), 20(3):1–33, 2020.

42. Toni Taipalus, Mikko Siponen, and Tero Vartiainen. Errors and complications in sql query formulation. ACM Transactions on Computing Education (TOCE), 18(3):1–29, 2018.

43. Tao Xie and Jian Pei. Mapo: Mining api usages from open source repositories. In Proceedings of the 2006 international workshop on Mining software repositories, pages 54–57, 2006.

44. Haoxiang Zhang, Shaowei Wang, Heng Li, Tse-Hsun Peter Chen, and Ahmed E Hassan. A study of C/C++ code weaknesses on stack overflow. IEEE Transactions on Software Engineering, 2021.

45. Yuhao Wu, Shaowei Wang, Cor-Paul Bezemer, and Katsuro Inoue. How do developers utilize source code from stack overflow? Empirical Software Engineering, 24(2):637–673, 2019.

46. Chun-Nan Hsu and Craig A. Knoblock. Using inductive learning to generate rules forsemantic query optimization. In Advances in Knowledge Discovery and Data Mining, pages 425–445. AAAI/MIT Press, 1996.

47. U. S. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic queryoptimization. ACM Transactions on Database Systems, 15:162–207, 1990.

48. C. Welty. Correcting user errors in SQL. International Journal of Man-Machine Studies 22:4, 463-477, 1985.

49. Terry Gaasterland, Parke Godfrey and Jack Minker. An Overview of Cooperative Answer-ing. Journal of Intelligent Information Systems 21:2, 123–157, 1992.

50. Wesley W. Chu, Hua Yang, Kuorong Chiang, Michael Minock, Gladys Chow and ChrisLarson. Cobase: A scalable and extensible cooperative information system. Journal ofIntelligent Information Systems, 1996.

ДОДАТОК А
(Обов’язковий)

РУЧНИЙ АНАЛІЗ ЗАПИТІВ

№	№ Правила	Запит	Запит вірний
1	0001	select a1.owner,a1.table_name from all_tab_columns a1 , all_tab_columns a2 where a1.owner=a2.owner and a1. table_name=a2.table_name and a1.column_name='location ' and a2.column_name='asset_id' order by a1.owner,a1. table_name;	+
2	0001	select t1.userid from userrole t1 join userrole t2 on t1 .userid = t2.userid and t2.roleid = 2 join userrole t3 on t2.userid = t3.userid and t3.roleid = 3 and t1.roleid = 1;	+
3	0001	select deptno from dept where exists(select * from emp x where x.deptno = 20 and exists(select * from emp y where y.job = x.job and y.deptno = dept.deptno)) and deptno <> 20;	+
4	0001	select count(distinct patients.id) from public. patients, public.subscriptions, public.users, public. calendar_days where patients.user_id = users.id and patients.id = calendar_days.patient_id and subscriptions .user_id = patients.user_id and (date_trunc('day', patients.last_sync) > current_date - inter...	+
5	0001	select * from tmp t0 where exists (select * from tmp t1 join tmp t2 on t2.clientid = t1.clientid and t2. serverid = t1.serverid and t2.logtime > t1.logtime where t1. status = 'aborted' and t2. status = 'failed' and t1 .clientid = t0.clientid and t1.serverid = t0.serverid and t1.logtime = t0.logtime o...	+
6	0001	select distinct t.name from hospital_a t, hospital_a v where t.op_date=v.date_of_birth and t.surname=v.surname and t.op_id = 'p619920' and v.op_id = 'i552015';	-
7	0001	select im1.imageid as id, im1.attributevalue as color, im2.attributevalue as quality from imageattribute im1, imageattribute im2 where im1.imageid = im2.imageid and im1.attributetype = "color" and im2.attributetype = " quality" and im2.attributevalue = "good";	+
8	0001	select t1.a3 from test t1, test t2 where t1.a1 = t2.a1 and t2.a2 = t1.a2 and t1.a1 = t2.a2;	+
9	0001	select * from audit a1 where not exists(select * from audit a2 where a2.status='approve' and a1.status=' decline' and a2.id=a1.id);	+
10	0001	select a.post_id, b.post_id, a.ul_value as "likes", b .ul_value as "dislikes" from wp_like_dislike_counters as	-

		a, wp_like_dislike_counters as b where a.post_id = b.post_id and a.ul_key = 'u_like' and b.ul_key = ' u_dislike';	
11	0001	update users set online = 1 where exists(select null from users t where t.email = in_email and t.password = in_password and t.id = id) and id = 'result_id';	+
12	0001	select p1.domain_id, p2.domain_id, count(p1.domain_id) as d1, count(p2.domain_id) as d2 from pdb as p1, interacting_pdbs as i1, pdb as p2, interacting_pdbs as i2 where p1.id = i1.pdb_first_id and p2.id = i2. pdb_second_id and i1.id = i2.id group by p1.domain_id, p2.domain_id having d1 > 100	+
13	0001	select a.object from mytable a, mytable b where a.object = b.object and a.key = 'a' and a.value = 'a' and b.key = 'b' and b.value = 'b';	+
14	0001	select * from cats outside where not exists(select * from cats cat where exists(select dog.foo,dog.bar from dogs dog where cat.foo = dog.foo and cat.bar = dog.bar) and outside.foo = cat.foo and outside.bar=cat.bar);	+
15	0001	select username.name, useremail.email, userphone.phone from users as username inner join users as useremail on username.user = useremail.user and username.field = 'name' and useremail.field = 'email' inner join users as userphone on username.user = userphone.user and userphone.field = 'phone';	-
16	0001	select count(*) from (select u1.userid from vote u1, vote u2 where u1.itemid = u2.itemid and u1.userid = user1 and u2.userid = user2);	+
17	0001	update mark set mark= case when mark.val<= 5 then val*1.1 else val end where mark.id_classes = classes. id_classes and classes.id_subject = subject.id_subject and subject.id_subject = 5;	+
18	0001	select a1.owner,a1.table_name from all_tab_columns a1 , all_tab_columns a2 where a1.owner=a2.owner and a1. table_name=a2.table_name and a1.column_name='location ' and a2.column_name='asset_id' order by a1.owner,a1. table_name;	+
19	0001	select p1.gameid from participants as p1, participants as p2 where p1.name = 'team1' and p2.name='team2' and p1 .gameid = p2.gameid;	-
20	0001	select n1.id, n1.name,n2.name, a3.age from name n1, name n2, age a3 where n1.id=n2.id and n1.id=a3.id and	+

		n1. type=0 and n2.type=1;	
21	0002	select * from mytable1 ca left outer join mytable2 dcn on dcn.dstrct_code = ca.dstrct_code where ca. dstrct_code = '0001' and ca.req_232_type = 'p' and ca .requisition_no = '264982 000' and ca.alloc_count = '01' order by ca.alloc_count asc;	+
22	0002	select * from ranked where ranking = 1;	+
23	0002	select * from orc_users as t1 inner join orc_files as t2 on t1.id = t2.userid where t1.email='sdfsdf';	+
24	0002	select * from person where name = 'robert';	+
25	0002	select * from myuser as u join friend_pivotal as p on u. id = p.user_rec_id or u.id = p.user_inv_id where status = 'accepted';	+
26	0002	select * from (select * from table order by value desc, date_column) where rownum = 1;	+
27	0002	select * from a_table where a_column = '&avariable ' union select * from a_table where b_column = '&& avariable';	+
28	0002	select incurredcharges.procedure_no, incurredcharges. patient_no, charges.procedure from incurredcharges inner join charges where incurredcharges.patient_no=12;	+
29	0002	select u.uid ,u.uname,p.uid , p.profname,e.uid,e.eduname from user u inner join profession p on u.uid=p.pid inner join education e on u.uid = e.uid where u.uid=p. uid and u.uid=e.uid and i.uid=1;	+
30	0002	select arbogast.node.nid as anid, mcguffin.node.nid as mnid, arbogast.node.title as atitle, mcguffin.node. title as mtitle from arbogast.node, mcguffin.node where arbogast.node.nid = 1 and mcguffin.node.nid = arbogast. node.nid;	+
31	0002	select imagename, sum(ratingvalue) as "lol" from ratings group by imagename having imagename = 'myimagename';	+
32	0002	select max(id) as id, type, max(other_id) as other_id, max(def_id) as def_id, ref_def_id from t where type = ' ref' group by type, ref_def_id;	+
33	0002	select distinct name from sys.objects where type in ('u ','v') and name= 'myname';	+
34	0002	select customerid, orderdate, count(1) cnt from sales	+

		.salesorderheader where customerid = 11300 group by customerid, orderdate order by cnt desc;	
35	0002	select booking.bookno, booking.courno, course.coursename from booking, course, coursename where booking.bookno = 6200 and booking.courno = course.courno and coursename. coursenameno = course.coursenameno;	+
36	0002	select table1.personcode, table1.name, table2.location, max(table2.servicedate) from table1 inner join table2 on table1.id = table2.table1id where table1.personcode = 'xyz' group by table1.personcode,table1.name, table2. location;	+
37	0002	select t1.userid, t2.date, min(t1.time) as in_time, max(t1.time) as out_time from test t1 join (select distinct date from test where userid = 609) t2 where t1.date = t2 .date and userid = 609 group by t1.userid, t2.date;	+
38	0002	select * from mm_tfs where product_description like '% football%' and schoolid = '8' and category_id ='21';	+
39	0002	select * from customer where c_role = 'dev' order by c_id limit 2;	+
40	0002	select attrsmall, attrlarge, max(rating_id) as ratingmax from (select case when c.attr1_id < c.attr2_id then c .attr1_id else c.attr2_id end as attrsmall, case when c .attr1_id < c.attr2_id then c.attr2_id else c.attr1_id end as attrlarge, c.rating_id from compatibility c) as c1 group by atrrsmall	+
41	0003	insert into personpet(fk_person, fk_pet) select id, id from person;	+
42	0003	select identity(int) as tempid, *, sectionid as fix2ids from files_sections;	+
43	0003	select *, cond2 as cond2, cond3 as cond3 from table having cond1 and (cond2 or cond3);	+
44	0003	select country, count(*) as cnt1, count(*) as cnt2 from orders group by country having cnt1=2 and cnt2>2;	+
45	0003	select c77,c77,c125,c126,c127,c74 from mytable;	+
46	0003	select itemnumber as '@id', itemnumber as 'itemnumber', price as 'price/@value', datefrom as 'price/datefrom', dateto as 'price/dateto' from #tempxml;	+
47	0003	select videos.id, videos.id as video_id, videos. video_title as video_title, group_concat(distinct t. tag_name separator '|') as tag_names from videos join	+

		video_tags as vt join tags as t where videos.id <= (select max_doc_id from sph_counter where counter_id = 1) group by videos.id;	
48	0003	select distinct t.date_effective, t. acct_account_transaction_id, p.method, t.amount, c. business_name, t.amount from contact c join contact_role cr on cr.contact_fk = c.contact_id join acct_account a on a.contact_fk = c.contact_id	+
49	0003	insert into test_table(column_1, column_2) select val, val from x;	+
50	0003	select e.empid,e.fname,e.lname,c.description as hair,c. description as race from employee2 e inner join code c;	+
51	0003	select uuid, uuid from data;	+
52	0003	select value, value from public.customers where nodevalue.key3 = 'key3' and nodevalue.key4 = 'key4';	+
53	0003	select owner, count(distinct object_name || 'this is a really long string in my expression, don''t you think? actually, it''s really, r	+
54	0003	select projects.project_id, projects.title, projects. start_time, projects.description, projects.user_id, projects.winner_user_id, users.username as owner, users .username as winner from projects,users where projects. user_id=users.user_id and projects.winner_user_id=users. user_id;	+
55	0003	select productgroupid as product23_1_, articleid as articleid1_, articleid as articleid18_0_, inventory_name as inventory3_18_0_, inventory_unitofmeasure as inventory4_18_0_, businesskey as business5_18_0_, name as name18_0_, servespeople as servespe7_18_0_, instock as instock18_0	+
56	0003	select *, t1.image_id as image from event.dbo. dia_tracker t1 where exists (select 1 from event.dbo. dia_tracker t2 where t2.patient_id = 'dsma' group by case when t2.active = '1' then t2.image_id end having max(t2.image_id) = t1.image_id);	+
57	0003	select *, 'full' from abc;	+
58	0003	select 1 except select 1;	-
59	0003	select *, regexp_replace(port, '[0-9/.]', '', 'g') port_name, string_to_array(regexp_replace(port, '[a-zaz-]', '', 'g'), '/')::float[] port_number from device order	+

		by name, regexp_replace(port, '[0-9/.]', '', 'g'), string_to_array(regexp_replace(port, '[a-za-z-]', '', ' g'), '/')::float[];	
60	0003	select *, match(pages) against('doodle') as score from books where match(pages) against('doodle') order by score desc;	+
61	0004	select emp.salary from employee emp inner join sap s on emp.id = s.id where s.id = 111;	+
62	0004	select * from child c,parent p where c.id=p.parentid and c.parentid !=0	+
63	0004	select t1.col_1 from table t1 join table t2 on t1.col_2= t2.col_1 where t1.col_1=t2.col_2 and t1.col_1='one';	+
64	0004	select m.actor from movies m where m.movie = 'pulp fiction' and not exists (select 1 from movies m1 join movies m2 on m1.movie = m2.movie and m1.actor <> m2. actor and m2.movie <> 'pulp fiction' and m2.actor in (select actor from movies where movie = 'pulp fiction') where m.actor = m1.actor);	+
65	0004	select c.name, if(find_in_set('type_a', group_concat (substring_index(f1.filename,'_',2))), 'yes', 'no') as type_a, if(find_in_set('type_b', group_concat(substring_index(f1.filename,'_',2))), 'yes', 'no') as type_b, if(find_in_set('type_c', group_concat(substring_index(f1.filename,'_',2))), 'yes'	+
66	0004	select u.firstname, u.lastname, u.rep, u.email, u. password, u.gender, u.level, u.birthday, u.achievements, u.height, u.unit, u.cityid, u.countryid, r.regdate, ci. name as city, co.name as country from users u, registry r, cities ci, countries co where u.id = 1 and r.uid = u. id and u.cityid = ci.id	+
67	0004	select images.*, users.username from images left join users on images.user_id = users.id left join user_follow on images.user_id = user_follow.follow_id where images. user_id = 3 or user_follow.user_id = 3 order by images. date desc;	+
68	0004	select * from tblpe t1 where t1.date = (select max(date) from tblpe t2 where t1.id = t2.id) and t1.id = 39;	+
69	0004	select pd.id,pd.price_date,pd.name_id,pd.class_id,pd. currency_id,pd.price, pd.price - (select price from price_data as x where x.price_date < pd.price_date and x .name_id = pd.name_id and x.class_id = pd.class_id and x .currency_id = pd.currency_id having max(x.price_date)) as `change`	+

70	0004	select activitytext, actiontext from activity join activityaction on activity.activityid = activityaction .activityid join action on activityaction.actionid = action.actionid where activity.activityid = 1;	+
71	0004	select studentid, firstname, lastname, gender from student join major on student.majorid = major.majorid where major.majorid = 2;	+
72	0004	select col.column_name, col.constraint_name from information_schema.constraint_column_usage col where col.constraint_name = tab.constraint_name and col. table_name = tab.table_name and constraint_type = ' primary key' and col.constraint_name like '%adhoc%';	+
73	0004	update purchase as p inner join artwork as a on p. purchaseid = a.purchaseid set p.total = sum(a.price) where a.purchaseid = 'd4758';	+
74	0004	select client.*, cat1id.client_catid_1 as cat1, cat2id. client_catid_2 as cat2 from tb_clients as client left join tb_clients_categories cat1id on client.client_id = cat1id.client_id left join tb_clients_categories cat2id on client.client_id = cat2id.client_id where client. client_id = 65447;	+
75	0004	select c.c_id, s.s_fname, count(r.c_id) from results r, candidates c, student s,positioning p, organization o where r.c_id = c.c_id and c.sid = s.sid and c.pos_id = p .pos_id and o.org_id = c.org_id and o.org_id = 1 group by c.c_id;	+
76	0004	select pt.projecttaskid, isnull(sum(case when pte. assigneduserid = @userid then 1 end),0) as assignedtome, isnull(sum(case when pte.assigneduserid <> @userid then 1 end),0) as assignedtoothers from projecttask pt inner join project p on p.projectid = pt.projectid left join projecttaskentity pte on p	+
77	0004	select ip_settings.ip from server inner join network_interfaces on network_interfaces.id = server .fk_network_interfaces inner join ip_settings on ip_settings.id = network_interfaces.fk_ip_settings where server.id=6;	+
78	0004	update table_to tt, table_from tf set tt.name = "chandi" where tt.id = tf.id and tf.id = 1;	+
79	0004	select t.* from terms t join term_relationships tr on tr.term_id = t.term_id join posts p on p.post_id = tr. post_id where p.post_id = 1;	+

