Міністерство освіти і науки України
Державний університет «Одеська Політехніка»
Навчально-науковий інститут комп'ютерних систем
Кафедра системного програмного забезпечення

Добрєв Олег Юрійович,
студент групи АС-161

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Мовний транслятор DML команд з реляційної мови SQL на мультимодельну мову ArangoDB Query Language

Спеціальність:
121 – Інженерія програмного забезпечення

Освітня програма:
Інженерія програмного забезпечення

Керівник:
Паулін Олег Миколайович,
док. техн. наук, професор

Одеса – 2021

ЗМІСТ

Міністерство освіти і науки України
Державний університет «Одеська Політехніка»
Навчально-науковий інститут комп'ютерних систем
Кафедра системного програмного забезпечення

Рівень вищої освіти: другий (магістерський)
Спеціальність: 121 – Інженерія програмного забезпечення
Спеціалізація: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Любченко В. В.

«____»____ 20__р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Добрєва Олега Юрійовича, група АС-*161*

1. Тема роботи: Мовний транслятор DML команд з реляційної мови SQL на мультимодельну мову ArangoDB Query Language.
Керівник роботи: Паулін Олег Миколайович, док. техн. наук, професор
затверджені наказом ректора від «25» жовтня 2021 р. № 374-в

2. Зміст роботи: проблеми існуючих рішень для програмної системи; розробка технології синтаксичного аналізу мовного транслятору; специфікація вимог до мовного транслятору; планування програмного проекту; проектування програмного продукту; програмна реалізація програмного продукту; випробування системи та аналіз результатів.

3. Перелік ілюстративного матеріалу: Згідно зі слайдами презентації

4. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

5. Дата видачі завдання: «25» жовтня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання	Примітка
1	Переддипломна практика	30.08.2021	Вик.
2	Вивчення предметної області	22.10.2021	Вик.
3	Опис вимог	27.10.2021	Вик.
4	Проектування архітектури	14.11.2021	Вик.
5	Розробка транслятора	19.11.2021	Вик.
6	Розробка інтерфейсу користувача	21.11.2021	Вик.
7	Тестування	22.11.2021	Вик.
8	Розробка керівництва користувача	24.11.2021	Вик.
9	Висновки	28.11.2021	Вик.

Здобувач вищої освіти _____ О. Ю. Добрєв

Керівник роботи _____ О. М. Паулін

АНОТАЦІЯ

Метою роботи є зменшення часу, необхідного для переносу існуючої реляційної бази даних, створеної на мові SQL, до мультимодельної мови ArangoDB, за рахунок розробки та програмної реалізації мовного транслятора. Технологіями розробки є ANTLR4, як інструмент генерації парсеру, на основі граматики та синтаксичних правил; мова програмування Java, а саме бібліотека Swing, для реалізації інтерфейсу користувача та стандарті бібліотеки «java.lang», «java.util». Як результат роботи, виконана програмна реалізація транслятору з реляційної мови SQL у мультимодельну ArangoDB, яка дозволяє перетворювати запити DML у відповідні команди роботи з документами та графами.

Ключові слова: транслятор, запит, синтаксичний аналізатор, Java, ANTLR, ArangoDB, SQL, DML.

ABSTRACT

The aim is to reduce the time required to transfer the existing relational database created in the SQL language to a multimodal language, ArangoDB, through the development and software implementation of a language translator. Development technologies are ANTLR4 as a parser tool based on grammar and syntactic rules, Java programming language, namely the Swing library for the implementation of the user interface and the library standard "java.lang", "java.util". As a result, the software implementation of the translator from the relational SQL language to the multimodel ArangoDB, which allows you to convert DML queries into appropriate commands for working with documents and graphs.

Keywords: translator, query, parser, Java, ANTLR, ArangoDB, SQL, DML.

ВСТУП

Програмне забезпечення безупинно розвивається, спочатку для зберігання даних використовували таблиці, які підпорядковувалися моделі сутність - зв’язок, яка дозволяє описувати концептуальні схеми за допомогою узагальнених конструкцій блоків. Але цього було недостатньо, тому з підвищенням популярності об’єктно-орієнтованого підходу, програмісти почали розглядати дані, в своїх базах даних, у якості об'єктів. Можна сказати, що якщо дані людини були в базі даних, атрибути цієї особи, такі, як їх адреса, номер телефону, і вік, в даний час належать до цієї людини замість того, щоб мати стосунок до сторонніх даних. Об'єктні бази даних і об'єктно-реляційні бази даних намагаються вирішити цю проблему шляхом створення об'єктно-орієнтованої мови (іноді в якості розширень SQL), яку програмісти можуть використовувати в якості альтернативи до, чисто реляційного SQL. На стороні програмування, бібліотеки, відомі як об'єктно-реляційні відображення, намагаються вирішити ту ж проблему. NoSQL бази даних часто дуже швидкі, не вимагають фіксованих схем таблиць, мають здібність до масштабованості. Мультимодельний клас сучасних баз даних, який створений забезпечити таку ж масштабовану продуктивність систем NoSQL, для оперативної обробки транзакцій (читання і запису), при цьому використовуючи SQL і підтримку ACID транзакцій традиційної системи баз даних. Отже, структури даних стають складнішими, їх стає важко зберігати у табличному (реляційному) представлені, один з варіантів вирішення цієї проблеми, перенести структури даних до мультимодельного представлення (NoSQL). Такі сховища даних слабоструктуровані, що дозволяє зберігати об’єкти одного типу з різними властивостями (атрибутами). Тому, щоб розробити нове нереляційне сховище необхідно знати мову ArangoDB, а розробляємий транслятор допоможе скоротити час, необхідний для міграції реляційної бази даних на мультимодельну.

Метою роботи є – зменшення часу необхідного для переносу існуючої реляційної бази даних створеної на мові SQL, до мультимодельної мови ArangoDB, та розширення функціональності, існуючих трансляторів, шляхом додавання

команд маніпуляції з даним, тобто SQL команди «INSERT», «DELETE», «UPDATE», за рахунок розробки та програмної реалізації мовного транслятора.

У кваліфікаційній роботі використовувалися такі методи дослідження як – аналіз, моделювання, спостереження та порівняння [2]. Далі описано що досліджено цими методами та для чому були обрані саме вони.

- Аналіз – за допомогою цього методу, була розкладена на частини предметна область, а саме як реляційні бази даних можна перетворити до інших баз даних, що програма повинна бути транслятором, які там будуть основні компоненти. Метод був обраний тому що, він дозволяє отримати повне представлення про предметну область.

- Моделювання – за допомогою цього методу, отримана структура та поведінка, з урахуванням значної кількості факторів, програми, тобто архітектура програми, взаємодія між компонентами та як програмою будуть користуватися. Метод був обраний, тому що він дозволяє замінити неіснуючий об`єкт або такий об`єкт дослідження, який неможливо чи недоцільно залучати до експерименту та фактично, є інформаційним зображенням об`єкта, на модель.

- Спостереження – за допомогою цього методу, отримані характеристики про роботу програми, а саме коректність отриманих результатів та час отримання цих результатів. Метод був обраний, тому що він дозволяє отримати узагальнюючі дані, які дозволяють рівномірно відобразити характеристики всієї системи.

- Порівняння – за допомогою цього методу встановлена подібність і розходження програмних аналогів та архітектурних шаблонів проектування. Метод був обраний, тому що він дозволяє порівнювати явища, між якими може існувати визначена об’єктивна спільність.

Наукова новизна одержаних результатів полягає у тому, що в роботі отриманий подальший розвиток, перехід з реляційної структури даних на нереляційну, а саме мультимодельну структуру, зі збереженням таких переваг як – транзакційність і об’єднання даних при вибірці.

