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 командах 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.