80	0004	select t1.service from table1 t1, table2 t2 where t1. cnty = t2.cnty and t1.zip = t2.zip and t2.zip = '1234';	+
81	0005	select * from (values (1), (2)) as tbl(col) where not (col = null or col = 1);	+
82	0005	select r.id, r.authtoken.instagram,r.username from root r where r.abc <> null;	+
83	0005	select loans.*, if(ul.name = null, ub.name, ul.name) as name, if(ul.id = null, ub.id,ul.id) as uid from loans left join users ul on users.id = loans.lender_id left join users ub on users.id = loans.borrower_id where ul. id = 2 or ub.id = 2;	+
84	0005	select count(*) from table1 where request_time < timestamp'2012-05-19 14:00:00' and (end_time > timestamp '2012-05-19 14:00:00' or end_time=null);	+
85	0005	select * from my_table where some_field = null;	+
86	0005	select * from customer c where so.orderid = null;	+
87	0005	select name from bbc where gdp > all (select gdp from bbc where region = 'africa' and gdp<>null);	+
88	0005	select * from hotel where address != null;	+
89	0005	select * from t1 where a=b or (a=null and b=null);	+
90	0005	select concat(area, yearlevel, code) as subjectcode, count(student) from studenttakessubject where result < 50 and result <> null group by code having count(student) > 1;	+
91	0005	select checklists.id from checklists left join (select checklist_id from checklist_items where completed = 'f' union distinct select checklist_id from checklist_items group by checklist_id having count (*) = 0) partial_checklists where partial_checklists. checklist_id = null;	+
92	0005	select lefttable.id, righttable.id as nullid from lefttable left join righttable on righttable.id = lefttable.id where nullid = null;	+
93	0005	select e.id from employments e where not exists (select 1 from emails em where em.employment_id=e.id and em. deactivated_on = null)	+
94	0005	select secid, min(date) as startdate, max(date) as enddate, price from bigtable group by secid, enddate having min(date) != max(date) and date != null;	+
95	0005	select * from student left join course where student. std_id = null or student.std_name = null or student.	+

		std_start_date = null or student.std_end_date = null or student.std_gender = null or student.course_id = null;	
96	0005	select top (10) ctry, sales from table1 union all select 'other', sum(sales) from table1 where table2.ctry = null group by table1.ctry;	+
97	0005	update salesorders o set transref = (select t.transref from transfers t where o.orderno = t.orderno and (o. ordext = t.ordext or (t.ordext=null and o.ordext=null)) and t.transref <> null) where ordext = null;	+
98	0005	select h.clientnumber, iif(h.checkoutdate=null,"yes ","") as currentvisitor from visitstable as h inner join (select clientnumber, max(lastvisitdate) as lastvisitstart from visitstable group by clientnumber) as t;	+
99	0005	select * from t left join m on t.sub_id = m.id where t. sub_id != null;	+
100	0005	select m.provider_id, m.provider_name, p. purchase_order_code, null as purchase_order_sample_code, p.provider_id as provider_order from mst_provider as m left join trx_purchase_order as p where p.provider_id != null union select m.provider_id, m.provider_name, p. purchase_order_code, null as purchase	+
101	0006	select sum(amt) where session=x and order >= (select max(order) where atype='set' and session=x);	+
102	0006	select t.*, row_number() over(partition by "name" order by "date") as rank from tablea t where "date" >= (select max("date") from tablea where "type" = 'b')	+
103	0006	select * from `table_name` where id >= (select floor(max(id) * rand()) from `table_name`) order by id limit 30;	+
104	0006	select id from table_name where id >= (select max(id) from table_name where id <= 12);	+
105	0006	select thedate from datetable where thedate >= (select max(thedate) from datetable where thedate < getdate());	+
106	0006	select distinct maker, price from product inner join printer where color = 'y' and price <= (select min(price) from printer where color = 'y');	+
107	0006	select * from numbers where nr >= (select max(nr) from numbers where nr < 6.20) and nr <= (select min(nr) from numbers where nr > 6.20);	+

108	0006	select name, message from flux_chat_messages where id >= (max(id) -5) order by id asc;	+
109	0006	select submissionid, input, value from yourtable where submissionid >= (select max(submissionid) from yourtable) - 1 order by submissionid;	+
110	0006	select * from yourtable where id >= 4 and id <= (select min(id) from yourtable where b = 'f' and id >= 4);	+
111	0006	select * from my_table where datetime_column >= date_sub ((select max(datetime_column) from my_table), 7);	+
112	0006	select distinct d.phonenum,d.sourcetable,n.fullname ,c.fk_applicationid as ref,t.subject,t.createddate from dial d join database.dbo.dm_phonenumbers p on p. phonenum1 = d.phonenum collate latin1_general_ci_as join database.dbo.dm_phonenumbers on p.phonenum2 = d. phonenum collate latin1_general_ci_as j	+
113	0006	select * from yourtable where id >= 4 and id <= coalesce ((select min(id) from yourtable where b = 'f' and id >= 4), (select max(id) from yourtable));	+
114	0006	select * from users where id >= (select floor((max(id) - min(id) - 1) * rand()) + min(id) from users) limit 1;	+
115	0006	select foo from bar where id >= (abs(random()) % (select max(id) from bar)) limit 1;	+
116	0006	select ename, job, sal from emp where sal >=(select max (sal) from emp where sal < (select max(sal) from emp where sal < (select max(sal) from emp))) order by sal;	+
117	0006	select foo from bar where _rowid_ >= (abs(random()) % (select max(_rowid_) from bar)) limit 1;	+
118	0006	select * from customers where cno = (select cno from (select count(*) as ordcount, cno from orders group by cno having ordcount >= (select max(ordcount) from (select count(*) as ordcount, cno from orders group by cno))));	+
119	0006	select * from table where id >= (select max(id) from table) - 10 order by id desc;	+
120	0006	select name from world where gdp >= (select max(gdp) from world where continent = 'europe');	+
121	0007	update run inner join snp_hgvs set run.comment=concat(' rs',snp_hgvs.snp_id) where run.compute not like 'tron';	+

122	0007	insert into cesc_pf_stmt_ext_wrk(pf_emp_code , pf_dept_code , pf_sec_code , pf_prol_no , pf_fm_seq , pf_seq_no , pf_sep_tag , pf_source) select pfl_emp_code , pfl_dept_code , pfl_sec , pfl_prol_no , pf_fm_seq , pf_seq_no , pfl_sep_tag	+
123	0007	select id into id_historical from historical where volume_id = id_volume and (action like 'expedicao' or action like 'conference');	+
124	0007	select * from first_test_name where firstname like 'bob ';	+
125	0007	select * from table where fiels1 not like 'x' and field2 not like 'y';	+
126	0007	select name from memberdb where name like 'lim %' or name like '% lim' or name like '% lim %' or name like ' lim';	+
127	0007	select forum.user.userid, forum.user.usergroupid, forum .user.password, forum.user.salt, forum.user.pmunread, forum.subscriptionlog.expirydate from forum.user inner join forum.subscriptionlog where forum.user.username like 'someuser';	+
128	0007	select column_name from information_schema.columns where table_name='lime_survey_98673' and column_name like '98673';	+
129	0007	select * from emp where ename like 'king';	+
130	0007	select *, count(`event_target`) `totalsum` from `testdb ` where `account_id` = ? and (`event_target` like ' searchquery' or `event_title` like 'searchquery' or ` event_name` like 'searchquery') group by `username`, ` event_title`, `event_target` order by `totalsum` desc;	+
131	0007	select * from your_name where category_id not like '90';	+
132	0007	insert into table1(user_uuid, login_id) select users_uuid, '1234' from table2 where first_name like 'ortal';	+
133	0007	select * from products where p.name not like 'brand1%' and p.name not like 'specific product';	+
134	0007	select busname, email, render_pic, area,logo, url, email, map, description, tag, catch_phrase, region from results where style like 'varstyle' and region like ' varregion' and bedrooms like 'varbedrooms' and bathrooms like	+

		'varbathrooms' and price between varminprice and varmaxprice order by id desc	
135	0007	select c.id from `tags` `t` left join comictags as ct left join comics as c where ((t.tag like 'tag1') or (t. tag like 'tag2')) group by c.id;	+
136	0007	select description from tproduct where description like 'diamond';	+
137	0007	select distinct motor from general g where g.year between 1998 and 2004 and g.motor like '4 cil 1.8 lts';	+
138	0007	select * from administrators where username like ' thierry';	+
139	0007	select * from all_updatable_columns where column_name like 'reqd col name';	+
140	0007	select 'inactive' "status",nvl(region,'total') region, count(region) regcount from temppivottest where status like 'inactive' group by rollup (region);	+
141	0008	select paramone, paramtwo from tablename where search_param = 'x' union all select null, null from dual where not exists (select paramone, paramtwo from tablename where search_param = 'x');	+
142	0008	insert into a (vala1, vala2, vala3, vala4) select valb1 , valb2, valb3, valb4 from b where not exists(select vala1, vala2, vala3, vala4 from a where a.vala1 = b. valb1 and a.vala2 = b.valb2 and a.vala3 = b.valb3 and a. vala4 = b.valb4);	+
143	0008	select match_ref, count(match_ref) from raw_mwbe where exists(select match_ref, wbe from raw_mwbe where wbe like "p" or wbe like "n") group by match_ref, wbe having count(match_ref)>1;	+
144	0008	select id, type, number from roads where number = '12' union select id, type, number from roads where number like '12%' and not exists (select id, type, number from roads where number = '12');	+
145	0008	select l.email as email, l.date as date, l.ip as user_ip , l.logid as id from table as t where exists(select email,max(date)date from table group by email) and (ip is not null) group by t.email;	+
146	0008	select * from address a where not exists (select country,state union select 'us' as country, 'la' as state union select 'ind' as country, 'del' as state where e.country != a.country and e.state != a.state);	+

147	0008	select * from cats outside where not exists(select * from cats cat where exists(select dog.foo,dog.bar from dogs dog where cat.foo = dog.foo and cat.bar = dog.bar) and outside.foo = cat.foo and outside.bar=cat.bar);	+
148	0008	select a.id,a.name,split.item,a.flag from source_excel a where not exists (select a.id,split.item from source_excel a join source_dw b);	+
149	0008	delete from mycard t1 where t1.idmoney = 5 and t1.idcard = 80 and exists (select idcard, year, money from mycard t2 where t2.idcard = t1.idcard and t2.year = t1. year and t2.money = t1.money group by t2.idcard, t2.year , t2.money having count(t2.idcard) > 1) and t1.id not in (select min(id) from mycartd)	+
150	0008	select users.userid, year(membership.memyear) as memyear , users.mailto, users.streetaddress, users.address2, users.city, statelookup.state, users.zip from users inner join membership inner join statelookup where year (membership.memyear) = '2013' and not exists (select x. userid, year(membership.memyear)	+
151	0008	select sgtins_tmp_table.epc, sgtins.store from sgtins_tmp_table, sgtins where exists (select organization_id, sgtin from sgtins where client_id = 4 and sgtins.sgtin = sgtins_tmp_table.sgtin) and sgtins. client_id = 4 and sgtins_tmp_table.sgtin = sgtins.sgtin;	+
152	0008	select r12.id, r23.id, r13.id from relations r12 join relations r23 join relations r13 where not exists (select relation1_id, relation2_id, relation3_id from triangles where relation1_id = r12.id and relation2_id = r23.id and relation3_id = r13.id);	+
153	0008	select item, size from t1 where t1.date = '2013-02-11' and not exists (select item, size from t1 where t1.date = '2013-02-13');	+
154	0008	update pm set pm.amt = newvalue where exists (select t. acct, t.amt from pm t where pm.acct = t.acct group by t. acct, t.amt having count(t.acct)>1);	+
155	0008	insert into tablename (col1,col2) select @par1, @par2 where not exists (select col1,col2 from tablename where col1=@par1 and col2=@par2);	+
156	0008	select id,name from tbl_bkp t1 where not exists (select id,name from tbl_namecode where id=t1.id and name=t1. name);	+
157	0008	select i.agent , i.agency , i.customer , i.company from	+

		table_1 as i where not exists (select p.agent , p. agency , p.customer , p.company from table_2 as p where i.agent = p.agent and i.agency = p.agency and i.customer = p.customer and i.company = p.company);	
158	0008	select t1.id, avg(t1.score) avg1 from foo t1 group by t1 .id having not exists (select t2.id, avg(t2.score) avg2 from foo t2 group by t2.id having avg(t2.score) > avg(t1.score));	+
159	0008	select valb1, valb2, valb3, valb4 from b where not exists(select vala1, vala2, vala3, vala4 from a where a .vala1 = b.valb1 and a.vala2 = b.valb2 and a.vala3 = b. valb3 and a.vala4 = b.valb4);	+
160	0008	select a.number, a.c1, a.c2, a.c3 from table a where exists (select count(1), b.number from table b where b. number = a.number group by b.number having count(1) = 1)	+
161	0009	select * from user where name like '%mike%' or color = ' blue';	+
162	0009	select sc.id, sc.number, sc.name from tempdb..syscolumns sc inner join tempdb..sysobjects so on sc.id = so.id where so.name like '%mytable%';	+
163	0009	update person set addr = left(addr, len(addr) - 2) + ' street' where addr like '% st';	+
164	0009	select * from emp where name like '%a%e%';	+
165	0009	select columnname from tablename where columnname not like '%stack%';	+
166	0009	select * from street where street_name like '%park%ave %10%';	+
167	0009	select * from `words` where `word` like '%person%';	+
168	0009	select * from fiberbox where field like '1740 %' or field like '%1938 ' or field like '%1940 % test';	+
169	0009	select * from parameters where name like '%n%'	+
170	0009	select * from tablename where column like '%company%'	+
171	0009	select name from table where age like '%5%';	+
172	0009	select a.s_oid, a.s_id, a.area_acre, a.power_peak, a. nearby_city, a.solar_total from global_site a cross join na_utility_line b where (a.power_peak between 1.0 and 100.0) and a.area_acre >= 500 and a.solar_avg >= 5.0 and a.pc_num <= 1000 and (a.fips_level1 = '06' and a.	+

		fips_country = 'us' and a.fips_level1)	
173	0009	select * from table_name where name like '%word1%' and name like '%word2%'	+
174	0009	select m.* from dbo.mytable m where acolumn like '%val' order by (case when acolumn like '[a-z]%' then 0 else 1 end), acolumn;	+
175	0009	select distinct city from station where city like '%a' or city like '%e' or city like '%i' or city like '%o' or city like '%u';	+
176	0009	select * from yourtable where field like '%a%' or field like '%b%' or field like '%c%';	+
177	0009	select * from table1 where table1.id like '%1234';	+
178	0009	update table set col = substring(col,1,charindex('like', col,0)-1) where column like '%like%';	+
179	0009	select movie.id, movie.title, alsokownas.aka from movie left join alsoknowas on movie.id = alsoknowas.movieid where title like '%searchterm%';	+
180	0009	select a.noteid, a.firstname, a.lastname, a.status, a. category, a.name, a.followupdate from (select n.noteid , t.firstname, t.lastname, s.name as status, tc.name as category, d.name, n.followupdate as followupdate, row_number() over (partition by t.firstname, t.lastname order by n.noteid desc) rnk	+
181	0010	select `utm source`, `users` , count(*) from (select distinct `utm source`, `company id`, sum(distinct ` active users`) as users from customers group by `utm source`, `company id`) customers_2 group by `utm source `, `users`;	+
182	0010	select sum(distinct money) as sum_money, m.created_at from receipt r join material m group by m.created_at;	+
183	0010	select itemid from stock group by itemid having sum(distinct warehouseid) = (select sum(warehouseid) from warehouse)	+
184	0010	select department, sum(distinct price) from products join ratings on product_id=products.id where rating=5 group by department;	+
185	0010	select sum(distinct case when a.gender like 'm%' then 1 else null end) as males;	+
186	0010	select sum(distinct value) as total from table;	+