Результати роботи транслятора, це запити написані на мові ArangoDB, які являються аналогом реляційних запитів SQL. Рекомендовано використовувати ці запити, під час переходу з реляційної бази даних, яку вже використовують компанії, до мультимодельної ArangoDB.

1 ПРОБЛЕМИ ІСНУЮЧИХ РІШЕНЬ ДЛЯ ПРОГРАМНОЇ СИСТЕМИ

1.1 Опис предметної області

У наш час, програм все більше, потрібно обробляти дуже велику кількість даних. Традиційний спосіб, а саме, табличне (реляційне) представлення, не підходить для цього. Так як, реляційні системи керування базами даних починають давати відповідь на запит дуже довго, або взагалі, дають неправильну відповідь

Тому з’явився новий напрямок - NoSQL [1]. В даний час не існує, ні загальноприйнятого визначення цього терміну, ні авторитетного органу, який би запропонував таке визначення, тому можна лише обговорювати деякі загальні властивості баз даних, що відносяться до категорії NoSQL.

Для початку очевидний факт: бази даних NoSQL не використовують мову SQL. Деякі з них мають свою мову запитів, і їх доцільно було б зробити схожим на SQL, щоб їх легше було вивчити. Однак до сих пір не була реалізована жодна мова, яка би досягла хоча б тієї ж ступеня гнучкості, що і стандартна мова SQL.

Інша важлива властивість цих баз даних полягає в тому, що вони представляють собою проекти з відкритим вихідним кодом. Незважаючи на те, що термін "NoSQL" часто застосовується до систем із закритим вихідним кодом, існує думка, що NoSQL - це феномен з відкритим вихідним кодом.

Бази даних NoSQL працюють без схеми, дозволяючи вільно додавати поля в базу даних, без попереднього зміни структури. Це дуже важливо для баз даних з неоднорідними даними та призначеними для користувача полями.

Однією реалізацією NoSQL підходу є – ArangoDB.

ArangoDB - багатоцільова відкрита СУБД, що надає гнучкі моделі зберігання документів, графів та даних у форматі ключ-значення. Робота з базою здійснюється через SQL-подібну мову запитів AQL або через спеціальні розширення JavaScript. Засоби зберігання даних відповідають вимогам ACID (атомарність, узгодженість, ізольованість, надійність), підтримують транзакції та забезпечують як горизонтальну, так і вертикальну масштабованість.

У зв’язку з цим, актуальним є розробка методу переходу від реляційного представлення, до мультимодельного. Документоорієнтовані СКБД спеціально призначені для зберігання ієрархічних структур даних, тому вони підходять для проектів, які мають зберігати складні данні, наприклад комп’ютерна гра та інші, а графові СКБД, додатково зберігають зв’язок між сутностями, що у свою чергу дозволяє продумати архітектуру заздалегідь. Отже визначимо, що таке SQL.

SQL є інструментом, призначеним для організації, управління, вибірки і обробки інформації, що міститься в базі даних [3]. Для того щоб отримати інформацію з бази даних, ви запитуєте її у СКБД за допомогою SQL. СКБД обробляє запит, знаходить необхідні дані і повертає їх вам. Процес запиту даних у бази даних і отримання результату називається запитом до бази даних.

SQL використовується для реалізації всіх функціональних можливостей, які СКБД надає користувачеві.

• Визначення даних. SQL дозволяє користувачеві визначити структуру і організацію даних, що зберігаються і взаємини між елементами збережених даних.

• Вибірка даних. SQL дає користувачеві або додатку можливість брати з бази містяться в ній дані і користуватися ними.

• Обробка даних. SQL дозволяє користувачу або додатку змінювати базу даних, тобто додавати в неї нові дані, а також видаляти або відновлювати вже наявні в ній дані.

• Управління доступом. За допомогою SQL можна обмежити можливості користувача по вибірці, додаванню і зміни даних та захистити їх від несанкціонованого доступу.

• Спільне використання даних. SQL застосовується для координації спільного використання даних користувачами, що працюють одночасно, з тим щоб зміни, що вносяться одним користувачем, не приводили до випадкового знищення змін, внесених приблизно в той же час іншим користувачем.

• Цілісність даних. SQL дозволяє забезпечити цілісність бази даних, захищаючи її від руйнування через неузгоджені зміни або відмови системи.

Отже, визначивши, що таке SQL, стає зрозуміло що буде потрапляти на вхід транслятору – SQL запит, який необхідно перетворити у аналог запиту ArangoDB, яка являється мультимодельною СКБД.

Однією з концепцій в мультимоделних базах даних є документ. База даних зберігає і витягує документи в форматах XML, JSON, BSON і т.д.. Ці документи представляють собою самоописані ієрархічні деревоподібні структури даних, які можуть складатися з асоціативних масивів, колекцій і скалярних значень. Документи зберігаються приблизно однаково, але не обов'язково повинні бути однаковими. Документні бази даних зберігають документи в якості значень в сховищах типу "ключ-значення"; документні бази даних можна інтерпретувати як сховища типу "ключ-значення", в яких значення допускає перевірку [5].

Для того щоб, зробити перетворення необхідно створити транслятор.

Транслятор - обслуговуюча програма, яка перетворює вихідну програму, надану на вхідній мові програмування, в робочу програму, представлену на об'єктному мовою. Наведене визначення відноситься до всіх різновидів транслюючих програм. Однак у кожної з таких програм можуть бути свої особливості щодо організації процесу трансляції .

Для перетворення з однієї мови в іншу, транслятору необхідно мати певний синтаксис вихідної мови, та аналізатор цього синтаксису.

Синтаксис – сукупність правил деякої мови, що визначають формування його елементів. Інакше кажучи, це сукупність правил освіти, семантично значущих послідовностей символів в даній мові. Синтаксис задається за допомогою правил, які описують поняття деякої мови.

Синтаксичний аналізатор - компонента компілятора, що здійснює перевірку вихідних операторів на відповідність синтаксичним правилам і семантиці даної мови програмування [5].

Отже, на вхід транслятору необхідно подавати певний синтаксис мови і запит SQL, і програма перетворить вхідний запит до запиту ArangoDB.

1.2 Аналіз проблем користувача та програмних аналогів

Для аналізу проблем користувача, використано канву ціннісної пропозиції, іншими словами value proposition canva. Це – короткий і простий список переваг, які отримує споживач при покупці вашого продукту. Канва опирається на алгоритм, який складається з двох логічних блоків – профілю клієнта та ціннісної пропозиції компанії.

- Профіль клієнта
    1) Вигоди – переваги, яких очікує та потребує клієнт, що порадує клієнтів, і те, що може збільшити ймовірність прийняття цінної пропозиції.
    2) Біль – негативні переживання, емоції та ризики, які відчуває клієнт у процесі виконання роботи.
    3) Робота клієнта – функціональні, соціальні та емоційні завдання, які клієнти намагаються виконати, проблеми, які вони намагаються вирішити, і потреби, які вони хочуть задовольнити.

Профіль клієнта слід створити для кожного сегмента клієнтів, оскільки кожен сегмент має певні переваги, проблеми та завдання.

- Ціннісна пропозиція компанії
    1) Створювачі прибутку – як продукт або послуга, створюють переваги для клієнтів і як вони пропонують додаткову цінність для клієнта.
    2) Знеболюючі – опис того, як саме продукт або послуга полегшує біль клієнта.
    3) Продукти та послуги – продукти та послуги, які створюють прибуток і полегшують біль, яка лежить в основі створення цінності для клієнта.

Далі наведений аналіз проблем, за допомогою канви.

Сегмент споживача: Програмісти SQL які хочуть перенести реляційну базу даних до мультимодельної або вивчити синтаксис Arango Query Language (AQL)

- Цінності для клієнта:
    1) Зручність – можна використовувати вже створені DML запити на SQL

до запиту AQL

2) Економія часу – не потрібно самому писати запити, можна просто завантажити файл з запитом у програму.