187	0010	select col1, sum(distinct col2) as s from tbl1 where col1='abbc' group by col1 order by s asc;	+
188	0010	select col1, sum(distinct col2) as s from tbl1 where col1='abbc' group by col1 order by s asc;	+
189	0010	select product_type, segment_type, sum(distinct promotion_value)promotion_value from sample_data group by product_type, cube(segment_type) order by product_type;	+
190	0010	select p.color, array_agg(distinct pg.group_id) as groups, sum(distinct installs) as installs from performance p join performance_groups pg group by color;	+
191	0010	select avg(distinct event_count), min(event_count), max (event_count) from numstest a join (select patient_id , count(*) as event_count from numstest group by patient_id) b;	+
192	0010	select sum(distinct money_spent.value), sum(distinct money_earned.value) from user join money_spent on money_spent.userid = user.userid join money_earned on money_earned.userid = user.userid;	+
193	0010	select sum(distinct cast(ar_all_bills.a_unpaid_balance as decimal(5,2))) as "total unpaid balance" from ar_all_bills where a_ar_customer_cid = 100059 group by a_ar_customer_cid;	+
194	0010	select blocks.user_id, sum(distinct payout_history. amount) as amount from blocks left join payout_history where confirms > 520 group by blocks.user_id;	+
195	0010	select `utm source`, `users` from (select distinct `utm source`, `company id`, sum(distinct `active users`) as users from customers group by `utm source`, `company id `) customers_2;	+
196	0010	select work.workid, work.description, machine. machinedescription, name.name, work2.regmin, work. minutes, (select sum(distinct minutes) from work w where w.machineid = machine.machineid) total_minutes from work work join machine machine left join work2 work2 left join name name;	+
197	0010	select `utm source`, `users` from (select distinct `utm source`, `company id`, sum(distinct `active users`) as users from customers group by `utm source`, `company id `) as customers_2 group by `utm source`;	+
198	0010	select a.user_id , count(distinct b.client_id) count_client_id, sum(distinct c.client_price)	+

		sum_client_price, max(b.act_date) max_act_date from tbl_user a inner join tbl_sactivity b inner join tbl_client c group by a.user_id;	
199	0010	select `utm source`, sum(`users`) from (select distinct `utm source`, `company id`, sum(distinct `active users `) as users from customers group by `utm source`, ` company id`) customers_2 group by `utm source`;	+
200	0010	select id_a, id_b, nu_b, date_trunc('month', dt_date) :: date as dt_month, sum(distinct (1 << (date_part('day', dt_date)::int) -1)) as nu_bit_days from test group by id_a, id_b, nu_b, date_trunc('month', dt_date) :: date ;	+
201	0011	select analytics.source as referrer, count(analytics.id) as frequency, sum(if(transactions.status = 'completed', 1, 0)) as sales from analytics left join transactions on analytics.id = transactions.analytics where analytics .user_id = 52094 group by analytics.source order by frequency desc limit 10;	+
202	0011	select count(order_header_id) from order_lines where sum (accounting_total) <= 500;	+
203	0011	select a.source as referrer, count(a.id) as frequency , sum(t.sales) as sales from (select id, source from analytics where user_id = 52094) a left join (select analytics, case when status = 'completed' then 1 else 0 end as sales from transactions) t on a.id = t.analytics group by a.source order by frequency	+
204	0011	select source, count(id) as frequency from analytics where user_id = 52094 group by source order by frequency desc limit 10;	+
205	0011	select count(order_id) from orders;	+
206	0011	select e.id, e.name, count(d.social_security) as number_of_departments from employee e inner join department d where d.social_security=e.social_security group by social_security;	+
207	0011	select count(id) from profile where registration_date between now() - interval 7 day and now();	+
208	0011	select * from table_name where primarykey in (select primarykey from table_name group by primarykey having count(primarykey) > 1) order by primarykey;	+
209	0011	select customerid, count(customerid) from maintenance where actiontype = 2 group by customerid having count(customerid) >= 1;	+
210	0011	select t1.id, t1.name, t1.country, count(bid) as	+

		bookings, sum(case when t2.vehicle = 'plane' then 1 else 0 end) as plane, sum(case when t2.vehicle = 'train' then 1 else 0 end) as train, sum(case when t2.vehicle = 'bus' then 1 else 0 end) as bus from table1 t1 inner join table2 t2 on t1.id = t2.orig	
211	0011	select id from table group by id having count(id) > 2;	+
212	0011	select id, names, count(names), total from tbl_products , (select count(distinct names) as total from tbl_products) as total where type = '1' group by names;	+
213	0011	select name, count(email) from users group by email having count(email) > 1;	+
214	0011	select a.post_name,a.post_title,a.id,b.meta_value as _sku from wp_posts a inner join (select meta_value,max (post_id) as post_id from wp_postmeta where meta_key=' _sku' group by meta_value having count(meta_value) > 1) b on a.id=b.post_id where post_type = 'product' and post_status = 'publish';	+
215	0011	select id, count(id) from table1 group by id having count(id)>1;	+
216	0011	select customer, count(liked) as total_likes from table where liked = 'true' group by customer	+
217	0011	select m.start_time, count(r.seconds) from dbo.minutes m group by m.start_time;	+
218	0011	select a.id from result a cross join result b where b. count = (select max(count) from result);	+
219	0011	select orders.entrydate as "date", count(orders.orderno) as "orders", sum(case when orders.reason is null then 1 else 0 end) as "replacements" from orders where orders. reason is null and orders.entrydate = '09-may-2014' and orders.customerno = 'a001' group by orders.entrydate;	+
220	0011	select count(f.id_foo) from my_remote_server_public. my_remote_server_public_foo f where f.date < _my_date;	+
221	0012	select count(*) as countbyid from items where fkid = 2003799 group by fkid having countbyid>1 order by countbyid;	+
222	0012	select v.concode,c.conname, min(v.rate), v.vendor from venprices v, country c where c.conid = v.concode group by c.conid;	+

223	0012	select u from user u where size(u.comments) = (select max(count(c.id)) from user u2 inner join u2.comments c group by u2.id)	+
224	0012	select t.person_id from table t group by t.personid having count(t.personid) > 3;	+
225	0012	select import_values.id, import_values.part_id, import_values.qty, import_values.note, parts. partterminologyname, group_concat(basevehicle.yearid, ' ', make.makename, ' ', model.modelname, ' ', submodel .submodelname separator ', '), group_concat(distinct(enginedesignation.enginedesignationname)	+
226	0012	select count(*) from tbl t group by t.ip_address order by count(*) desc limit 10;	+
227	0012	select name, max(b), max(c), min(b), min(c) from tablename group by name, b, c;	+
228	0012	select customer_name, count(purchases.customer_id) as number_of_purchaes from customer left join purchases on customer.customer_id = purchases.customer_id group by customer.customer_id;	+
229	0012	select c.name, count(*) as mycount from coupon c left join coupon_users u on c.id = u.coupon_id group by c.id order by mycount desc;	+
230	0012	select game.mdate, game.team1, sum(case when goal.teamid =game.team1 then 1 else 0 end) score1, game.team2, sum (case when goal.teamid=game.team2 then 1 else 0 end) score2 from game inner join goal group by game.mdate, goal.matchid, game.team1, game.team2;	+
231	0012	select itemprices.id, min(itemprices.lowprice) as minprice, itemprices.locationid from itemprices left join t where t.id is null group by locationid;	+
232	0012	select employees.empid, sum(workhours.hoursworked) as totalhours from employees inner join workhours where wh_month = 4 group by lastname, firstname, employees. empid;	+
233	0012	select sum(points) as total_points from sometable where total_points > 25 group by username;	+
234	0012	select count(1) from file_item p join type t on t.id = p.family_type_id group by p.family_type_id order by t. name;	+

235	0012	select group_name, group_id, sum(case when pass_fail = 'pass' then 1 else 0 end) as pass, sum(case when pass_fail = 'fail' then 1 else 0 end) as fail from log a join group b group by b.group_name, a.group_id;	+
236	0012	select knowledge.*, sorting.* from knowledge, sorting where knowledge.id = sorting.kid group by kid having count(id) < 2;	+
237	0012	select t.historyid from synk_isheet_1_int t where t. historyid = 6 and t.group1id = 27 and t.group2id = 4 group by t.historyid,t.requestentityid;	+
238	0012	select classes.classname, students.userid from classassociation join students join classes where classassociation.classid = 1 group by classname;	+
239	0012	select avg(m.a) from maintable m inner join #temptable t on m between t.startdate and t.enddate group by t. startdate	+
240	0012	select p.p_pid, p.p_name, p.p_url from products p inner join activity a on p.p_pid = a.a_pid where a.a_uid= ". $uid_int." group by p_pid, p_name, p_url order by max(a. a_time) desc limit 6;	+
241	0013	select customerkey from factinvoices i where i. dossierkey =2 and i.reportingdate between '2016-01-01' and '2017-12-31' group by customerkey;	+
242	0013	select ins1 as insurance from insauth2 where ins1 is not null group by ins1 union select ins2 as insurance from insauth2 where ins2 is not null group by ins2 union select ins3 as insurance from insauth2 where ins3 is not null union select ins4 as insurance from insauth2 where ins4 is not null union	+
243	0013	select cast(somecol as decimal(10,3)) from sometable group by cast(somecol as decimal(10,3));	+
244	0013	select distinct productid from x_product_ship group by productid having shipid <> 3;	+
245	0013	select country from yourtable group by country;	+
246	0013	select reservations.idcustomer from reservations (nolock) where excludedreservations.idcustomer is null and reservations.idcustomer is not null group by reservations.idcustomer;	+
247	0013	select zip from table1 where created between '2014-08-04 00:00:00' and '2014-08-08 23:59:59' group by zip order by count desc;	+

248	0013	select trackid from tracks where timestamp between tmin and tmax group by trackid;	+
249	0013	select t.purchase_date, t.qty, approx_percentile(t1.qty, 0.9) tp90 from mytable t inner join mytable t1 group by t.purchase_date, t.qty;	+
250	0013	select parent_id, listagg(child_id, ',') within group (order by child_id) as "children" from parentchildtable where parent_id = 0 group by parent_id;	+
251	0013	select product_name from purchase_history group by product_name;	+
252	0013	select animal from pets group by animal having animal not in (select animal from pets where name not in (' homer','bart','marge','lisa','maggie') group by animal) ;	+
253	0013	select pers_key, pers_name from visit_info a join valid_dates b where a.visit_date between b.start_date and b.end_date group by pers_key, pers_name;	+
254	0013	select id, group_concat(name order by name asc separator ', ') from my_table group by id;	+
255	0013	select country from countries group by country having sum(building) is null	+
256	0013	select t1.groupguid, t2.memberguid from temp t1, temp t2 where t2.isgroup = 0 group by t1.groupguid, t2. memberguid;	+
257	0013	select p.name from pc p left join sc s on p.color=s. color where s.color is not null group by p.name;	+
258	0013	select maker_id, status_id from cars where status_id != 0 group by maker_id, status_id union all select maker_id, max(status_id) max_status_id from cars group by maker_id having max_status_id = 0;	+
259	0013	select tb.id from tablea ta inner join tableb tb group by tb.id;	+
260	0013	select student_id, name, group_concat(subject separator ' ') from table1 join table2 on table1.student_id = table2.student_no group by student_id, name;	+
261	0014	select count(*) from t1 where value='0' union select count(*) from t1 where value='1';	+
262	0014	select * from mytable where a=x union all select * from mytable where b=y and a!=x	+
263	0014	select cc.contactpersonid, cc.clientcontactid, ad.city, ad.addressid from savedlist sl inner join clientcontacts	+

		cc on cc.contactpersonid = sl.objectid inner join clients c on c.clientid = cc.clientid inner join address ad on c.clientid = ad.objectid where sl.savedlistid = 2117 and (ad.city is not null)	
264	0014	select id,name,age from student where age < 15 union select id,name,age from student where name like '%a%' order by name;	+
265	0014	select * from citizen c where c.name = 'smith' union select * from citizen c where p.area = 'moon';	+
266	0014	select * from a_table where a_column = :a_var union select * from a_table where b_column = :a_var;	+
267	0014	select id, name from color where parentid=4 union select id, name from color where parentid=(select id from color where parentid=4);	+
268	0014	select * from table where id1 = :var1 and id2 = :var2 union all select * from table where id1 = :var2 and id2 = :var1;	+
269	0014	select resourceid from mytable where startdate between '2009-01-01' and '2009-01-20' and datediff(day, case when enddate < '2009-01-20' then enddate else '2009-01-20' end, startdate) >= 5 union select resourceid from mytable where enddate between '2009-01-01' and '2009-01-20' and datediff(day, enddate)	+
270	0014	select fiscal_period, fiscal_year, amount from maxtable where fiscal_period = 5 union all select fiscal_period, fiscal_year, amount from maxtable where fiscal_period = (select max(fiscal_period) from maxtable) and not exists (select * from maxtable where fiscal_period = 5);	+
271	0014	select * from xx where f_colour = "green" union all select * from xx where id not in (select distinct id from xx where f_colour = "green")	+
272	0014	select customers.firstname, customers.surname, customers .dob, customers.customeraddress from customers where customers.customeraddress like '%'+ 'main' + '%' union select customers.firstname, customers.surname, customers .dob, customers.customeraddress from customers where customers.customeraddress = null)	+
273	0014	select * from stock where stockid in (33) union select * from stock where stockid in (12) union select * from stock where stockid in (53) union select * from stock where stockid in (4) union select * from stock where	+

		stockid in (99) union select * from stock where stockid in (88);	
274	0014	select id from foo where a = 1 and b = 2 union all select id from foo where a = 3 and b = 4 union all select id from foo where a = 5 and b = 6;	+
275	0014	select a.id, b.model, c.color from cars a join models b join colors c join brands d where b.id=1 union all select a.id, b.model, c.color from cars a join models b join colors c join brands d where b.id=3;	+
276	0014	select id, name, group_id from mytable where id in (select min(id) from mytable group by group_id) union all select id, name, group_id from mytable where id in (select min(id) from mytable where id not in (select min (id) from mytable group by group_id) group by group_id) order by group_id;	+
277	0014	select meetingid, billid from mytable where billid is not null group by billid union all select meetingid, billid from mytable where billid is null;	+
278	0014	select id, event, seen, time_stamp from notifications n where id_user = :id and seen is null union all (select id, event, seen, time_stamp from notifications n where id_user = :id and seen is not null limit 15);	+
279	0014	select name from foo where name like 'm%' order by name desc union all select name from foo where name not like 'm%' order by name asc;	+
280	0014	select friend_user_id from friends where user_id = 1 union select friend_user_id from friends where user_id in (select friend_user_id from friends where user_id = 1);	+
281	0015	select parent.object_id,parent.event_time,parent.state, min(child.event_time) as ch_event_time, case when parent .state<>'done' and min(child.event_time) is null then (select localtimestamp)-parent.event_time else min(child .event_time)-parent.event_time end as step_time from objectstate parent where parent.object_id = 1	+
282	0015	select title, pubid as 'publisher id', pubdate as ' publish date' from books where pubid = 4 or pubdate > '01-jan-01' order by pubid asc;	+
283	0015	select * from tv_watchers where mins_watching_tv <= 60 or id=10 order by mins_watching_tv desc, id asc limit 6;	+

284	0015	select t.name as tablename, p.rows as rowcounts, convert (decimal,sum(a.total_pages)) * 8 / 1024 / 1024 as totalspacegb, sum(a.used_pages) * 8 / 1024 / 1024 as usedspacegb , (sum(a.total_pages) - sum(a.used_pages)) * 8 / 1024 / 1024 as unusedspacegb from sys.tables t inner join sys.indexes i on t.obj	+
285	0015	select a.column1, a.column2, b.column1, c.column1, cc. column1, cc.column2, cc.column3, d.column1 from table_a a inner join table_b b inner join table_c c left join table_d d inner join table_c cc where a.column1 = 1000 and b.column3 = 1 and c.column3 = 0 order by a.column1 asc;	+
286	0015	select q.postid, a.postid, c.commentid from posts q where q.postid = 1234 order by q.postid, a.postid, c. commentid;	+
287	0015	select salary from table_name where name='inputfromphp' and (surname='inputfromphp' or country='inputfromphp') order by case surname when 'inputfromphp' then 0 else 1 end, case country when 'inputfromphp' then 0 else 1 end limit 1;	+
288	0015	select name, score_string, place from scores s where game_id =1 and game_size =15 and game_level =1 and id = (select id from scores si where si.game_id =1 and si. game_size =15 and si.game_level =1 and si.name = s.name and si.place = s.place and si.date > "2010-10-01" order by si.game_id, si.game_ size = 16	+
289	0015	select id,title,release_date from tbl_movies where release_date > '2014-02-20' or release_date='' order by release_date asc;	+
290	0015	select m.discovery_time, m.ip, m.matched_id, m.uuid from mydata m where m.ip = '12.34.56.78' order by m.ip, m. discovery_time;	+
291	0015	select max(speed) as speed, time from info where id = 1 and time > 1234 order by id;	+
292	0015	select p1.store_number as st# from personnelfile as p1 left join payrollfile as p2 where p2.pay_date > '2010-05-14' and p2.uniform_allowance_amt_cppd in (8.25,8.50,300) and p1.jobs_series in ('2380','1458') and p1.ssn = '123456789' order by p1.ssn,p2.pay_date;	+
293	0015	select column1, column2, column3, coalesce(column4, 'foo ') as column4 from tablename where column1 = 'bar' order by column1, column2;	+

294	0015	select v.id, v.active, v.reg_no, p.install_date, p. remove_date from vehicle v left join period p on (v.id = p.car_id) where v.id = 1 order by v.id, p.install_date asc;	+
295	0015	select code, name from table1 where scope1='here' or scope2='room' group by code, name order by min(scope1), min(scope2), min(seq);	+
296	0015	select br.bm_tracking_number, (select tolist(appt.fact_date) from bm_fact appt where appt. bm_review_sk = br.bm_review_sk and appt.fact_type_code =183050) "appointments" from bm_review br where row_delete_date_time is null order by min(select appt .fact_date from bm_fact appt where appt.bm_review_sk =	+
297	0015	select `rp_products`.`product_code`, `rp_log`.` customer_id`, `rp_products`.`product_name` from rp_products, rp_log where (`rp_log`.`customer_id` = '111') order by `rp_products`.`product_code` asc, ` rp_log`.`customer_id` asc;	+
298	0015	select * from label where tenantid = 'jim' and tagfieldname = 'work' and tagid = 'work1' order by value, tenantid, tagfieldname, tagid, key limit 2;	+
299	0015	select pd.id,pd.price_date,pd.name_id,pd.class_id,pd. currency_id,pd.price, pd.price - (select price from price_data as x where x.price_date < pd.price_date and x .name_id = pd.name_id and x.class_id = pd.class_id and x .currency_id = pd.currency_id having max(x.price_date)) as `change` from price_data.	+
300	0015	select mq, im, pf, kmct from vehicules where mq='renault ' order by mq;	+
301	0016	select count(*) as countbyid from items where fkid = 2003799 group by fkid having countbyid>1 order by countbyid;	+
302	0016	select t.project, t.employee_id, sum(timestampdiff(hour , t.start_time, t.end_time)) as number_of_hours, max(e. billable_rate) as unit_price, sum(timestampdiff(hour, t .start_time, t.end_time)) from my_db.timesheet t join my_db.employee e on e.id = t.employee_id where t.project = 'ait' group by t.proje.	+
303	0016	select `periode_class_members`.`id`, `classes`.`id ` as `class`, `periode_class_members`.`periode`, ` user`.`firstname` as `firstname`, `user`.`lastname ` as `lastname`, `periode_class_members`.`status`, min(`periode_class_subjects`.`id`) as pcs from `	+

		periode_class_subject_members` left join `periode_c...	
304	0016	select class, min(grade) as highestgrade, freq, ranking from ranked where ranking = 1 group by class, freq, ranking;	+
305	0016	select car.personid as person, count(car.carid) as cars , null as pets from car where car.personid = 1 group by car.personid union all select pet.personid as person, null as cars, count(pet.petid) as pets from pet where pet.personid = 1 group by pet.personid	+
306	0016	select product.productname, component.componentname from product_component join component join product where product.productname = 'bread' group by product. productname;	+
307	0016	select name, year, sum(hours) from test_hours join test_users where year = 2020 group by users_id, year	+
308	0016	select speed , lat as lat1 , lon as lon1 from table1 where speed <> 0 union all select speed , lat as lat1 , lon as lon1 from table1 where speed = 0 group by speed, lat1,lon1 order by lat1;	+
309	0016	select id, code, sum(sale) as sale from tablename where code = 11 group by id, code;	+
310	0016	select d.full_date, count(*) from actions a join date_dimension d where d.full_date = '2010/01/01' group by d.full_date;	+
311	0016	select distinct from_member_id, to_member_id from ` single_chat` where from_member_id = 175 or to_member_id = 175 group by from_member_id, to_member_id;	+
312	0016	select customer_id, tax_code from orders where customer_id = 'some customer id' group by customer_id , tax_code;	+
313	0016	select service_name, metric_name, array_agg(value_textarray) from service_data where service_name = 'activitydataservice' group by service_name, metric_name ;	+
314	0016	select s.transaction_id, group_concat(i.item_name) from stock s inner join ref_item i where s.transaction_id = 123 group by s.transaction_id;	+
315	0016	select label1, label2, sum(number) as mysum from mytable where label1 = 'foo' group by label1, label2 having mysum > 0;	+

316	0016	select product.productname, component.componentname from product_component join component join product where product.productname = 'bread' group by product. productname;	+
317	0016	select p.proj_uid, p.proj_name,p.agency,p.district ,p.division,p.projstatus,civilbill80.billcount as civilbill80, civilbill20.billcount as civilbill20 , civilbillpay.billcount as finalcivilbill,civilworkslip. billcount as civilworkslip, electribill80.billcount as electricbill80, electribill20.billcoun	+
318	0016	select sum(quantity) as total from tablename where productid =1 and shopid in (1,2) group by productid;	+
319	0016	select a.userid, a.code, a.country, listagg(b.email, ',') within group (order by b.email) as "emails" from tab1.a, tab2.b where a.userid = b.userid and a.code = b. code and a.userid = 'rishi' group by a.userid, a.code, a .country;	+
320	0016	select division, count(id) as ct from test where role >=101 and division=1 group by division;	+
321	0017	select sc.studentid, c.classname, u. usergrouporganizationname, c.academicyearid , c.usergroupid, c.schoolid, sc.classid , u.usergrouporganizationstatusid from studentclasscrossreference sc inner join class c on sc.classid = c.classid inner join school s on s.schoolid = c.schoolid inner join dbo.usergr	+
322	0017	select table1_id, count(*) as count from table1_table2 group by table1_id having count > 2;	+
323	0017	select active, is_featured, count(*) from tbl_sales group by active, is_featured having (active=0 or active =1) and (is_featured=0 or is_featured=1);	+
324	0017	select sum(child_id) from children group by child_id having child_id = 5;	+
325	0017	select `businessid`, count(*) c from `biz_listing` where updated_date between '2014/06/01' and last_day ('2014/07/01') group by `businessid`,`type` having c = 2	+
326	0017	select a, b, sum(d) as c from tbla inner join (tblb inner join tblc on tblb.a = tblc.a) group by a, b having f like '*808*';	+
327	0017	select d.doctorid, d.doctorname, count(p.patientid) as patients from doctor d inner join patient p group by d.	+

		doctorid, d.doctorname having patients > 1;	
328	0017	select company, project from table group by company, project having type = 'dummy';	+
329	0017	select val, array_agg(fkey) fkeys from mytable group by val having array_length(array_agg(fkey),1) > 1;	+
330	0017	select first.subscriber_id, second.tag_id, count(*) as c from content_hits first join content_tag second on first.content_id=second.content_id group by second. tag_id,first.subscriber_id having c = 0;	+
331	0017	select lr.ansattnr, lr.rom, rb.behandling from rom_behandling rb inner join lege_rom lr on lr.rom = rb. rom group by lr.rom having rb.behandling = 'konsultasjon ';	+
332	0017	select id parent_id, title, (select count(id) from tbltree group by id having parent = parent_id) child_count from tbltree;	+
333	0017	select id, kmstand, count(*) as cnt from (select id , kmstand, vacationame, vacationvalue from `db_1`.` table_new` where (vacationame='vacation1' or vacationame ='vacation2' or vacationame='vacation3' or vacationame ='vacation4') group by id, kmstand, vacationame, vacationvalue having count(*) = 1) t gr.	+
334	0017	select label1, label2, sum(number) as mysum from mytable group by label1, label2 having mysum > 0 or label1 = ' foo';	+
335	0017	select c.country from orders o left join customers c on o.customer_id = c.customer_id join order_detail od where o.order_id = od.order_id group by country, month having rank <= 3;	+
336	0017	select first_name, last_name, class from students group by class having class = 'jss1';	+
337	0017	select first_name from students group by class having class in('a');	+
338	0017	select s.buyer_id, p.product_name from sales s join product p group by s.buyer_id having p.product_name = '' s8'';	+
339	0017	select name from schools group by city having city = city;	+
340	0017	select string_agg(title, ', ') as titles, genre, count (*) from films group by genre, title having genre='SciFi ';	+
341	0018	select a.brand from brands a join cars b group by a.	+

		brand;	
342	0018	select a.empid, a.name, b.name as dept_name from emp a left join department b;	+
343	0018	select o.orderid from orders as o join orderitems as oi join products as p;	+
344	0018	select e.id, e.name, count(d.social_security) as number_of_departments from employee e inner join department d group by e.id, e.name;	+
345	0018	select a.x, a.y from table_a a left join table_b b where b.x is null;	+
346	0018	select e.id, e.name, count(d.social_security) as number_of_departments from employee e inner join department d group by e.id, e.name;	+
347	0018	select table1.id, table1.start_date as table1_start_date , table1.end_date as table1_end_date, table2.start_date as table2_start_date, table2.end_date as table2_end_date from table1 inner join table2 order by table1.id, table1.start_date, table2.start_date;	+
348	0018	select food, description from tbl_foods join tbl_dishes;	+
349	0018	select ts.pagesize from syscat.tablespaces ts join syscat.tables tb where tb.tabschema = 'sysibm' and tb. tabname = 'dual';	+
350	0018	select distnct station, slot, subslot, compid, compname from devicetrace as dt inner join complist as cl;	+
351	0018	select * from category c join (select distinct pt. category_id from part pt) pqparts;	+
352	0018	select * from a inner join b;	+
353	0018	select p.personname from people p left join addresses a where a.addressid is null	+
354	0018	select d.end_date, extract(hour from end_time) as end_hour, count(t.users) as total_users from (select distinct cast(end_time as date) as end_date from table) d cross join table t group by e.end_date, h.end_hour;	+
355	0018	select omode.vehicle, omode.ordercode, omode.actiondate -imode.actiondate from (select vehicle, ordercode, actiondate from table where mode='o') omode, (select vehicle, ordercode, actiondate from table where mode='i') imode where omode.vehicle = imode.vehicle and omode. ordercode = imode.ordercode;	+

356	0018	select a.id, a.account, a.mydate from awesometable as a join (select account, max(mydate) as maxdate from awesometable group by account having maxdate < date_sub(now(), interval 12 month)) as b;	+
357	0018	select t1.id as t1_id,t3.data as t3_data from table3 t3 inner join table2 t2 inner join table1 t1;	+
358	0018	select distinct saletype.id, saletype.code, saletype. name from customer left join saletype_customer where (saletypeid = saletype.id or saletypeid is null) and customer.id= 4;	+
359	0018	select foo from footable foo join bartable bar where bar .anotherid=:another;	+
360	0018	select * from meetings m join (select attendee_id, max(meeting_date) from meetings group by attendee_id) attendee_max_date;	+
361	0019	select fname, lname from employee where not exists ((select pnumber from project where dnum = 5));	+
362	0019	select case when exists (select patientid from table2 t2 where t2.patientid =t1.patientid) then 'yes' else 'no' end as patientexists from table1 t1;	+
363	0019	select * from tblrecords where not exists (select personid from tblpeople group by personid having count(personid) > 1);	+
364	0019	select * from event where exists (select 1 from dual where mod(start_date - to_date(1, 'j') + level - 1, 7) = 6 or (mod(start_date - to_date(1, 'j') + level - 1, 7) = 3 and to_char(start_date + level - 1, 'dd') = '13'));	+
365	0019	select (select max(if(index=80, value, null)) from unnest(customdimensions)) as is_app, (select hits. eventinfo.eventaction) as ea from `table-big-query .105229861.ga_sessions_201711*`, unnest(hits) hits where totals.visits = 1 and _table_suffix between '21' and '21' and exists(select 1 from unnest(hi	+
366	0019	select * from a a where not exists (select 1 from ab m where m.a_id = a.id and exists (select 1 from b b where m.b_id = b.id and b.type = 'c'));	+
367	0019	insert into tableb select sum(a), sum(b), sum(c) from tablea where not exists (select * from table b where a=a and b=b) group by a, b;	+
368	0019	select * from `riders` where exists(select * from ` ridersclasses` where ridersclasses.rid = riders.id and ` cid` = '6') order by `first_name` asc;	+

369	0019	select * from test1 t1 where exists(select 1 from test2 t2 where (t1.id = t2.idc or t1.id = idp) and exists(select 1 from test3 where t2.idc = id or t2.idp = id))	+
370	0019	select * from students where exists (select studentid from student_to_hostel where hostelid=2);	+
371	0019	select e.email, case when exists(select * from userstbl tu where e.email = tu.username) then 'exist' else 'not exist' end as ex from (values('email1'),('email2'),('email3'),('email4')) e(email);	-
372	0019	select 1 from dual where exists (select 1 from user_objects where object_name = upper('client_sys. clear_info') and object_type = 'procedure') or exists (select 1 from user_procedures where object_name = substr(upper('client_sys.clear_info'), 1, instr(upper ('client_sys.clear_info'), '.') - 1)	+
373	0019	select * from t1 where exists (select null from t2 where y = x);	+
374	0019	select staff_no from doctor where not exists (select * from patient where staff_no = consultant_no);	+
375	0019	select * from final_combined_result wfcr where not exists (select contact_id, account_id from temp_wfcr);	+
376	0019	select id from usertable where exists (select 1 from (values (user_addedon, user_deletedon, user_modified)) ud (ud) where ud >= ri and ud < re);	+
377	0019	select * from current_stock where not exists (select * from stock_record)	+
378	0019	select count(*) as row_count from catat where catat.id = 1007642 and catat.is_parent = 1 and exists(select 1 from cg where cg.c_id = catat.id and exists(select 1 from ccsd where ccsd.g_id = cg.id));	+
379	0019	select id from dbo.splitstringtotable('2,3,6,7') where not exists (select 1 from tab where col = id);	+
380	0019	select c.custname, p.pjtitle from customer as c join project as p on p.custno = p.custno join tack as t on t.pjno = p.pjno where t.empid = 'glc' and not exists (select null from task where empid = 'glc' and empid = ' cac');	+
381	0020	select * from formfields where formfields.id in(select fields from form);	+
382	0020	select ancestorid from myview where ancestorid in (select id from #t);	+

383	0020	select eid, employee_name from employee where eid not in (select user1 from assign) and eid not in (select user2 from assign);	+
384	0020	select eid from entidades e where distrito in (select id from distritos where distrito_t like '%lisboa%');	+
385	0020	select ids from my_temp_table where ids not in (select id from table_one);	+
386	0020	select m.msg_id, m.uid_fk, m.message, m.alert, m.created , m.uploads, m.owner, u.uid, u.first_name, u.last_name from users u join messages m on m.uid_fk = u.uid where u.uid = :uid or u.uid in (select f.friend_two from friends f where f.friend_one = :uid) order by m.created desc limit 10;	+
387	0020	select categoryname from categories where id_category not in (select supercategoryid from supercategories);	+
388	0020	select * from data d inner join user u on 1=1 where (u.id, d.id) not in (select user_id, data_id from collection)	+
389	0020	select * from temp where part_in in (select count(part_id) as duplicates from temp where 1 group by part_id) and duplicates > 1;	+
390	0020	select table1.* from table1 where table1.id not in (select table2.key_to_table1 from table2 where table2.id = some_parm);	+
391	0020	select * from table1 where info1 in (select info2 from table1) and info2 in (select info1 from table1) and id not in (select id from table1 where (info1 in (select info2 from table1) and info2 not in (select info1 from table1)) or (info2 in (select info1 from table1) and info1 not in (select info2 from iofo1)	+
392	0020	select id from user where id not in (select owner_id from hd_ticket union all select submitter_id from hd_ticket union all select owner_id from hd_archive_ticket union all select submitter_id from hd_archive_ticket);	+
393	0020	select * from table1 a where a.col1 in (select b.col2 from table2 b) ;	+
394	0020	select f.field1, f.field2, f.field3 from foo f where f. field4 in (select b.bar from bar b where b.type = 4 and b.other = 7);	+
395	0020	select * from worklog w where 1=1 and w.class = '	+

		activity' and w.recordkey in (select wonum from woactivity where parent = 'm2176') union all select * from worklog w where 1=1 and w.class = 'workorder' and w.recordkey in (select wonum from workorder where parent = 'm2176');	
396	0020	select no, action_dt, request_type, status_cd, min (action_dt) over (partition by no, grp) as request_start_dt from (select w.*, count(case when status_cd in ('approved', 'denied') then 1 end) as grp from w) order by action_dt;	+
397	0020	delete from r2_table where r02_r01_id_fk in (select column_value from table(r01_ids));	+
398	0020	select id,name,xs1 as download, xs2 as upload from sc_params where rfen = 'service_requested' and code = 'speed' or code in (select code from sc_params where rfen = 'service_requested' and id = (select parent from sc_params where rfen = 'service_requested' and code = ' speed'));	+
399	0020	select table1.articleno,table1.artdescription,table2. year from table1 join table2 where not table1.articleno in (select table2.year from table2);	+
400	0020	select * from conversation where (least(sender_id, receiverid), greatest(sender_id, receiverid), date) in (select least(sender_id, receiverid) x, greatest (sende_id, receiverid) y, max(date) max_date from conversation) and '$uid' in (sender_id, receiverid);	+
401	0022	select a.name from users a, friends b where a.id=b. user_b and b.user_a = (select b.user_a from friends where a.name='s1');	+
402	0022	select * from tablea a where not exists(select * from tableb b where a.pid = b.pid and a.startdate >= '20-jun - 10');	+
403	0022	select cola,colb,colc,cold from mytable where exists (select 1 from (select i.itemid from items as i where iitemname like '%xxx%') as itm where itm.itemid=mytable. cola or itm.itemid=mytable.colb); select count(distinct i.third_party_id) as uniques from db.ids i where i.third_party_type = 'cookie_1' and i. first_party_id not in (select i.first_party_id where i. third_party_id = 'cookie_2');	+
404	0022	select cola,colb,colc,cold from mytable where exists (select 1 from (select i.itemid from items as i where iitemname like '%xxx%') as itm where	+