- Яку задачу вирішує клієнт:
  1) Перехід бази даних з SQL до ArangoDB
  2) Вивчити AQL
- Які проблеми є у клієнта:
  1) Відсутність у розроблених трансляторів, можливість коректно перекладати SELECT запити, з конструкцією JOIN
  2) Для наглядного вивчення синтаксису AQL, не достатньо однієї документації
- Який товар чи послуга пропонується?
  1) Послуга з переводу запиту SQL, на аналогічний запит AQL
  2) Можливість завантажити вже створені скрипти у програму
- Що створює цінність?
  1) Можливість перекладати запити вибірки даних з об’єднанням
  2) Користувач отримує практичні знання в AQL
- Що вирішує проблему?
  1) Об’єднання практики та теорії для мови AQL
  2) Підтримка трансляції запитів JOIN

Відобразимо отримані данні на канві (див. рис 1.1). Вона допоможе виділити ключові переваги товару та пояснити, чому клієнт буде користуватись розробленим транслятором.

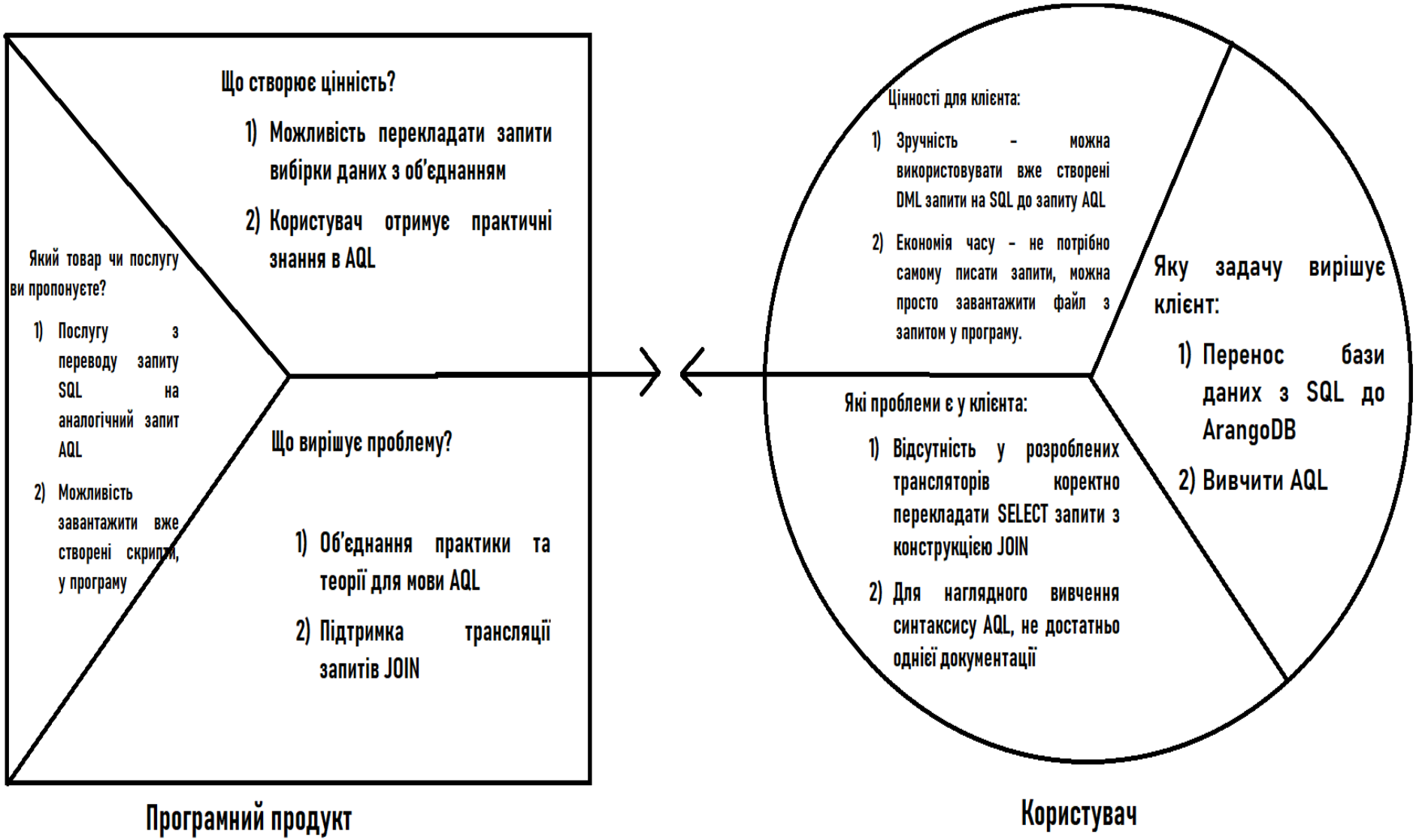

Рисунок 1.1 – Канва ціннісної пропозиції

Для проведення аналізу існуючих рішень, використано стратегічну канву, її також називають - стратегія блакитного океану. Ця стратегія описує бурхливе зростання та високу прибутковість компаній, які можуть генерувати продуктивні бізнес-ідеї, створюючи попит, що не існував раніше, на новому ринку (блакитний океан), де практично відсутні конкуренти, замість того, щоб конкурувати з безліччю конкурентів на малоприбуткових ринках (червоний океан).

Було виділено такі фактори цінності – коректність результату, запити SELECT з об’єднанням, час відповіді транслятору, INSERT запити, DELETE запити, UPDATE запити, SELECT запити.

Далі необхідно описати існуючі рішення

1) «Arango Translate» - програмне забезпечення, яке написано на Java, дозволяє тільки перетворювати запити типу SELECT, які є простими, тобто без агрегації, з'єднань і так далі.

2) «SQL_to_ArangoDB» - програмне забезпечення, яке написано на C#, має широкий функціонал, дозволяє транслювати запити SELECT з агрегацією, з'єднанням і т.д. Так само його відмінна риса, що можна звернутися до існуючої БД на MySQL, зробити запит на SQL і отримати відповідь з цієї БД в форматі ArangoDB.

Використовуючи шкалу від 1 до 10, де 1 – це реалізація відсутня, а 10 – реалізовано ідеально, без помилок і у повному обсязі, створена стратегічна канва (див. рис. 1.2)

За допомогою створеної стратегічної канви необхідно виділити сильні та слабі сторони програмних аналогів та розробляємого транслятору. Результат порівнянь програмних забезпечень, представлений у таблиці 1.1.