		itm.itemid=mytable. cola or itm.itemid=mytable.colb);	
405	0022	select * from t1 where t1.id in (select t2.id from t2 where t1.a = 'aa');	+
406	0022	select atc.owner, atc.table_name, atc.column_name from all_tab_columns atc where not exists (select acc.owner , acc.table_name, acc.column_name from all_cons_columns acc join all_constraints ac on acc.owner = ac.owner and ac.constraint_name = acc.constraint_name and ac. constraint_type in ('p', 'r')	+
407	0022	select * from table1 a where a.d > (select b.d from table2 b where a.id = b.id and a.something = 1);	+
408	0022	select * from (select e.id,e.name,sum(s.amount) as ' total_amount' from employee e inner join sale s on e.id= s.emp_id group by s.emp_id,e.id,e.name) as t1 where(0) =(select count(distinct(total_amount)) from(select e.id ,e.name,sum(s.amount) as 'total_amount' from employee e inner join sale s on e.id	+
409	0022	select * from table1 a where a.d > coalesce((select b.d from table2 b where a.id = b.id and a.something = 1), '0');	+
410	0022	select u.name from user u inner join (select uid from user_profile where p.address = 'some constant') p on u. uid = p.uid;	+
411	0022	select t2.word, t1.frequency, t2.secondword, t2. secondfrequency from (select * from (select word, secondword, secondfrequency, row_number() over(partition by word order by secondfrequency desc) as num from table_2) t where t.num <= 3) t2 join table_1 as t1 on t2.word = t1.word order by t2.secondfre	+
412	0022	select u.e_id, case when e_type_id = 1 then u.e_name else ' - ' + u.e_name end e_name, e_type_id, su.n_id from table1 u inner join table3 su on u.e_id = su.e_id where exists (select n_id from table2 where n_id = case when u.e_type_id = 1 then u.e_id else n_id end) order by e_type_id, u.e_name,n_id;	+
413	0022	select s.*, u.a, u.b, u.c, u.d, u.e, u.f, c. location_country, st.location_state, ct.location_city from (select user_id from tags where t.skillid = 52772) as t left join search s on t.user_id = s.user_id left join users u on t.user_id = u.user_id left join countries c on s.countryid = c.countryid lef	+
414	0022	select timeslots.timeslot, users.role, users.surname,	+

		users.clinic from timeslots, users where timeslots.id not in (select timeslot from appointments where appdate = getdate() and (users.clinic = 'werrington') and (users .role = 'doctor' or users.role = 'nurse')) and (users. role = 'doctor' or users.	
415	0022	select * from customer_tbl where exists(select 2 as customer_tbl where customer_tbl.country = 'mexico');	+
416	0022	update e set e.employeenumber = (select top 1 employeenumber from #employees where e.id = e.id order by newid()) from #employees e	+
417	0022	select distinct winner.person from (select case when t2_1.last_post > t2_2.last_post then person1 else person2 end as person from t1 inner join t2 t2_1 on t1. person1 = t2_1.person inner join t2 t2_2 on t1.person2 = t2_2.person) winner left join (select case when t2_1. last_post < t2_2.last_post then ..	+
418	0022	select `name` , count(*) as `count` from `t1`, `t2` where `t2`.`id` = `t1`.`id` group by `t2`.`id` union select name, 0 as count from t1 where not exists (select 1 from t2 where `t2`.`id` = `t1`.`id`);	-
419	0022	select a.id,a.type,a.date,b.status1,a.status2,a.status3 from table1 a inner join table2 b inner join table2 c group by a.type having count(a.type)>0 and b.status1=' aaa' union select a.id,a.type,a.date,b.status1,a.status2 ,a.status3 from table1 a inner join table2 b inner join table2 c group by a.type	+
420	0022	select * from bi_employee e where exists (select null from bi_user_access ua where ua.division_id = e. division_id and ua.product_id = e.product_id and ua. sub_product_id = e.sub_product_id and ua.region_id = e. region_id and (e.confidential = 'n' or ua.confidential = 'y') and ua.user_id = :user_id);	+
421	0022	select dbp.mob_num as mobile_number, dbp.name as name, dbp.area_code as area_code from db_phonebook dbp inner join (select mob_num from db_phonebook dbp where dbp. area_code = 4817 having count(distinct dbp.mob_num) = count(dbp.mob_num)) c_dbp where dbp.area_code = 4817	+
422	0022	select uv.id, if(uv.voc_id = 0,uv.word,sv.word) as word from user_vocabulary uv left join system_vocabulary sv having word like '%user_input%';	+
423	0022	select name, height * weight as inchpounds from	+

		sashelp. class having inchpounds > 5000;	
424	0022	select *, cond2 as cond2, cond3 as cond3 from table having cond1 and (cond2 or cond3);	+
425	0022	select * from card c where exists (select 1 from history h where h.cardid = c.cardid having count(case when h. statusid = 310 then 1 end) = 0);	+
426	0022	select department.dname from department join deptloc where deptloc.city in ('boston', 'dallas') having count(distinct deptloc.city) = 1;	+
427	0022	select * from item a having orderid not in (select orderid from table_excluded_item);	+
428	0022	create table tallest as select name, height from sashelp .class having height = max(height);	+
429	0022	select * from a t where exists (select 1 from a where id = t.id having count(distinct partid) > 1);	+
430	0022	select customerid, salesorderid, year(orderdate) as ' year' from sales.salesorderheader where year(orderdate) in (2011,2014) having count(year(orderdate))=2;	+
431	0022	select client_id from my_table having balance <> 0;	+
432	0022	select tag from tagging having count(distinct resource) > 2;	+
433	0022	select name from (select name, min(track) as track from horses group by 1 having count (distinct track) = 1) horses_one_race where track = 'sa';	+
434	0022	select userid, sum(money_spent), sum(money_spent_on_candy) / sum(money_spent) as percentcandyspend from moneytable where date >= '2010-01-01' having percentcandyspend > 0.1;	+
435	0022	select distinct name, version from table1 where name in ("asdf", "ghjk") having max(version) = version;	+
436	0022	select column_name, count(column_name) as column_name_tally from table_name where column_name < 3 having count(column_name) >= 3	+
437	0022	select title.id, title.title from titles as title having points > 0 union all select title.id, title.title from titles as title having points > 1;	+
438	0022	select person_id, max(salary) from yourtable a where exists (select 1 from yourtable b where a.person_id = b. person_id having (a.salary < max(b.salary) and count(*)	+

		> 1) or count(distinct salary) = 1);	
439	0022	select quizzes.*, count(submissions.id) as submissions_count from "quizzes" inner join "submissions " having count(distinct submissions.correct) >= 2;	+
440	0022	select id, name, count(name) from table group by 2,1 having count(name) = 2;	+
441	0023	select * from table where entry='cow' or entry = 'appl%' or entry = 'roo%';	+
442	0023	select medications.clinname from medications right join patientdata on patientdata.medid=medications.medid where patientdata.id='*the actual patient id*';	+
443	0023	select airline, flt_no, fairport, tairport, depart, arrive, fare from flights inner join airports from_port on (from_port.code = flights.fairport) inner join airports to_port on (to_port.code = flights.tairport) where from_port.code = '?' or to_port.code = '?' or airports.city='?';	+
444	0023	select count(tweet_id) from tweets where from_user = '%s ';	+
445	0023	select flights.*, fromairports.city as fromcity, toairports.city as tocity from flights left join (airports as fromairports, airports as toairports) where flights.fairport = '?' or fromairports.city = '?';	+
446	0023	select * from table where foo='#foo#';	+
447	0023	select * from employees where name = 'chris%'	+
448	0023	select * from providers where id='$[var1]'	+
449	0023	select * from sysobjects where name = '#temp_table';	+
450	0023	select * from `members` where `memberid` = '[id]' limit 1 union select * from `members`;	+
451	0023	select * from information_schema.columns where table_name = '[table name]';	+
452	0023	select column_name, table_name from information_schema .columns where schema_name = 'db_name' and table_name ='%98673%' and column_name like '%98673%';	+
453	0023	delete from tshirt where sku='%s';	+
454	0023	delete from tshirt del where del.sku = '%s'	+
455	0023	select count(*) from bo_labels l left join	+

		bo_contract_hardwood_deal c on (c.bo_document_fkey = l.bo_doc_base_fkey) where 1=1 and exists (select 1 from bo_party party where 1=1 and party.id = l.bo_party_fkey and party.inn = '?');	
456	0023	select * from tablename where fieldname = '#value#';	+
457	0023	select distinct owner, object_name from dba_objects where object_type = 'table' and owner = '[some other schema]';	+
458	0023	select * from table where entry='cow | appl* | roo*';	+
459	0023	select pg_terminate_backend(pg_stat_activity.pid) from pg_stat_activity where pg_stat_activity.datname = '[database to copy]' and pid <> pg_backend_pid();	+
460	0023	pdate wp_postmeta set meta_value = '0.25' from wp_postmeta as a inner join wp_woocommerce_order_itemmeta as b inner join wp_woocommerce_order_item as c where c.value = '%250g%';	+
461	0024	select clicks / impressions as probability, round(100 * probability, 1) as percentage from raw_data;	+
462	0024	select dev_cost / sell_cost from software ;	+
463	0024	select (won/total) as 'rankpercentage' from dbo. filmranking order by rankpercentage desc;	+
464	0024	select itm_num from itemconfig where (pal_qty/case_qty) > 500 and case_qty > 0;	+
465	0024	update mytable set ave_cost = cost / num;	+
466	0024	select * from table where (col1 / col2) between 1 and 8 and (col1 / col2) = floor(col1 / col2);	+
467	0024	select capacity_used / capacity_total from tablename where capacity_total is not null and capacity_total <> 0;	+
468	0024	select (a.quantity + b.quantity + c.quantity) as totalquantity, sum(a.quantity * a.rate) + sum(c.quantity * c.rate) as totalamount, totalquantity/totalamount as result from a, b, c where (a.userid = 1 and a.companyid = 1) and (a.userid = b.userid and a.userid = c.userid and a.companyid = b.companyid.	+
469	0024	select (select count(distinct s.lastfirst) from students s join cc on s.id = cc.studentid join courses c on cc.course_number = c.course_number where cc.schoolid ='109' and c.course_name like 'ap %' and substr(cc. termid,0,1) <> '-' and cc.dateenrolled between to_date ('08/01/2010','mm/dd/yyyy') and to..	+

470	0024	select h.total_sale, s.f1 / h.total_sale as f1_percent from sales s, (select id, f1 + f2 as total_sale from sales) h where s.id = h.id;	+
471	0024	update employee e set e.payroll = e.payroll + 1000 where e.payroll > (select department.dep_payroll / department .dep_amount from department where department.dep_id = e. dep_id);	+
472	0024	select *, r1.value / r.value - 1 as return from rownums r inner join rownums r1	+
473	0024	select id, count / maxcount as score from result;	+
474	0024	select round(noofboys / noofgirls) as ration from student;	+
475	0024	select * from xyz where x/y = (select max(x/y) from xyz) limit 1;	+
476	0024	select (current_salary/start_salary) appraisal, * from employee;	+
477	0024	select * from properties order by (price / floorsize);	+
478	0024	update factsales set unitcost = (select revenue / quantity from factsales);	+
479	0024	select material_id, cost/v_cost_total from materials where material_id >=0 and material_id <= 10;	+
480	0024	select *, (100-(table.price/table.oldprice))*100 as discount from table;	+

ДОДАТОК Б
(Обов'язковий)

КОПІЇ НАУКОВИХ ПУБЛІКАЦІЙ

ISSN 2307-5732
DOI 10.31891/2307-5732

НАУКОВИЙ ЖУРНАЛ **2.2023**

ВІСНИК

Хмельницького національного університету

Том 1

Технічні науки

Technical sciences

SCIENTIFIC JOURNAL
HERALD OF KHMELNYTSKYI NATIONAL UNIVERSITY

2023, Issue 2, Volume 319

Хмельницький

ВІСНИК
ХМЕЛЬНИЦЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ
серія: Технічні науки

Затверджений як фахове видання категорії «Б»,
РІШЕННЯ АТЕСТАЦІЙНОЇ КОЛЕГІЇ № 1643 ВІД 28.12.2019 та №409 від 17.03.2020

Засновано в липні 1997 р. ***Виходить 6 разів на рік***

Хмельницький, 2023, № 2(319)

Засновник і видавець: Хмельницький національний університет
(до 2005 р. – Технологічний університет Поділля, м. Хмельницький)

Наукова бібліотека України ім. В.І. Вернадського http://nbuv.gov.ua/j-tit/Vchnu_tekh

Включено до науково-метричних баз:

Google Scholar	http://scholar.google.com.ua/citations?hl=uk&user=aIUP9OYAAAAJ
Index Copernicus	http://jml2012.indexcopernicus.com/passport.php?id=4538&id_lang=3
Polish Scholarly Bibliography	https://pbn.nauka.gov.pl/journals/46221
CrossRef	http://doi.org/10.31891/2307-5732

Рекомендовано до друку рішенням вченої ради Хмельницького національного університету, протокол № 10 від 27.04.2023 р.

Адреса редакції: редакція журналу "Вісник Хмельницького національного університету"
Хмельницький національний університет
вул. Інститутська, 11, м. Хмельницький, Україна, 29016

☎ (038-2) 67-51-08 **web:** http://journals.khnu.km.ua/vestnik
http://lib.khnu.km.ua/visnyk_tup.htm

e-mail: visnyk.khnu@khmnu.edu.ua
visnyk.khnu@gmail.com

Зареєстровано Міністерством України у справах преси та інформації.
Свідоцтво про державну реєстрацію друкованого засобу масової інформації
Серія КВ № 24922-14862ПР від 12 липня 2021 року

ПРАВОРСЬКА НАТАЛІЯ, ЯШИНА ОКСАНА, НЕТРЕБА ІГОР, ДОМІНА АНАСТАСІЯ КИРИЧЕНКО ОЛЕКСАНДР
МЕТОД КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗГІДНО АНАЛІЗУ ПОМИЛОК SQL-ЗАПИТІВ 302

РОЗЛОМІЙ ІННА, НАУМЕНКО СЕРГІЙ
ШИФРУВАННЯ ТА СТИСНЕННЯ ІНФОРМАЦІЇ З ВИКОРИСТАННЯМ АВТОРСЬКИХ ШАБЛОНІВ, КЕРОВАНИХ МАТРИЦЕЮ 308

ФАНТ МИКОЛА
АРХІТЕКТУРА СИСТЕМИ МАШИННОГО НАВЧАННЯ ДЛЯ СТВОРЕННЯ ПАРАЛЕЛЬНИХ ДВОМОВНИХ КОРПУСІВ ТЕКСТІВ 314

ЗАХАРКЕВИЧ ОКСАНА, КОШЕВКО ЮЛІЯ, ЕЛЬНАШАР ЕЛЬСАЄД, ШВЕЦЬ ГАЛИНА, СЕЛЕЗНЬОВА АННА
ВПРОВАДЖЕННЯ ВІЗУАЛЬНОГО СЛОВНИКА З ТЕКСТИЛЮ ТА МОДИ У МОБІЛЬНИЙ ДОДАТОК 320

БОРТНИК ГЕННАДІЙ , КИРИЛЮК СЕРГІЙ, БРИЛЬ МИХАЙЛО
ШВИДКОДІЙНИЙ АНАЛОГО-ЦИФРОВИЙ ПЕРЕТВОРЮВАЧ З КОРИГУВАННЯМ ЧАСОВИХ ЗСУВІВ ІМПУЛЬСІВ ДИСКРЕТИЗАЦІЇ 329

ГОРЯЩЕНКО СЕРГІЙ , СИНЮК ОЛЕГ, ДРАПАК ГЕОРГІЙ, ГОРЯЩЕНКО КОСТЯНТИН, РОМАНЕЦЬ ТАРАС
АВТОМАТИЗАЦІЯ ПРОЦЕСУ НАНЕСЕННЯ ПОЛІМЕРНОГО ПОКРИТТЯ НА ДЕТАЛІ ЛЕГКОЇ ПРОМИСЛОВОСТІ..... 334

КРАСИЛЕНКО ВОЛОДИМИР, ПІДЛУБНИЙ ВЛАДИСЛАВ, НІКІТОВИЧ ДІАНА
ДОСЛІДЖЕННЯ ТА МОДЕЛЮВАННЯ МЕТОДУ ГЕНЕРУВАННЯ ПОТОКУ МАТРИЧНИХ КЛЮЧІВ ПЕРЕСТАНОВОК ТА ЇХ ХАРАКТЕРИСТИК ДЛЯ ЗАШИФРУВАННЯ-МАСКУВАННЯ ВІДЕОКАДРІВ..... 339

МИХАЙЛОВСЬКА ОКСАНА, НАДОПТА ТЕТЯНА
ПОКАЗНИКИ ЯКОСТІ ДЛЯ СПЕЦІАЛЬНОГО ВЗУТТЯ ВІЙСЬКОВОГО ПРИЗНАЧЕННЯ 348

ПАВЛОВСЬКИЙ ПАВЛО, ПРИСЯЖНИЙ ДМИТРО, АБРАМЧУК ІГОР, САВРАЦЬКИЙ В., БІЛОУС В.
ПІДВИЩЕННЯ ЗАХИСТУ ВІД НЕСАНКЦІОНОВАНОГО ДОСТУПУ ПІД ЧАС ГОЛОСУВАННЯ В ОРГАНАХ ДЕРЖАВНОЇ ВЛАДИ НА ОСНОВІ АПАРАТНОЇ БІОМЕТРИЧНОЇ ІДЕНТИФІКАЦІЇ ГОЛОСУЮЧОГО 355

ОЛІЙНИК ГАЛИНА, КОРНИЦЬКА ЛАРИСА, ДАНЧЕНКО ЮЛІЯ, РАСТОРГУЄВА МАРІЯ, ЄВТУШЕНКО ВАЛЕНТИНА
КЛАСИФІКАЦІЯ МЕБЛЕВО-ДЕКОРАТИВНИХ ТКАНИН 360

МАКАРЕНКО ВАЛЕРІЙ, МЄШКОВ ЮРІЙ, СЕЛІВЕРСТОВ ІГОР, ЛАЗОРИК ВЛАДИСЛАВ
ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ШДЯХІВ ПІДВИЩЕННЯ КОРОЗІЙНОЇ СТІЙКОСТІ СТАЛЕВИХ ТРУБОПРОВОДІВ 367

RUTKEVYCH VOLODYMYR, SHAPOVALUK SERHIY
ANALYSIS OF TRANSIENT PROCESSES IN THE ADAPTIVE HYDRAULIC DRIVE OF THE STEM FORAGE UNLOADER 199

FEDUSHKO SOLOMIIA
MODERN APPROACHES TO THE STUDY OF CYBERSECURITY AND CYBER HYGIENE IN THE FRAMEWORK OF DIGITAL TRANSFORMATION OF SOCIETY 210

LEZHNIUK PETRO, KOZACHUK OLEG, GALUZINSKIY OLEKSANDER
USE OF ACTIVE CONSUMERS FOR BALANCE OF ELECTRICITY IN THE ELECTRIC GRID 214

PASICHNIUK ANTON, TYKHOKHOD VOLODYMYR
METHODS AND TOOLS OF DOMAIN-DRIVEN DESIGN OF COMPLEX SOFTWARE SYSTEMS ON THE .NET CORE PLATFORM 222

KHROKALO LIUDMYLA
TESTING OF BACTERIAL FILTERS AND PRESERVATIVES FOR QUALITY ASSURANCE OF LYOPHILIZED SNAIL MUCUS AS A COSMETIC COMPONENT 229

KRAVCHENKO IGOR, MAMUTA MARYNA
EMISSIVITY OF FILAMENT LAMPS 234

SUBBOTA IRYNA
THE USE OF SILICEOUS ROCKS TO INCREASE THE STRENGTH OF CONSTRUCTION CERAMICS 240

BAHRII OLENA
USING A FINITE ELEMENT MODEL TO DETERMINE LATERAL PRESSURE OF A GRANULAR MEDIUM ON A RETAINING WALL UNDER DISPLACEMENT 245

HURMAN IVAN, BOBROVNIKOVA KIRA, POPOV YURY, BOYCHUK YAROSLAV, KACHUR VOLODYMYR
MACHINE LEARNING BASED METHODS FOR CYBERATACS DETECTION IN THE INTERNET OF THINGS INFRASTRUCTURE 251

VAHSCHYSHAK SERHII , STYSLO TARAS, STYSLO OKSANA, DEMCHYNA MYKOLA, SHKATULIAK VASYL
ADAPTIVE MODEL OF GAMIFICATION FOR HIGHER EDUCATION LEARNING PROCESS 258

KLIMENKO ANZHELIKA, SOKOLSKY GEORGII, KAMENSKA TETIANA
THERMOGRAVIMETRIC ANALYSIS OF BAKING BREAD PRODUCTS WITH ROSE HIP EXTRACT 265

KOSTYRKO VASYL, ANILOVSKA HANNA, PLESHA VASYL
DESIGNING A LIBRARY TO SIMPLIFY PROGRAM VERIFICATION CONDITIONS 273

ZUBKO OLGA, SHVETS GALINA, SVITLANA KULESHOVA, SELEZNEVA ANNA
DIGITAL TECHNOLOGIES OF CONSUMER IMAGE DEVELOPMENT 280

MOLCHANOVA KATERINA , ANDREYEVA OLGA, PERVAIA NATALIYA
APPLICATION OF PEPTIDES FOR THE MANUFACTURE OF COSMETIC CREAMS 288

NICHEPORUK ANDRII, NICHEPORUK ANASTASIIA, DANCHUK SERHII, KOROTKOV YURII, TSAVOLYK TARAS
SYSTEM FOR DATA COLLECTION AND DETECTION OF DISTRIBUTED DENIAL OF SERVICE ATTACKS IN THE RPL-BASED NETWORKS 296

PRAVORSKA NATALYA, YASHYNA OKSANA, NETREBA IHOR, DOMINA ANASTASIYA KYRYCHENKO OLEXANDER
A METHOD OF SOFTWARE DESIGN ACCORDING TO THE ANALYSIS OF SQL QUERY ERRORS 302

DOI 10.31891/2307-5732-2023-321-3-302-307
УДК 004.052.4

ПРАВОРСЬКА НАТАЛІЯ
Хмельницький національний університет
ORCID ID: 0000-0001-6001-3311
e-mail: margana2000007@gmail.com
ЯШИНА ОКСАНА
Хмельницький національний університет
ORCID: 0000-0001-7816-1662
e-mail: ipzhnu@gmail.com
НЕТРЕБА ІГОР
Хмельницький національний університет
ORCID: 0009-0009-1366-2429
e-mail: crfichyga@gmail.com
ДОМІНА АНАСТАСІЯ
Хмельницький національний університет
ORCID: 0009-0002-2170-5299
e-mail: anastasiya.domina.2015@gmail.com
КИРИЧЕНКО ОЛЕКСАНДР
Хмельницький національний університет
ORCID: 0009-0006-4149-212X

МЕТОД КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗГІДНО АНАЛІЗУ ПОМИЛОК SQL-ЗАПИТІВ

У статті наведено результати дослідження аналізу методів виявлення семантичних помилок для декларативної мови програмування – результатів аналізу Брасса і Голдберга, які здійснили аналіз виявивши, що бувають не тільки синтаксичні, але й семантичні помилки, які впливають на роботу програми. Подано список семантичних помилок, які часто зустрічаються під час створення запитів, для аналізу семантичних помилок використовувався набір з 191 834 із зібраних запитів більше ніж 36 000 містили помилку. Для кожної помилки здійснено опис проблеми, наведено приклад типічної помилки, та шляхи її вирішення, можливий варіант реалізації інструменту для їх виявлення для подальшого застосування. Аналізуючи поширеність семантичних помилок у SQL-запитах, виявилось, що найбільше поширеними помилками є відсутність предикатів сполуки, за якими слідують постійні помилки вихідного стовпця, та непотрібні аргументи лічильника. Також було виявлено, що спільна поява семантичних проблем у SQL-запитах для всього набору даних досить низька, що вказує на те, що запити рідко містять більше однієї семантичної помилки. Найбільша схожість між двома проблемами становить 20% для непотрібного аргументу підрахунку та непотрібного угруповання по атрибуту. Також було виявлено, що більш складні запити з точки зору кількості використовуваних сполук, предикатів та функцій, як правило, страждають від більшої кількості семантичних помилок, цікаве відкриття, яке може бути використане в майбутньому як метрика для раннього прогнозування того, чи буде запит може містити семантичні помилки або ні. На сьогоднішній день в Інтернеті багато ресурсів, які містять багато запитів з проблемами такого типу. Тому розробники які ознайомляться з даним дослідженням та з описаними вище проблемами, оцінять проблему та будуть виділяти більше часу для виявлення цих проблем, щоб усунути їх відразу після їх виявлення, а не в процесі роботи програми.

Ключові слова: помилки, SQL, семантичні помилки, декларативна мова програмування, аналіз, метод, програмне забезпечення, конструювання програмного забезпечення, програмування Інтернет, веб-технології.

PRAVORSKA NATALYA, YASHYNA OKSANA, NETREBA IHOR, DOMINA ANASTASIYA, KYRYCHENKO OLEXANDER
Khmelnytskyi National University, Ukraine

A METHOD OF SOFTWARE DESIGN ACCORDING TO THE ANALYSIS OF SQL QUERY ERRORS

The article presents the results of the analysis of methods for detecting semantic errors for a declarative programming language - the results of the analysis by Brass and Goldberg, who performed the analysis and discovered that there are not only syntactic, but also semantic errors that affect the operation of the program. A list of frequently encountered semantic errors during query generation is provided, a set of 191,834 of the collected queries was used to analyze semantic errors, more than 36,000 contained an error. For each error, a description of the problem is made, an example of a typical error is given, and ways to solve it, a possible option for implementing a tool for their detection for further use. Analyzing the prevalence of semantic errors in SQL queries, the most common errors were found to be missing join predicates, followed by persistent source column errors, and unnecessary counter arguments. It was also found that the co-occurrence of semantic problems in SQL queries for the entire data set is quite low, indicating that queries rarely contain more than one semantic error. The highest similarity between the two problems is 20% for an unnecessary count argument and an unnecessary grouping by attribute. It was also found that more complex queries in terms of the number of compounds, predicates and functions used tended to suffer from more semantic errors, an interesting finding that could be used in the future as a metric to early predict whether a query might contain semantic errors or not. Today, there are many resources on the Internet that contain many queries with problems of this type. Therefore, developers who read this study and the problems described above will appreciate the problem and will allocate more time to identify these problems in order to eliminate them immediately after they are discovered, and not during the operation of the program.

Keywords: errors, SQL, semantic errors, declarative programming language, analysis, method, software, software design, Internet programming, web technologies.

Постановка проблеми

Мова структурованих запитів, також відома як SQL, є спеціальною мовою програмування, яка використовується для керування та взаємодії з системами управління реляційними базами даних. Дана мова вважається однією з перших, що використовувала реляційну модель, представлену Коддом [1] у своїй роботі, а пізніше стала стандартом ANSI та ISO Це зробило її найпоширенішою мовою баз даних, оскільки більше 70% розробників використовують SQL. Слід зауважити, що SQL використовується не лише в IT, але й в інших галузях, таких як банківська справа, бухгалтерський облік, авіація, торгівля тощо. Це робить цю мову програмування однією з найпоширеніших. В той же час, це означає, що існує багато людей з різними рівнями навичок і досвіду, які пишуть і використовують запити SQL. Залежно від рівня розуміння SQL, деякі користувачі можуть напряму використовувати запити з форумів або інших веб-сайтів із запитаннями та відповідями, наприклад StackOverflow. Це має невід'ємний ризик, якщо ці запити містять помилки. З цієї причини, важливо мати інструменти, які можуть допомогти розробникам у виявленні не лише синтаксичних помилок у запитах SQL, а й семантичних. Подібно до того, як сучасні інтегровані середовища розробки пропонують переформатування коду для різних інших мов програмування.

Аналіз останніх досліджень і публікацій

У своїй роботі Стефан Брас і Крістіан Голдберг[1] зосередилися на роботі з SQL-запитами, які мають коректний синтаксис, але можуть містити семантичні проблеми. Ці проблеми можуть бути розділені на дві категорії: перша категорія включає помилки, які призводять до того, що запит неможливо виконати, і друга категорія містить коректні запити, але які не повертають очікуваного результату. Перша категорія помилок легше виявити та виправити, оскільки система управління базами даних надає повідомлення про помилку. Друга категорія може бути важчою для виявлення, оскільки проблема не завжди буває очевидною.

Автори додатково класифікували семантичні помилки у своїй роботі на дві групи, для першої завдання відомо заздалегідь, а для другої було достатньо одного SQL-запиту, щоб визначити помилку. Основний внесок їхньої роботи являє собою систематизацію семантичних помилок, які часто з'являються в SQL, усі помилки було зібрані із домашніх завдань та екзаменаційних матеріалів для одного з курсів в Університеті Галле.

Формулювання цілей статті

Хоча SQL-механізми здатні виявляти значну кількість синтаксичних помилок, найчастіше не виявляються семантичні помилки, які можуть призвести до серйозних проблем з продуктивністю програмного забезпечення або навіть до вразливостей у безпеці системи. У цій статті пропонується набір з підтверджених евристик разом з новим інструментом статичного аналізу на основі правил для виявлення найбільш поширених типів семантичних помилок в SQL-запитах на основі доказів з попередніх досліджень.

Виклад основного матеріалу

Семантична помилка в SQL-запиті, як визначено Брассом і Голдбергом [1], відноситься до законного запиту, який не (завжди) дає очікуваний результат. Якщо бути більш точним, це означає, що запит використовує правильний синтаксис SQL, проте результати, які він видає, є неправильними для даного завдання, що означає, що запит має певну семантичну помилку. Це типи помилок, які інструмент, представлений у цій статті, намагається виявити. Однак це завдання є нетривіальним, особливо коли немає інформації про завдання, для якого був написаний запит, або про те, як структуровані запитувані дані.

Семантичні помилки особливо часто з'являються в запитах, написаних студентами, які ще не оволоділи мовою SQL, однак вони також можуть бути присутніми в реальних програмах, що робить їх більш небезпечними. Оскільки ці проблеми зазвичай не генерують жодних попереджень від механізмів баз даних, тому недосвідченим розробникам важче їх виявити, особливо через те, що в деяких випадках запити можуть справді давати правильні результати. Саме з цієї причини можна стверджувати, що семантичні помилки в SQL є більш небезпечними, ніж синтаксичні помилки, які є можливість дуже легко виявити та виправити.

Ось чому інструменти для виявлення таких типів проблем мають бути безпосередньо інтегровані в середовища розробки або, навіть краще, у плагіни, які можуть перевіряти правильність запитів під час виконання. Крім того, семантичні помилки часто можуть впливати на загальну продуктивність запиту, що призводить до збільшення часу обчислення або неефективного використання ресурсів. Це знову ж таки дуже важливо в програмах, де швидкість є критичною, і виконання запитів, які містять семантичні помилки, може призвести до вузьких місць, як обговорювалося Мюз та ін. [2] у своїй статті. Хоча попередні дослідження не показали значної кореляції між ознаками проблемного коду SQL та іншими традиційними, це показали Мюз та ін. [2] що як тільки семантична помилка SQL з'являється в певній системі, вона, як правило, має більший термін служби, ніж інші більш традиційні помилки семантичного коду, що робить завдання швидкого виявлення таких типів помилок ще більш важливим. У цій стати пропонується кілька евристик, які використовуються для виявлення семантичних помилок у запитах SQL. Кожне з них реалізовано, як правила в розробленому інструменті статичного аналізу. Помилки, які виявляються запропонованою евристикою, були спочатку представлені та класифіковані в роботі Брасса та Голдберга [1], яка представляє повний список семантичних помилок, що зустрічаються в SQL. Невелику підмножину семантичних помилок також було зібрано за допомогою інструменту SQL Enlight. Це закритий інструмент статичного аналізу, розроблений Yubitsoft Eood, який також має на меті виявити ряд проблем у запитах SQL. Іншими

інструментами для аналізу SQL і виявлення помилок коду є TOAD і SQL Prompt, які також мають закритий код і потребують набору вхідних запитів, а також схем бази даних, щоб виявити будь-які проблеми.