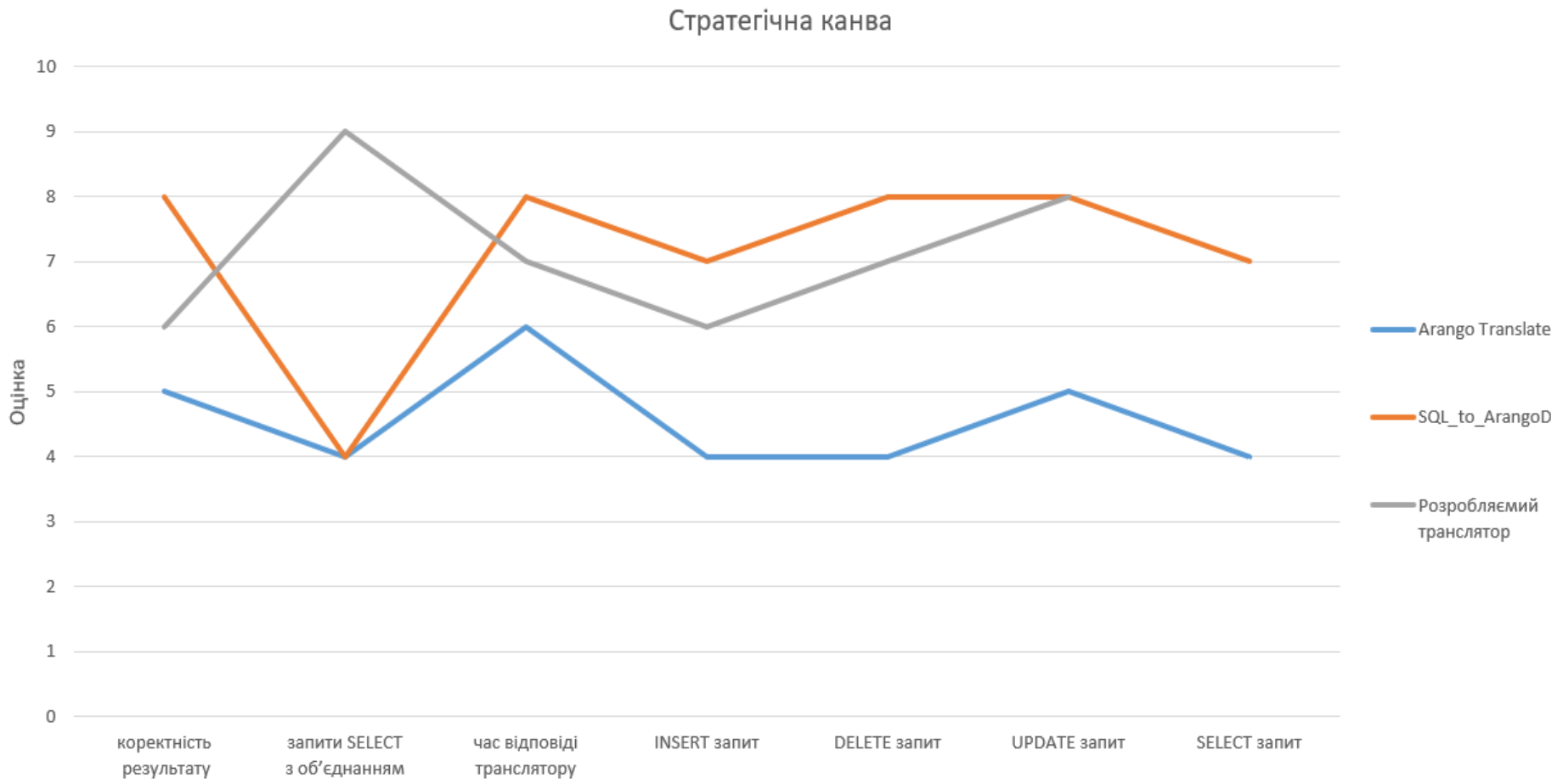

Рисунок 1.2 – Стратегічна канва

Таблиця 1.1 – Порівняння аналогів

Сервіси / Критерії	Розробляємий транслятор	Arango Translate	SQL_to_Arango DB
Коректність результату	-	-	+
Запити SELECT з об’єднанням	+	-	-
Час відповіді транслятору	+	+	+
INSERT запити	+	-	-
DELETE запити	+	-	+
UPDATE запити	+	+	+
SELECT запити	+	-	-

Отож, як можна побачити з таблиці 1.1, основний конкурент розробляємого транслятора, це програмне забезпечення «SQL_to_ArangoDB», однак у нього слабо реалізовані запити «INSERT», «SELECT».

Після проведеного аналізу можна сформувати задачу дослідження – розробити транслятор з SQL в ArangoDB, який зможе перекладати запити вибірки даних з об’єднанням.

2 РОЗРОБКА ТЕХНОЛОГІЇ СИНТАКСИЧНОГО АНАЛІЗУ МОВНОГО ТРАНСЛЯТОРУ

2.1 Опис процесу трансляції та формальних граматик

Перед проектуванням програмного продукту, спочатку необхідно повністю описати процес отримання кінцевого результату користувачем, а саме перетворений вхідний запит на мові SQL, до мови ArangoDB.

Коли користувач відкрив програмний продукт, то він може написати запит самостійно у редакторі, або завантажити його до програми з носія, де зберігається файл у форматі «.sql». Далі, він натискає на кнопку і запит відправляється до транслятору. Потім відбувається етап трансляції, для того щоб, його описати, необхідно спочатку визначити, як організована високорівнева мова програмування. Мова програмування, як і будь-яка складна система, визначається через ієрархію понять, що задає взаємозв'язок між його елементами. Ці поняття пов'язані між собою відповідно до синтаксичних правил. Кожна з програм, побудована за цими правилами, має свою ієрархічну структуру. У зв'язку з цим для всіх мов і їх програм, можна додатково виділити наступні загальні риси: кожна мова має містити правила, що дозволяють породжувати програми, відповідні цій мові, або розпізнавати відповідність, між написаними програмами і заданим мовою. Зв'язок структури програми з мовою програмування називається синтаксичним відображенням [4].

Синтаксичні правила описуються за допомогою метамови, а саме – формальних граматик.

Формальна граматика - це множина G (формула 2.1), а також правила до неї (формула 2.2 – 2.4).

$$G = \{N, T, P, S\}, \tag{2.1}$$

$$T \cap N = \emptyset, \tag{2.2}$$

$$S \in N, \tag{2.3}$$

$$(T \cup N)^+ \times (T \cup N)^* \quad (2.4)$$

де: T — алфавіт термінальних символів, терміналів. Термінальні символи є алфавітом мови.

N — алфавіт нетермінальних символів, нетерміналів. Нетермінали не входять в алфавіт мови.

S — аксіома, спеціально виділений нетермінальний символ, з якого починається опис граматики.

P — правила виводу, скінченна підмножина множини.

Алфавіт — скінченна множина символів. ε — порожній ланцюжок, слово, послідовність. Алфавітом є об'єднання, перетин, різниця алфавітів. Нехай T — алфавіт, тоді:

T^+ — множина усіх можливих послідовностей, що складені з елементів цього алфавіту, крім порожньої послідовності ε.

T^* — множина усіх можливих послідовностей, що складені з елементів цього алфавіту, будь-якої довжини (формула 2.5).

$$T^* = T^+ \cup \{\varepsilon\} \quad (2.5)$$

T^k — множина усіх можливих послідовностей, що складені з елементів цього алфавіту, довжини не більше k.

Мова в алфавіті T – це множина ланцюжків скінченної довжини в цьому алфавіті. Зрозуміло, що кожна мова в алфавіті T, є підмножиною множини T*.

Для створення транслятору необхідна контекстно-вільна граматика у вигляді розширеної форми Бекуса - Наура (РБНФ).

Контекстно-вільна граматика – по Хомському, це другий тип граматики, Всі правила мають вигляд по формулі 2.6

$$\alpha \rightarrow \beta \quad (2.6)$$

$$\alpha \in N$$

$$\beta \in (T \cup N)^*$$

де: T — алфавіт термінальних символів, терміналів Термінальні символи є алфавітом мови.

N — алфавіт нетермінальних символів, нетерміналів. Нетермінали не входять в алфавіт мови.

Тобто, в усіх правилах цього виду, зліва стоїть тільки один нетермінал. Тому вони і контекстно вільні, бо на використання правила для даного нетерміналу, не впливають символи що оточують його. Ці символи називають контекстом.

Розширена форма Бекуса - Наура (РБНФ) - формальна система визначення синтаксису, в якій одні синтаксичні категорії послідовно визначаються через інші. Використовується для опису контекстно-вільних формальних граматик. Запропоновано Ніклаус Віртом. Є розширеною переробкою форм Бекуса - Наура, відрізняється від БНФ більш «ємними» конструкціями, що дозволяють при такій же здатності, спростити і скоротити в обсязі опис (рис 2.1).

Число = ["+" | "-"] *Натуральне_число* ["." *[Натуральне_число]*] [("e" | "E") ["+" | "-"] *Натуральне_число*].

Натуральне_число = *Цифра {Цифра}*.

Цифра = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".

Ідентифікатор = *Буква{Буква | Цифра | "_"}*.

Рисунок 2.1 - Приклад граматики створеної за допомогою РБНФ

Використовуючи РБНФ, створені правила для розбору запитів, які маніпулюють з даними таблиць, оновлюють або показують. Наступний етап, розбір запиту, по правилам, у результаті якого отримують синтаксичне дерево, це дерево необхідно обійти обробником, зліва на право, знаходячи термінальні символи та в

програмі змінювати на аналоги ArangoDB. Увесь процес трансляції зображений на рисунку 2.2.

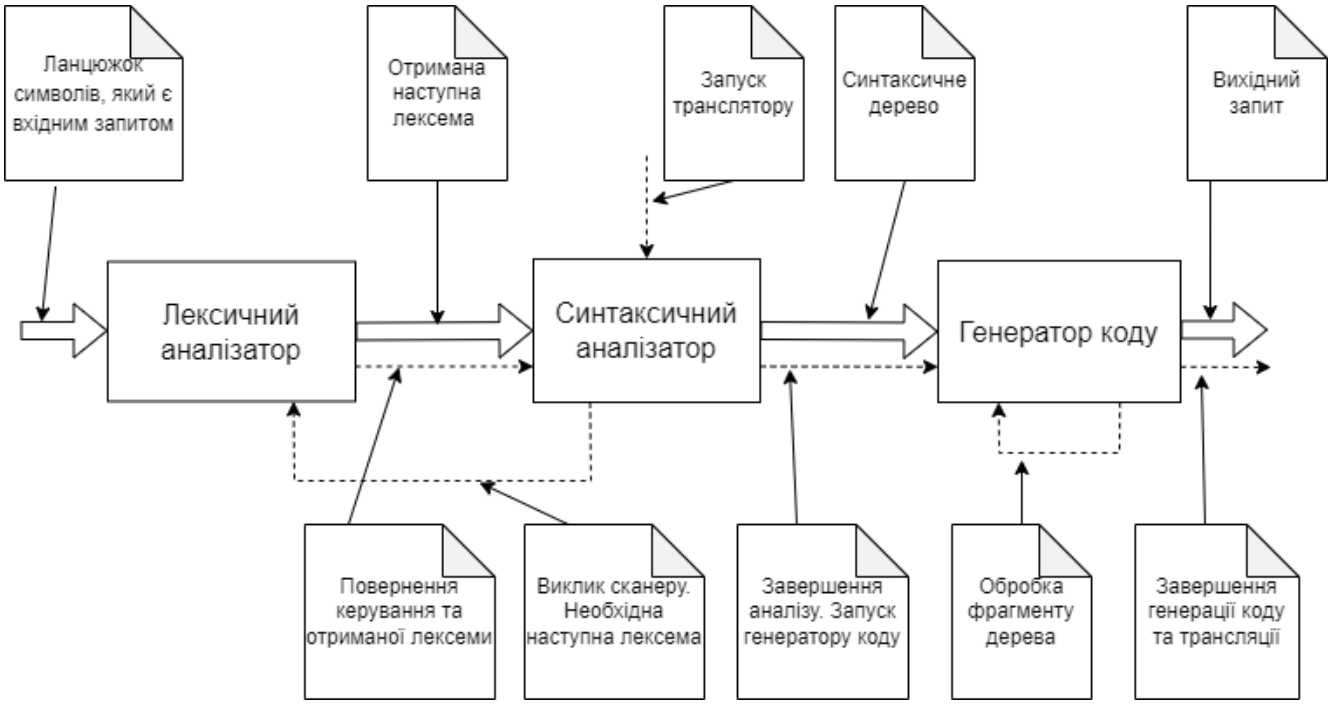

Рисунок 2.2 – Схема транслятора

Потім, результат виводиться до користувача, який він може зберегти у форматі «.json»

2.2 Опис існуючих алгоритмів для синтаксичного аналізу

У попередньому розділі був описаний процес трансляції та її модулі. Один з модулів це – синтаксичний аналізатор, для нього необхідний більш детальний опис.

Перед синтаксичним аналізатором стоять дві основні задачі: перевірити правильність конструкцій програми, яка представляється у вигляді вже виділених слів вхідної мови; перетворити її на вигляд, зручний для подальшої семантичної обробки та генерації коду. Одним із способів такого представлення є дерево синтаксичного аналізу. Існує два методи обходу синтаксичного дерева – низхідний та висхідний.

Низхідний синтаксичний аналіз – це завдання побудови дерева розбору для вхідного рядка, починаючи з кореня та описуючи вузли дерева розбору у прямому порядку обходу.

Висхідний синтаксичний аналіз, відповідає побудові дерева розбору для вхідного рядка, починаючи з листів та йдучи до кореня. Можна розглядати висхідний синтаксичний аналіз, як процес згортки рядка до цільового символу граматики. На кожному кроці згортки, певний підрядок, який відповідає тілу правила, замінюється нетерміналом з лівої частини цього правила. Ключові рішення, що приймаються в процесі висхідного синтаксичного аналізу – коли виконувати згортку та яке правило застосовувати. Оскільки за визначенням, згортка є крок, зворотний породженню, то мета висхідного синтаксичного аналізу полягає у побудові породження зворотному порядку.

Далі необхідно пояснити, яка в системі використовується граматика, вона називається LL(k). Ця граматика розшифровується як – Left Left, ланцюжок символів зчитується зліва направо та формується лівосторонній висновок, а (k) > 0 – це якщо на кожному кроці виводу, для однозначного вибору чергової альтернативи, достатньо знати символ на верхівці стека та розглянути перші k символів від поточного положення, у вхідному ланцюжку символів [12].

Для LL(k)-граматик відомі такі властивості:

- всяка LL(k)-граматика для будь-якого k > 0, є однозначною;
- існує алгоритм, що дозволяє перевірити, чи є задана граматика LL(k)-граматикою для певного k.

У системі використовується граматика LL(1), а вона в свою чергу, краще обходиться за допомогою методу рекурсивного спуску (див. рис. 2.3), який являється низхідним алгоритмом, тому був обраний низхідний синтаксичний аналіз. Програма синтаксичного аналізу, є методом рекурсивного спуску, складається з набору процедур, по одній для кожного нетерміналу. Робота програми починається з виклику процедури для стартового символу та успішно закінчується, у разі сканування всього вхідного рядка.