Крім того, жоден із згаданих раніше інструментів не може аналізувати запити, вбудовані у вихідний код. Інструмент статичного аналізу на основі правил використовує жорстку реалізацію, тобто завжди передбачається найгірше, оскільки немає додаткової інформації, пов'язаної із завданнями чи схемами бази даних. Це означає, що для певних запитів інструмент може генерувати хибні позитивні попередження, як буде обговорюватися пізніше. Однак це було краще, оскільки, це дає хороший компроміс для налаштування, коли є мало інформації про запити. У наступних підрозділах подано опис кожної помилки, приклади SQL-запитів, які містять семантичну помилку, представлено реалізацію кожної стратегії виявлення разом із будь-якими припущеннями, зробленими евристичними методами, які використовуються в даному інструменті. Короткий перелік усіх семантичних помилок, які виявляє пропонований інструмент, також представлено в табл. 1

Таблиця 1

Список семантичних помилок

Семантична помилка	Опис
Порівнювання з NULL	Використання звичайних операторів порівнювання з NULL замість виразу IS NULL
Зайвий DISTINCT в агрегаціях	Для деяких функцій агрегації ключове слово DISTINCT не впливає на результат
Ділення на нуль	Можливе ділення на нуль через порядок виконання операцій
Відсутня умова JOIN	Відсутнє посилання для об'єднання таблиць
Непотрібна команда GROUP BY	Зайвий атрибут у групі GROUP BY, який не відображається в групі SELECT або HAVING поза агрегаціями.
Неефективне використання команди HAVING	Умова всередині виразу HAVING, яку можна перемістити всередину виразу WHERE, що робить запит більш ефективним

Порівнювання з NULL

Проблема: у деяких системах керування базами даних синтаксично допустимо використовувати A = NULL, однак цей вираз завжди має постійне значення істинності або NULL, або невідоме. Щоб уникнути таких ситуацій, завжди слід використовувати IS NULL або IS NOT NULL. Наступний приклад на рис. 1 повинен бути виявлений як такий, що має семантичну помилку.

```
SELECT t1.id, t1.surname , t1.name FROM table t1
WHERE t1.phone <> NULL;
```

Рис. 1. Семантична помилка порівнювання з NULL

У цьому прикладі t.phone порівнюється з NULL. Для деяких механізмів баз даних це може не вважатися синтаксичною помилкою, однак умова поверне NULL або unknown. Щоб уникнути подібних ситуацій, коли використовується значення NULL, ми завжди повинні використовувати вирази IS NULL або IS NOT NULL замість операторів порівняння. Потенційне виправлення для прикладу запиту наведено на рис. 2.

```
SELECT t1.id, t1.surname, t1.name FROM table t1
WHERE t1.phone IS NOT NULL;
```

Рис. 2. Виправлення помилки порівнювання з NULL

Стратегія виявлення: щоб виявити цю семантичну помилку, щоразу, коли в запиті використовуються оператори рівний або не рівний, ми перевіряємо, чи є один із термінів ключовим словом NULL. Якщо це так, тоді оператор порівняння слід замінити виразом IS NULL або IS NOT NULL відповідно.

Зайвий DISTINCT в агрегаціях

Проблема: для функцій агрегування MIN і MAX ключове слово DISTINCT ніколи не потрібне, оскільки воно не впливає на базові результати запиту. Крім того, у більшості випадків наявність дублікатів є, ймовірно, суттєвою при обчисленні результатів функцій агрегування SUM і AVG, тому дублікати не слід виключати, якщо для цього немає вагомих причин. Таким чином, присутність ключового слова DISTINCT у цих функціях агрегації є дивною і має викликати попередження. Для інших функцій агрегування неможливо визначити, чи потрібен DISTINCT у команді, не маючи додаткової інформації про завдання запиту, тому

інші функції агрегування виключаються з цієї перевірки. У наступному прикладі на рис. 3 слід виявити семантичну помилку.

```
SELECT SUM(DISTINCT price) AS income FROM orders
```

Рис. 3. Семантична помилка з зайвим DISTINC в агрегації

У цьому прикладі ключове слово DISTINCT використовується всередині функції SUM. Якщо для цього немає вагомих причин, присутність тут DISTINCT вважається дивною, оскільки дублікати, швидше за все, значні в цьому випадку, однак, не маючи додаткової інформації про завдання, для якого був написаний запит, наш інструмент видасть попередження про потенційну семантична помилку. Не маючи додаткової інформації про основне завдання, для якого був написаний запит, ми не можемо надати потенційне виправлення для цього запиту.

Стратегія виявлення: стратегія виявлення перевіряє, чи присутні в запиті будь-які функції агрегування MIN, MAX, SUM або AVG. Для кожного з них він потім визначає, чи використовувалося ключове слово DISTINCT у команді функції, і якщо так, виникає попередження.

Ділення на нуль

Проблема: Однією з поширених проблем, пов'язаних із операторами типів даних, є ділення на нуль. Оскільки в SQL немає гарантій на послідовність оцінки операторів у команді WHERE, розробникам важко уникнути таких ситуацій. Наступний приклад на рис. 4 вважається небезпечним і має бути виявлений як семантична помилка.

```
SELECT items FROM orders
WHERE (amount / count) > 500 AND count > 0;
```

Рис. 4. Семантична помилка з діленням на нуль

У цьому прикладі, хоча умова count > 0 присутня в запиті WHERE, оскільки порядок операцій не виконується, цей запит усе ще небезпечний. Таким чином, попередження повинно бути викликано, коли в запиті є ділення, за винятком ситуацій, коли ділення знаходиться в операторі SELECT, а стовпець дільника перевіряється на нерівність нулю в команді WHERE, оскільки в цих випадках WHERE буде оцінено перед SELECT будь-якою системою бази даних.

Стратегія виявлення: реалізація цього правила аналізує запит і зберігає всі знайдені оператори ділення. Окрім цього, також створюється логічний прапорець, який вказує, чи був термін поділу після SELECT чи ні. Крім того, усі стовпці, які перевіряються на нерівність нулю в команді WHERE, також зберігаються. Нарешті, ділення, яке знаходяться в запиті SELECT, але для яких стовпець дільника перевірено в операторі WHERE, видаляються з початкового набору, а решта членів ділення повертаються як результат для цього правила, яке генерує попередження про можливе ділення на нуль.

Порівнювання з NULL

Проблема: ця помилка з'являється в запитах із об'єднаними таблицями, для яких джерела об'єднаних таблиць не мають жодного стовпця, на який посилаються дані в умові JOIN, ані в WHERE. Якщо предикат JOIN відсутній, запит включатиме декартовий добуток усіх рядків, також відомий як перехресний добуток, що, безсумнівно, призведе до збільшення продуктивності для запитів і потенційно неправильних результатів. Тому важливо, щоб об'єднані таблиці посилалися в командах JOIN ON або WHERE, щоб уникнути подібних проблем. Однак є один виняток, зокрема, коли використовується CROSS JOIN. У цих випадках немає потреби, щоб залучені джерела таблиці мали посилання на будь-які стовпці, оскільки для цих запитів очевидною метою є отримання перехресного добутку всіх рядків, тому попередження не повинно створюватися. У наступному, на рис. 5, прикладі слід виявити семантичну помилку.

```
SELECT name , surname, email , phone , customer_id
FROM organizations AS t1 INNER JOIN customers as t2;
```

Рис. 5. Семантична помилка з порівнюванням з NULL

У цьому прикладі є дві об'єднані таблиці, organizations і customers, які не мають жодних посилань на стовпці ні в умовах JOIN, ні в WHERE. Це означає, що результат запиту включатиме перехресний добуток усіх рядків. Незалежно від того, чи був це намір автора запиту чи ні, відсутні умови з'єднання є потенційною причиною зайвих витрат на продуктивність і повинні бути повідомлені як семантичні

помилки. Не маючи додаткової інформації про основне завдання, для якого був написаний запит, ми не можемо надати потенційне виправлення для цього запиту.

Стратегія виявлення: Виявлення цієї семантичної помилки здійснюється шляхом аналізу запиту та відстеження всіх використаних джерел таблиці. Якщо використовується менше двох таблиць або використовується CROSS JOIN, немає потреби продовжувати перевірку відсутності умов з'єднання, тому попередження не виникатимуть. Якщо використовуються принаймні два джерела таблиць, тоді перевіряються команди ON і WHERE відповідного підзапиту, щоб визначити, чи мають ці джерела таблиці будь-які стовпці, на які посилаються. Для будь-якої таблиці, на яку немає посилання, буде створено попередження про відсутність умови об'єднання.

Непотрібний команда GROUP BY

Проблема: кожен раз, коли атрибут, який з'являється в комнаді GROUP BY, функціонально визначається іншими атрибутами, і якщо він не відображається в командах SELECT або HAVING поза функціями агрегації, тоді його можна взагалі видалити з GROUP BY. Для нашого інструменту ця умова послаблена, оскільки ми розглядаємо лише запит, не знаючи схеми бази даних, тому неможливо визначити, чи атрибут функціонально визначається іншими атрибутами чи ні. Тому ми перевіряємо лише атрибут, який з'являється в команді GROUP BY, але не використовується ні в командах SELECT, ні в HAVING поза будь-якими функціями агрегації. У цьому випадку буде викликано попередження про цю семантичну помилку. У наступному прикладі на рис. 6 слід виявити семантичну помилку.

```
SELECT COUNT(*) AS counter FROM customers
WHERE amount = 122342
GROUP BY amount HAVING counter > 1 ORDER BY
counter;
```

Рис. 6. Семантична помилка з непотрібною командою GROUP BY

У цьому прикладі атрибут amount присутній у команді GROUP BY, але не використовується ні в команді SELECT, ні в команді HAVING поза функціями агрегації, тому його можна видалити з GROUP BY. Це ще більше спростить запит, оскільки amount є єдиним атрибутом, наявним у GROUP BY. Крім того, цей запит містить ще одну семантичну помилку, оскільки amount може мати лише одне значення, оскільки воно прирівнюється до деякої константи в команді WHERE, тому групування не матиме впливу на результат. Потенційне виправлення для прикладу запиту наведено на рис. 7.

```
SELECT COUNT(*) AS counter FROM customers
WHERE amount = 122342
HAVING counter > 1 ORDER BY counter;
```

Рис. 7. Виправлення помилка з непотрібною командою GROUP BY

Стратегія виявлення: стратегія виявлення цієї помилки аналізує запит і відстежує атрибути, що з'являються в командах SELECT і HAVING поза функціями агрегації. Крім того, атрибути, які з'являються в команді GROUP BY, також зберігаються. Після завершення аналізу запиту ми перевіряємо атрибути, знайдені в команді GROUP BY, яких немає в жодному з двох списків з атрибутами, виявленими в командах SELECT або HAVING. Для кожного з цих атрибутів виникає попередження, яке вказує на те, що в запиті виявлено семантичну помилку.

Неефективне використання команди HAVING

Проблема: якщо запит має точно такі самі атрибути в команді SELECT, перелічених у команді GROUP BY, і якщо в жодному з двох пунктів не використовуються функції агрегації, тоді команд GROUP BY можна замінити на SELECT DISTINCT. У більшості випадків оптимізатор SQL створює однакові або схожі плани виконання для двох запитів, тому в цих випадках не завжди є виграш у продуктивності, однак, переписавши запит, це стає коротшим і зрозумілішим. У більшості цих випадків використання команди GROUP BY по суті видаляє дублікати з набору результатів, тому оператор DISTINCT краще підходить для використання, на відміну від ситуацій, коли використовуються функції агрегації, і в цьому випадку GROUP BY справді потрібна. У наступному прикладі на рис. 8 слід виявити семантичну помилку.

```
SELECT t1.genre FROM film t1 LEFT JOIN cartoon t2
ON t1.actor = t2. actor
WHERE t2. actor IS NOT NULL GROUP BY t1. genre;
```

Рис. 8. Семантична помилка з неефективним використанням команди HAVING

У цьому прикладі всі атрибути SELECT перераховані під командою GROUP BY, а також не використовуються функції агрегації, тому кманду GROUP BY можна викинути та замінити на SELECT DISTINCT, зробивши запит коротшим і зрозумілішим. Потенційне виправлення для прикладу запиту наведено на рис. 9.

```
SELECT DISTINCT t1.genre FROM film t1 LEFT JOIN
cartoon t2 ON t1.actor = t2. actor
WHERE e2. actor IS NOT NULL;
```

Рис. 9. Виправлення помилки з неефективним використанням команди HAVING

Стратегія виявлення: реалізація цієї стратегії спочатку перевіряє, чи використовуються будь-які функції агрегації в командах SELECT або GROUP BY. Якщо це так, то немає необхідності продовжувати подальшу перевірку запиту на цю семантичну помилку. В іншому випадку ми продовжуємо, виявляючи всі змінні, що використовуються в команді SELECT, а також ті, що використовуються в команді GROUP BY. Якщо існує ідеальна відповідність між цими двома наборами термінів, це означає, що команду GROUP BY можна замінити на SELECT DISTINCT і спрацьовує попередження про цю семантичну помилку.

Реалізація

Для того щоб виявити попередньо описані семантичні помилки, потрібно перетворити SQL в дерево класів, для якого згодом можна використовувати шаблон проектування «Відвідувач», адже від дозволяє відслідковувати поведінку програми шляхом додання їй нових операцій, не змінюючи класи об'єктів, над якими, власне, і будуть виконуватись операції. На початку ми будемо отримувати потік, який буде починатись з якого вхідного запиту, в процесі буде перевірюватись кожне з правил, і якщо воно буде порушуватись, запит буде відмічатись як проблемний, і буде з'являтись попередження про те, що семантична помилка виявлена. Основну увагу тут слід приділити розробці інструментів для масового впровадження. Вибір цього, головного, рушійного фактора під час створення нових інструментів має допомогти переконатися, що розробники дійсно бачать цінність інструменту. Зокрема, наші висновки показують, що зараз розробникам важко перевіряти свої SQL-запити на наявність семантичних проблем через одну основну проблему, якою є відсутність відповідних інструментів і підтримки. Поточним реалізаціям інструментів, які інтегруються з IDE, дуже бракує підтримки виявлення семантичних проблем для запитів, крім того, наскільки нам відомо, також немає інструментів, інтегрованих із популярними фреймворками розробки, такими як PhpStorm, для перевірки цих проблем під час виконання програми.

Висновки

Таким чином, можна зробити висновок про те, що виявлення семантичних помилок в процесі розробки програмного забезпечення є важливим і необхідним процесом для збільшення надійності програмного забезпечення, на сьогоднішній день велика кількість додатків використовують SQL для запитів, тому важливо розуміти розробникам результати та наслідки їх запитів, які містять різні семантичні проблеми, на сьогоднішній день в Інтернеті багато ресурсів, які містять багато запитів з проблемами такого типу. Тому розробники повинні ознайомитись не тільки з описаними вище проблемами, але й виділяти більше часу для виявлення цих проблем, та їх усуненню відразу після їх виявлення.

References

1. Codd E.F. A Relational Model of Data for Large Shared Data Banks, IBM Research Laboratory, San Jose, California. URL: https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf.

2. Muse B.A. On the prevalence, impact, and evolution of sql code smells in data-intensive systems. In Proceedings of the 17th International Conference on Mining Software Repositories, p. 327–338, 2020. DOI: 10.1145/3379597.3387467.

3. Molinaro Anthony, de Graaf Robert. SQL Cookbook: Query Solutions and Techniques for All SQL Users 2nd Edition. 2020. 567 p.

4. Allen G. Taylor. SQL For Dummies (For Dummies (Computer/Tech)). 9th Edition. 2018. 512 p.

5. Viescas John. SQL Queries for Mere Mortals: A Hands-On Guide to Data Manipulation in SQL 4th Edition. 2018. 960 p.

6. Beaulieu Alan. Learning SQL: Generate, Manipulate, and Retrieve Data 3rd Edition. 2020. 377 p.

ДОДАТОК В
(Обов'язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРИАЛИ

Кафедра інженерії програмного забезпечення

Метод виявлення семантичних помилок у коді на декларативній мові програмування SQL за допомогою статичного аналізу

Виконав: ***студент гр. ІПЗм-22-1 Нетреба І.В.***

Керівник: ***канд. пед. наук*** ***Праворська Н. І.***

Попередні дослідження

Дана тема була висвітлена в статті Голдберга та Брасса «Stefan Brass and Christian Goldberg. Semantic errors in sql queries: A quite complete list. Journal of Systems and Software» та «Stefan Brass, Christian Goldberg, and Alexander Hinneburg. Detecting semantic errors in sql queries. Technical report, Technical Report, University of Halle».

Вони були першими хто зосередився на класі SQL-запитів, які правильні з синтаксичної точки зору, але містять семантичні проблеми, незалежно від завдання запиту.

Актуальність проблеми

- Актуальність теми роботи полягає в необхідності вирішення проблеми пошуку семантичних помилок в запитах, а також визначенню факторів, які впливають на появу таких помилок.
- SQL є одною з найпоширеніших мов програмування існує багато людей з різними рівнями навичок і досвіду, які пишуть і використовують запити SQL. Залежно від рівня розуміння SQL, деякі користувачі можуть напряму використовувати запити з форумів або інших веб-сайтів із запитаннями та відповідями, наприклад StackOverflow. Це має невід'ємний ризик, якщо ці запити містять помилки.