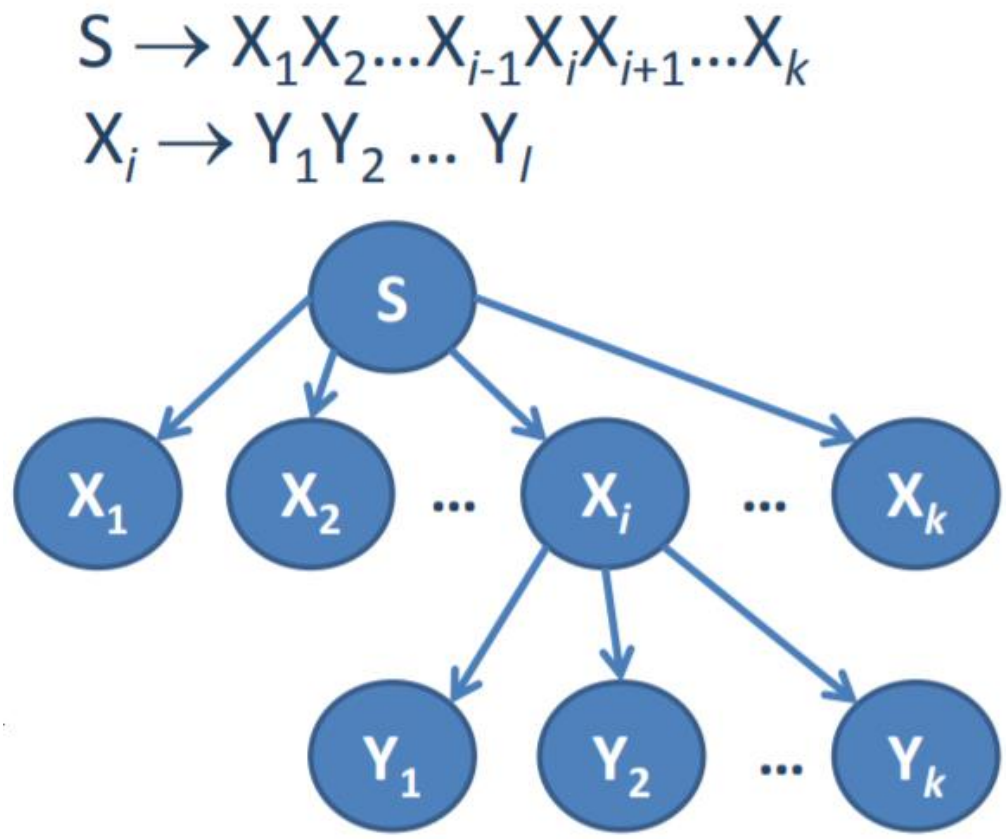

Рисунок 2.3 – Приклад методу рекурсивного спуску

Даний метод необхідно доробити, для того щоб, підтримати вибірку даних з об’єднанням та збільшити правильність результатів.

2.3 Модифікація методу рекурсивного спуску для синтаксичного аналізу

Як було описано у попередньому пункті, необхідно доробити метод синтаксичного спуску, для того щоб отримати найкращі результати. По-перше, необхідно розділити основні процедури на – вхід та вихід. По-друге, необхідно створити окремий компонент аналізу, який відповідає за вибірку даних з об’єднанням. Далі ці пункти, описані більш детально.

Отже, для забезпечення правильності результату, необхідно поділити процедуру нетерміналу на дві логічні частини. Перша частина – це вхід до

процедури, а друга – це вихід з процедури. Обумовлено це тим що, вкладених елементів у вхідному запиті, може бути дуже багато, тому необхідно додавати деяку частину запиту під час виходу з нетерміналу. Приклад процедури наведено на рисунку 2.4.

```
@Override
public void enterUpdatedElement(MySqlParser.UpdatedElementContext ctx) {
}

@Override
public void exitUpdatedElement(MySqlParser.UpdatedElementContext ctx) {
    updatedEl.add(ctx.fullColumnName().getText()+": \""+ctx.expression().getText()+"\"");
}
```

Рисунок 2.4 – Приклад, як основну процедуру необхідно ділити на вхід та вихід

Другий етап, створити модуль, який буде відповідати конкретно за запити вибірки з об’єднанням даних. Це необхідно для того щоб, мати шаблон запиту та швидко завдяки цьому перекладати запити на мову ArangoDB. Далі показана діаграма послідовності трансляції запиту для вибірки з об’єднанням даних (див. рис. 2.5).

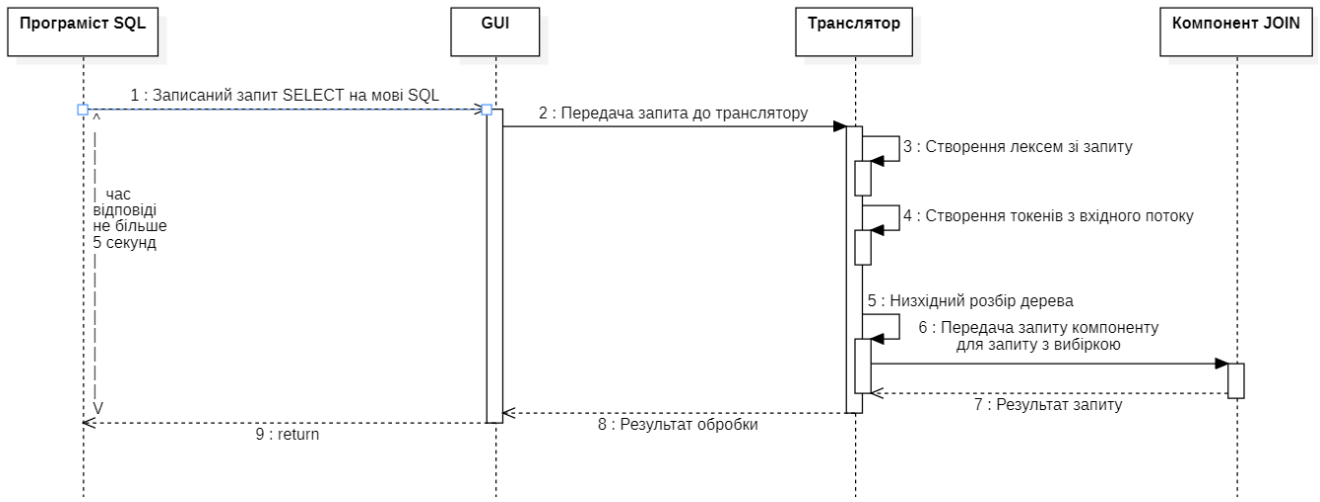

Рисунок 2.5 – Діаграма послідовності запиту вибірки даних з об’єднанням

На діаграмі послідовності зображені основні кроки обробки запиту. Спочатку програміст SQL вводить запит до програми, цей запит завдяки графічному інтерфейсу передається до транслятору. Він спочатку створює лексеми вхідної строки, далі з цих лексем утворюються токени (одиниця синтаксичного аналізу). Далі завдяки методу рекурсивного спуску, оброблюється запит і якщо він має у собі конструкцію JOIN, то такий запит передається до відповідного компоненту. Завдяки вже створеним шаблонам, запит оброблюється швидше та якісніше. На виході, отримуємо відповідний запит на мові ArangoDB.

3 СПЕЦИФІКАЦІЯ ВИМОГ ДО МОВНОГО ТРАНСЛЯТОРУ

3.1 Загальна характеристика продукту

Продукт має виконувати перетворення, або іншими словами трансляцію, з мови SQL, а саме її реалізації у вигляді MySQL, до мультимодельної мови AQL з її реалізацією – ArangoDB. Сам продукт не складний у використанні, так як має простий інтерфейс користувача, але користувач повинен знати мову SQL, інакше програма буде не ефективна для нього.

Функціональність продукту полягає перекладі DML команд SQL, а саме - прості SELECT, SELECT з об’єднанням, INSERT, UPDATE, DELETE. Інші команди не можуть бути перекладені, так як вони не мають відповідних аналогів у концепції мови NoSQL.

Для формалізації вимог до розроблюваної системи було проведено збір та аналіз інформації на основі вимог, сформованих майбутніми користувачами системи. Користувачів системи доцільно розбити на тільки одну категорію:

«Програміст SQL» - людина яка вміє створювати SQL запити, за допомогою СКБД MySQL. Транслятор повинен перетворювати різні види команд, а саме керування даними у таблицях, та отримання цих даних.

Обмеження для проектування програмного продукту:

1. Необхідно мати, або встановити Java SE Development Kit 8
2. Необхідно мати, або встановити будь-яку IDE, що дозволяє роботу с Java

Технічні (апаратні) обмеження зображені у табл. 3.1:

Таблиця 3.1 – Системні вимоги

Вимоги	Мінімальне значення	Рекомендоване значення
Платформа	Windows 7	Windows 10
Оперативна пам'ять	128 Мб	512 Мб і більше

Продовження таблиці 3.1

Вільний простір на жорсткому диску	100 Мб	512 Мб і більше
Процесор	Core i3	Core i5 і більше
Контролер	Клавіатура, миша	Клавіатура, миша
Розрядність	32	64

Отже, приведені обмеження не великі у нас час, тому користувач зможе використовувати розробляєму систему без негативного досвіду.

Для використання програми, так само як і для проектування, необхідно мати встановлену Java, не менш як восьмої версії.

3.2 Функціональні вимоги

Розглянемо основні функціональні вимоги, що пред'являються до розроблюваного транслятору. Функціональні вимоги дають чітке уявлення про вирішення проблеми, підвищуючи ефективність розробки системи та оцінюючи вартість альтернативних шляхів проектування. Дані вимоги можуть включати розрахунки, технічні деталі, маніпулювання та обробку даних, а також інші специфічні функції, які визначають, що система повинна виконувати. Функціональні вимоги керують архітектурою додатку системи, тоді як нефункціональні вимоги керують технічною архітектурою системи. Ці вимоги є вказівкою для випробувачів для верифікації (якісної оцінки) кожної технічної вимоги. Формально опис функціональності та поводження системи наведено на рис. 3.1.

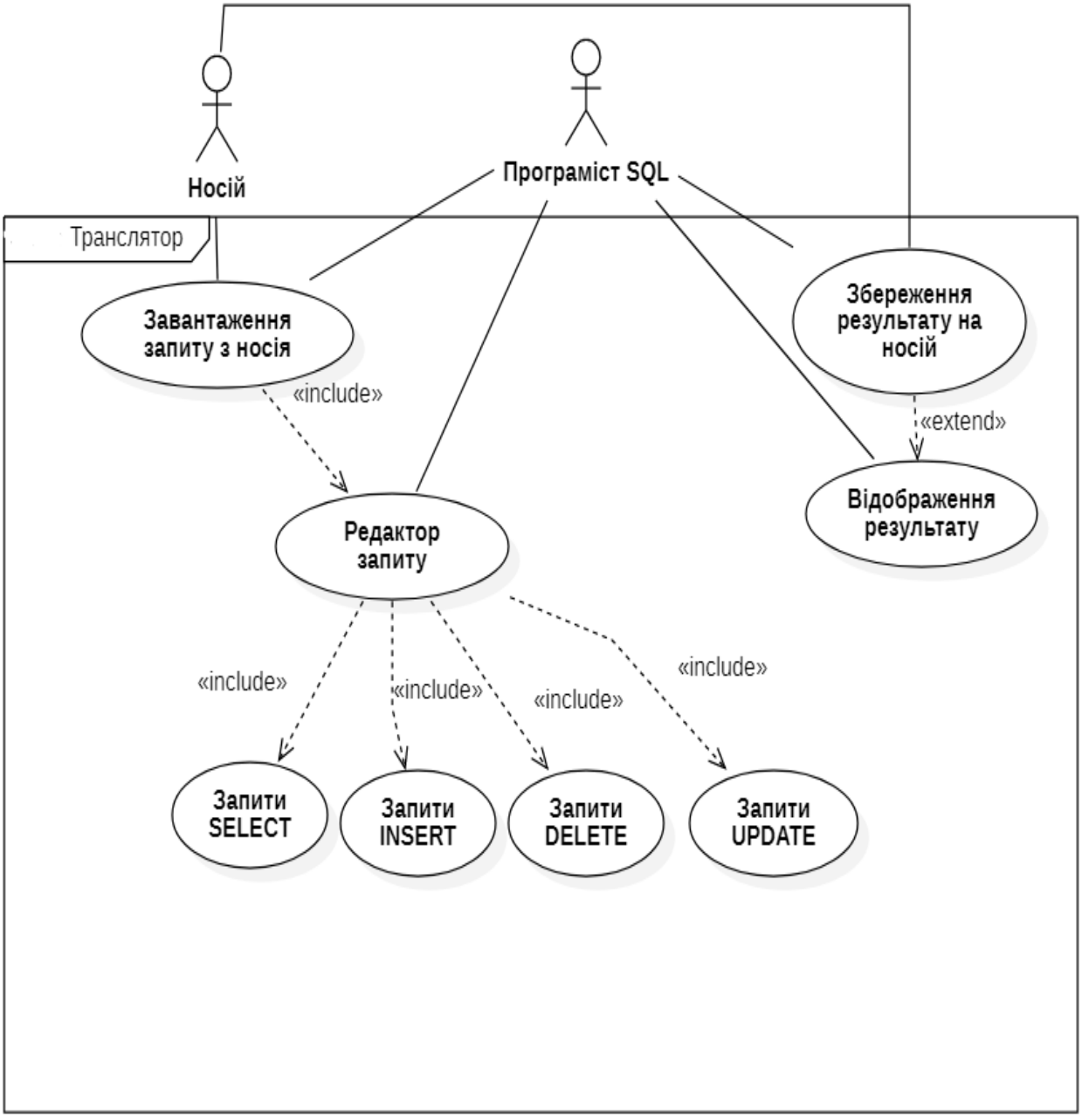

Рисунок 3.1 – Діаграма варіантів використання проектованої системи

Система повинна забезпечувати завантаження запиту з носія, редагування запиту, підтримувати різні типи запитів, виводити результат трансляції користувачам, а також можливість зберігати результат на носій.

Далі наведено повний опис варіантів. В табл. 3.2 знаходиться опис варіанту використання, який надає користувачу можливість ввести запит на мові SQL у систему.

Таблиця 3.2 – Варіант використання «Редактор запиту»

Складова варіанту використання	Опис
Основне діюче лице	Програміст SQL
Область дії	Система
Рівень	Користувача
Учасники та інтереси	Програміст SQL – хоче відредагувати запит і відправити транслятору
Мінімальні гарантії	Виводиться повідомлення про невдачу
Гарантія успіху	Програміст SQL відправив запит транслятору
Основний сценарій	1. Програміст SQL формує запит на мові SQL, система підтверджує 2. Система виконує перевірку синтаксису вхідного запиту 3. Запит пройшов перевірку на синтаксис, тому система передає його до транслятору
Розширення	3а. Синтаксис з помилками 3а1. Програміст SQL заново формує запит – перехід до п.1 3а2. Програміст SQL не бажає змінювати запит – кінець прецеденту.

В табл. 3.3 знаходиться опис варіанту використання, який надає користувачу можливість завантажити файл у форматі «.sql» в систему, для того щоб відобразити запит з цього файлу у редакторі.

Таблиця 3.3 – Варіант використання «Завантаження запиту з носія»

Складова варіанту використання	Опис
Основне діюче лице	Програміст SQL
Область дії	Система
Рівень	Користувача
Учасники та інтереси	Програміст SQL – хоче завантажити запит і відправити транслятору
Мінімальні гарантії	Виводиться повідомлення про невдачу
Гарантія успіху	Програміст SQL відправив запит транслятору
Основний сценарій	1. Програміст SQL обирає файл з носія, який має зберігає запит на мові SQL та відкриває його у системі, система підтверджує 2. Система виконує перевірку синтаксису вхідного запиту 3. Запит пройшов перевірку на синтаксис, тому система передає його до транслятору
Розширення	1а. Програміст SQL бажає відригувати запит 1а1. Програміст SQL редагує запит – перехід до п.2 3а. Синтаксис з помилками 3а1. Програміст SQL заново формує запит – перехід до п.1 3а2. Програміст SQL не бажає змінювати запит – кінець прецеденту.