Об'єкт, предмет, мета дослідження

Об'єкт дослідження. Інструмент для аналізу на семантичні помилки запити написаних на SQL

Предмет дослідження. Метод виявлення семантичних помилок у коді на декларативній мові програмування SQL за допомогою статичного аналізу

Мета роботи. Підвищити ефективність пошуку семантичних помилок в запитах SQL

Задача роботи. Дослідження можливостей для оптимізації перевірки запитів, рішення для виявлення та попередження на семантичні помилки

Завдання дослідження

- Основними завданнями роботи виступають:
- визначити, список помилок які найчастіше всього зустрічаються в запитах;
- проаналізувати існуючі рішення
- проаналізувати роботу бази даних при виявлені семантичних помилок
- дослідити бібліотеки для роботи та парсингу SQL-запитів
- дослідити вплив інших факторів
- дослідити, як краще аналізувати запити без зв'язку та інформації про базу даних
- проаналізувати, наскільки якісними є представлена методика

Практичне значення

- Практична цінність отриманих результатів полягає в успішній класифікації, аналізу та розробціметодик та алгоритмів для усунення проблеми появи семантичних помилок в застосунках.
- Завдяки результатам дослідження є можливість оцінити вплив різних факторів на швидкодію та вихідний результат отриманих запитів після виконання.

Проблема написання правильних запитів

Звичайно, загалом важко стверджувати, що синтаксично правильний запит є семантично неправильним, якщо не знати завдання, для якого був написаний запит. Однак запити можна вважати «імовірно не вірними», якщо вони надто складні. Припустімо, що користувач написав запит Q, і існує еквівалентний запит Q(t), який є значно простішим і може бути отриманий із Q шляхом видалення певних частин. Можуть бути наступні причини, чому користувач не написав Q(t):

- користувач знав, що Q(t) не є правильним формулюванням поставленого завдання.
- користувач не знав, що Q(t) еквівалентне.
- користувач знав, що Q(t) еквівалентний, але він або вона вірив, що Q працюватиме швидше .
- користувач знав, що Q(t) еквівалентний, але він або вона думав, що Q буде зрозумілішим для людини, яка читає, і простіше підтримувати.

Статичний аналіз коду

- Статичний аналіз коду (англ. static code analysis) — аналіз програмного забезпечення, який здійснюють (на відміну від динамічного аналізу) без реального виконання програм, що досліджуються. Зазвичай аналізу піддають початковий код, хоча іноді аналізу піддається об'єктний код, наприклад, P-код або код CIL. Термін зазвичай застосовують до аналізу, який проводить спеціальне програмне забезпечення (ПЗ), тоді як ручний аналіз називають «program understanding», «program comprehension» (розумінням або осягненням програми).

Семантичні помилки в запитах

Інструмент статичного аналізу на основі правил, розроблений для цього проекту, використовує жорстку реалізацію, а це означає, що завжди передбачається найгірше, оскільки немає додаткової інформації, пов'язаної із завданнями чи схемами бази даних. Це означає, що для певних запитів інструмент може генерувати неправдиві спрацьовування попереджень, як буде обговорюватися пізніше, однак це було кращим, оскільки він пропонує гарний компроміс для налаштування, коли відомо мало інформації про запити.

Статичний аналіз коду

- Статичний аналіз коду (англ. static code analysis) — аналіз програмного забезпечення, який здійснюють (на відміну від динамічного аналізу) без реального виконання програм, що досліджуються. Зазвичай аналізу піддають початковий код, хоча іноді аналізу піддається об'єктний код, наприклад, P-код або код CIL. Термін зазвичай застосовують до аналізу, який проводить спеціальне програмне забезпечення (ПЗ), тоді як ручний аналіз називають «program understanding», «program comprehension» (розумінням або осягненням програми).

Семантичні помилки в запитах

Інструмент статичного аналізу на основі правил, розроблений для цього проекту, використовує жорстку реалізацію, а це означає, що завжди передбачається найгірше, оскільки немає додаткової інформації, пов'язаної із завданнями чи схемами бази даних. Це означає, що для певних запитів інструмент може генерувати неправдиві спрацьовування попереджень, як буде обговорюватися пізніше, однак це було кращим, оскільки він пропонує гарний компроміс для налаштування, коли відомо мало інформації про запити.

Стратегії виявлення

- Перевірка синтаксису
- Перевірка назв таблиць та колонок
- Перевірка правильності запиту
- Перевірка зв'язків між таблицями
- Перевірка унікальності та обмежень
- Використання тестових даних для виконання SQL-запитів
- Використання логічних перевірок
- Запобігання SQL-ін'єкціям
- Використання SQL-інструментів для від лагодження
- Аудит запитів

Семантичні помилки

Постійний вихідний стовбець	Несумісна змінна кортежа
Дублювання вихідного стовпця	Непотрібна умова ORDER BY
Непотрібна інструкція JOIN	Непотрібна інструкція GROUP BY
Ідентичні змінна кортежа	Непотрібна умова UNION
Порівняння з NULL	Непотрібна фраза GROUP BY
Відсутня інструкція JOIN	Непотрібне загальне порівняння
Некорельованіпідзапити EXISTS	Використання LIKE без символів підстановки
Ділення на нуль	Непотрібний список SELECT в EXISTS
Дивна інструкція HAVING	Непотрібне сканування індексу
Дивні символи підстановки без LIKE	Непотрібний DISTINCT в агрегаціях
Неспівпадіння в підзапиті SELECT	Непотрібний атрибут GROUP BY
Дивні умови підзапита	Непотрібний аргумент COUNT
Неефективна інструкція HAVING	

Евристики

Порівняння з NULL

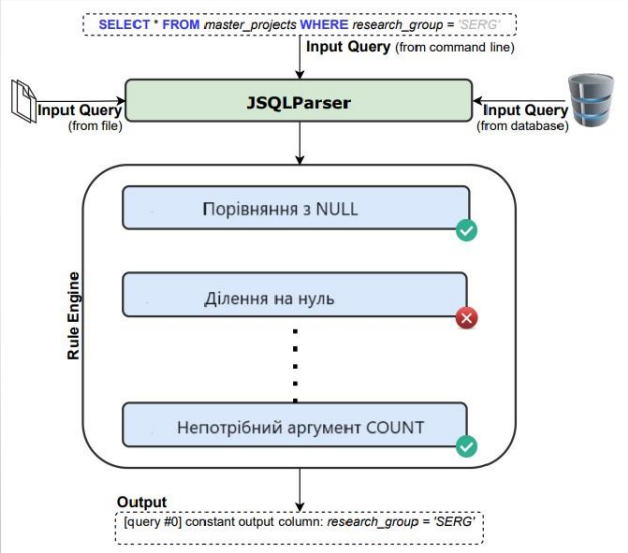
```
SELECR r.username FROM root r r.id, r authtoken.instagram, WHERE r.abc <> NULL
```

Проблема: Проблема: у деяких системах управління базами даних синтаксично допустимо використовувати A = NULL, проте цей вираз завжди має постійне істинне значення або NULL , або невідоме. Щоб уникнути таких ситуацій, натомість завжди слід використовувати IS NULL або IS NOT NULL .

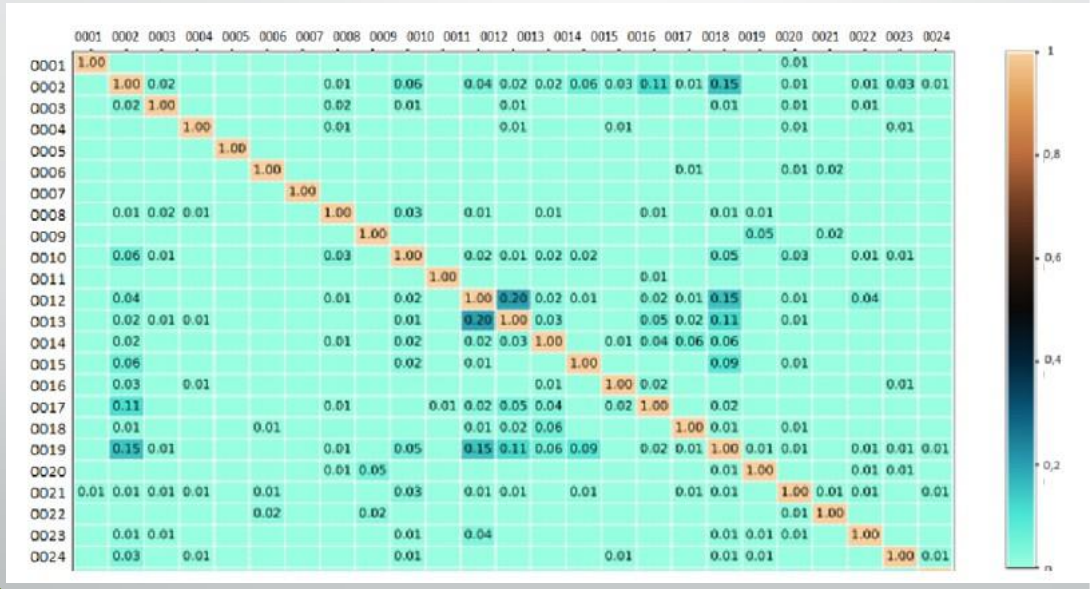
```
SELECT r.id, r.authtoken.instagram , r.username
FROM root r
WHERE r.abc IS NOT null;
```

Стратегія виявлення. Для виявлення цієї семантичної помилки щоразу, коли всередині запиту використовуються оператори «рівно» чи «не рівно», перевіряється, чи є одне з умов ключовим словом NULL . Якщо це так, оператор порівняння слід замінити виразом IS NULL або IS NOT NULL відповідно

Робота інструмента



Матриця збігів семантичних помилок



Наукова новизна

Удосконалено метод для аналізу запитів написаних на мові програмування SQL та пошуку в них семантичних помилок.

Розроблено евристики та інструмент на їх основі для аналізу вхідних запитів та виведенням правильного варіанту їх написання у разі виявлення помилки.

Публікації

- Праворська Н.І., Яшина О.М., Нетреба І.В., Доміна А.Р. Кириченко О.М. Метод конструювання програмного забезпечення згідно аналізу помилок SQL-запитів. Вісник Хмельницького національного університету – 2023, №3 – 302-307 с

Висновки та рекомендації

В результаті виконання дипломної роботи було проведено аналіз предметної області, було проаналізовано фактори, створено список факторів які впливають на появу семантичних помилок та їх список.

Запропоновано метод, який дозволить покращити роботу написання правильних запитів. На основі отриманих даних спроектовано та розроблено інструмент який в подальшому, можна імплементувати в середовища розробки.

Дякую за увагу

UNICHECK
by Turnitin

Ім'я користувача:
ІПЗ

ID перевірки:
1015983999

Дата перевірки:
08.12.2023 13:20:40 EET

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
08.12.2023 13:21:43 EET

ID користувача:
100012953

Назва документа: Диплом_Нетреба_ІПЗм_22_11

Кількість сторінок: 92 Кількість слів: 20109 Кількість символів: 154306 Розмір файлу: 884.43 KB ID файлу: 1015664690

6.87%
Схожість

Найбільша схожість: 1.22% з Інтернет-джерелом (https://jyx.jyu.fi/bitstream/handle/123456789/71720/978-951-39-8290-4_

6.81% Джерела з Інтернету | 926 ..... Сторінка 94

0.58% Джерела з Бібліотеки | 74 ..... Сторінка 103

0% Цитат

Не знайдено жодних цитат

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи | 2

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 6%**

	Документ		Сумарний збіг по Базі Даних	
ID: 122182 Назва: Метод виявлення семантичних помилок у коді на декларативній мові програмування SQL за допомогою статичного аналізу Додано в БД: 2023-12-08 Автора: Нетреба І.В. Керівники: Праворська Н.І. Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	136457	957	1634 (1%)	23 (2%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «магістр»

Магістр Нетреба Ігор Вікторович

Тема Метод виявлення семантичних помилок у коді на декларативній мові програмування SQL за допомогою статичного аналізу

Спеціальність 121 «Інженерія програмного забезпечення»

Обсяг кваліфікаційної роботи:

Кількість сторінок кваліфікаційної роботи 148 .

1. Короткий зміст роботи та прийнятих рішень У кваліфікаційній роботі детально проаналізовано помилки, що виникають під час написання запитів у коді на декларативній мові програмування SQL. Досліджено вплив цих проблем на результативність виконання. Проведена оцінка існуючих рішень та досліджень, на базі яких реалізовано новий метод, спрямований на вирішення проблеми виявлення семантичних помилок. Також розроблено евристики для аналізу помилок у інструменті статичного аналізу.

2. Висновок про відповідність роботи дипломному завданню Кваліфікаційна робота освітнього ступеня «магістр» у повній мірі відповідає поставленому завданню як у теоретичній, так і в практичній її частині.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи У вступі обгрунтовується актуальність теми роботи, формулюються мета та завдання дослідження, описується наукова новизна та практична цінність отриманих результатів. У першому розділі здійснено аналіз предметної області, останніх джерел та досліджень. Проаналізовано проблеми, які виникають під час написання правильних запитів. У другому розділі досліджено семантичні помилки. Проведено емпіричні дослідження в сфері навчання SQL та SQL-запитів. У третьому розділі обгрунтовано евристики, які дають змогу реалізувати інструмент для статичного аналізу та забезпечити його ефективність. У четвертому розділі розглянуто питання, що стосуються реалізації програмного засобу на основі прийнятих рішень, а також на основі результатів проаналізовано помилки та фактори їх виникнення. Проведено емпіричне дослідження, спрямоване на доведення працездатності розробленого програмного засобу та його функціональної придатності. Обгрунтована ефективність удосконаленого методу пошуку семантичних помилок у коді на декларативній мові програмування SQL. Розроблено інструмент та надано рекомендації з його застосування.

4. Позитивні сторони роботи Кваліфікаційна робота містить низку інноваційних рішень, зокрема, було доведено доцільність удосконалення методу пошуку семантичних помилок. Запропоновано використовувати евристики які дозволяють статично аналізувати код для виявлення помилок . Удосконалений метод показав свою ефективність під час використання розробленого інструменту аналізу помилок.

5. Негативні сторони роботи Реалізований метод вирішує помилку пошуку семантичних помилок, але має недоліки в складності інтеграції в середовища розробки оскільки сучасні фремворки застосовують не лише код на мові програмування SQL і кількість

описаних помилок недостатня і може бути розширена для збільшення якості коду.

6. Оцінка графічного оформлення та пояснювальної записки роботи Графічне оформлення виконане відповідно до теми кваліфікаційної роботи з дотриманням вимог стандартів. Пояснювальна записка відповідає вимогам стандартів до її оформлення.

7. Відгук про роботу в цілому В цілому кваліфікаційна робота заслуговує позитивної оцінки. Весь матеріал роботи структурований, чіткий та послідовний. Усі розділи роботи є послідовними та логічними, що дозволяє чітко розуміти викладений матеріал у рамках тематики кваліфікаційної роботи. Графічний матеріал дозволяє наочно побачити доцільність та ефективність рішень, які були прийняті для вирішення поставленої задачі.

8.Інші зауваження

9. Оцінка кваліфікаційної роботи Розглянувши позитивні та негативні сторони представленої кваліфікаційної роботи, можна зробити висновок, що вона заслуговує оцінки «відмінно».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)
Говорущенко Тетяна Олександрівна, доктор технічних наук, професор, зав. кафедри комп'ютерної інженерії та інформаційних систем (КІІС) ХНУ

1.12.2023
Дата

(Підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів подібності щодо роботи, продукованими програмно-технічним засобом(ами) перевірки текстів на плагіат.
Назва: «Метод виявлення семантичних помилок у коді на декларативній мові програмування SQL за допомогою статичного аналізу»
Автор: Нетреба Ігор Вікторович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Праворська Наталія Іванівна, кандидат. педагогічних. наук, доцент
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені у розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за два дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи.	
3	Виявлені запозичення не є плагіатом, але частково розміщені у розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того, як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:
Запозичення, виявлені у роботі, є законними і не є плагіатом, оскільки:
1) у тексті дипломної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, бланк завдання), у структурі змісту, назвах розділів/підрозділів тощо та в назвах публікацій у переліку джерел посилання;
2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посиланання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;
3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;
4) виявлені модифікації тексту не впливають на відсоток схожості.
Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає 1.0%. Обсяг запозичень, визначений системою Unicheck виявлення збігів ідентичності/схожості, складає 6.87% і адресується до 926 джерел з інтернету і 74 джерела з бібліотеки, що, з урахуванням наведених обгрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата _____

Завідувач кафедри ІПЗ _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Оксана ЯШИНА

Керівник кваліфікаційної роботи _____ Наталія ПРАВОРСЬКА

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти

Нетреби І.В.

Прізвище, ініціали

факультет ІТ, 2 курс, група ІПЗм-22-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

8.12.2023

дата

підпис