В табл. 3.4 знаходиться опис варіанту використання, який дозволяє обробляти запити на зміну даних які зберігаються у реляційних таблицях, а саме команду «INSERT».

Таблиця 3.4 – Варіант використання «Запити INSERT»

Складова варіанту використання	Опис
Основне діюче лице	Транслятор
Область дії	Система
Рівень	Системи
Учасники та інтереси	Транслятор – хоче перевірити запити і відправити результат
Мінімальні гарантії	Виводиться повідомлення про невдачу
Гарантія успіху	Система транслює запит у коректний результат
Основний сценарій	1. Транслятор починає обробку запиту на мові SQL 2. Транслятор перевіряє наяву ключових слів із запиту «INSERT», у сховищі, а також аналізує дотримання семантики запиту «INSERT» 3. Система транслює запит у коректний результат
Розширення	3а. Синтаксис або семантика з помилками 3а1. Система передає повідомлення про помилку – кінець прецеденту

В табл. 3.5 знаходиться опис варіанту використання, який дозволяє обробляти запити на зміну даних які зберігаються у реляційних таблицях, а саме команду «UPDATE».

Таблиця 3.5 – Варіант використання «Запити UPDATE»

Складова варіанту використання	Опис
Основне діюче лице	Транслятор
Область дії	Система
Рівень	Системи

Продовження таблиці 3.5

Складова варіанту використання	Опис
Учасники та інтереси	Транслятор – хоче перевірити запити і відправити результат
Мінімальні гарантії	Виводиться повідомлення про невдачу
Гарантія успіху	Система транслює запит у коректний результат
Основний сценарій	1. Транслятор починає обробку запиту на мові SQL 2. Транслятор перевіряє наяву ключових слів із запиту «UPDATE», у сховищі, а також аналізує дотримання семантики запиту «UPDATE» 3. Система транслює запит у коректний результат
Розширення	3а. Синтаксис або семантика з помилками 3а1. Система передає повідомлення про помилку – кінець прецеденту

В табл. 3.6 знаходиться опис варіанту використання, який дозволяє обробляти запити на зміну даних які зберігаються у реляційних таблицях, а саме команду DELETE».

Таблиця 3.6 – Варіант використання «Запити DELETE»

Складова варіанту використання	Опис
Основне діюче лице	Транслятор
Область дії	Система
Рівень	Системи
Учасники та інтереси	Транслятор – хоче перевірити запити і відправити результат
Мінімальні гарантії	Виводиться повідомлення про невдачу

Продовження таблиці 3.6

Складова варіанту використання	Опис
Гарантія успіху	Система транслює запит у коректний результат
Основний сценарій	1. Транслятор починає обробку запиту на мові SQL 2. Транслятор перевіряє наяву ключових слів із запиту «DELETE», у сховищі, а також аналізує дотримання семантики запиту «DELETE» 3. Система транслює запит у коректний результат
Розширення	3а. Синтаксис або семантика з помилками 3а1. Система передає повідомлення про помилку – кінець прецеденту

В табл. 3.7 знаходиться опис варіанту використання, який дозволяє обробляти запити на отримання даних які зберігаються у реляційних таблицях, а саме така команда як «SELECT».

Таблиця 3.7 – Варіант використання «Запити SELECT»

Складова варіанту використання	Опис
Основне діюче лице	Транслятор
Область дії	Система
Рівень	Системи
Складова варіанту використання	Опис
Учасники та інтереси	Транслятор – хоче перевірити запити і відправити результат
Мінімальні гарантії	Виводиться повідомлення про невдачу
Гарантія успіху	Система транслює запит у коректний результат

Продовження таблиці 3.7

Складова варіанту використання	Опис
Основний сценарій	1. Транслятор починає обробку запиту на мові SQL 2. Транслятор перевіряє наяву ключових слів із запиту «SELECT», у сховищі, а також аналізує дотримання семантики запиту «SELECT» 3. Система транслює запит у коректний результат
Розширення	2а. У вхідному запиті є ключове слово JOIN 2а1. Система додатково використовує дані про JOIN із сховища – перехід до п.3 3а. Синтаксис або семантика з помилками 3а1. Система передає повідомлення про помилку – кінець прецеденту

В табл. 3.8 знаходиться опис варіанту використання, який дозволяє користувачу отримати результат трансляції запиту на екран.

Таблиця 3.8 – Варіант використання «Відображення результату»

Складова варіанту використання	Опис
Основне діюче лице	Програміст SQL
Область дії	Система
Рівень	Користувача
Учасники та інтереси	Програміст SQL – хоче отримати результат
Мінімальні гарантії	Нічого не відбувається
Гарантія успіху	Програміст SQL отримав відповідь на запит

Продовження таблиці 3.8

Складова варіанту використання	Опис
Основний сценарій	1. Транслятор повідомляє користувача о завершенні трансляції 2. Транслятор передає результат запита користувачу
Розширення	2а. Користувач вирішив зберегти результат 2а1. Виклик варіанта використання «Зберегти результат на носій»

В табл. 3.9 знаходиться опис варіанту використання, який дозволяє користувачу зберегти результат трансляції запиту в файл формату «.json».

Таблиця 3.9 – Варіант використання «Зберегти результат на носій»

Складова варіанту використання	Опис
Основне діюче лице	Програміст SQL
Область дії	Система
Рівень	Користувача
Учасники та інтереси	Програміст SQL – хоче отримати результат
Мінімальні гарантії	Нічого не відбувається
Гарантія успіху	Програміст SQL отримав файл з ArangoDB запитом
Основний сценарій	1. Програміст SQL створює назву файлу, обирає шлях на носію, та відправляє транслятору 2. Транслятор зберігає файл 3. Транслятор передає повідомлення про збереження
Розширення	2а. Вказаний шлях не існує 2а1. Користувач заново вводить шлях файлу – перехід до п. 1.

3.3 Нефункціональні вимоги до програмної системи

По стандарту ДСТУ IS0/IEC 9126-1:2013, є такі атрибути якості: надійність, ефективність, супроводження, зручність використання. Для цих атрибутів далі наведені нефункціональні вимоги.

- Надійність

1) Якщо користувач формує запит, то система показує повідомлення о результатах, в не менш чим 98% випадків.

2) Якщо користувач намагається завантажити до системи файл з розширенням «.sql», то зміст цього файлу відобразиться у редакторі, в не менш чим 97% випадків.

3) Якщо користувач намагається зберегти файл з розширенням «.json», система збереже цей файл, в не менш чим 98% випадків.

- Ефективність

1) Якщо користувач формує запит, то система показує повідомлення о результатах, в не менш чим 5 секунд.

2) Якщо користувач виводить результат запиту, то система показує його, менш чим 0,5 секунд.

3) Якщо запит перевіряється на синтаксис, то транслятор покаже результат, менш чим за 1 секунду.

4) Якщо користувач намагається завантажити до системи файл з розширенням «.sql», то зміст цього файлу відобразиться у редакторі, не більш чим за 3 секунди.

5) Якщо користувач намагається зберегти файл з розширенням «.json», система збереже цей файл, не більш чим за 5 секунд.

- Супроводження

1) Якщо розробник додає новий синтаксис до транслятору, то система почне роботу с оновленим словником, не більш чим через 10 секунд

2) Якщо розробник додає новий формат для збереження запиту, система починає роботу з новим форматом, не більш чим через 20 секунд

- Зручність використання

1) Якщо користувач вирішує формувати запит, то користувачу це вдасться не більш, чим через 15 секунд.

2) Якщо користувач вирішує завантажити запит з носія, то користувачу це вдасться не більш, чим через 7 секунд.

3) Якщо користувач вирішує вивести результат запиту, то йому це вдасться не більш, чим через 3 секунди.

4) Якщо користувач вирішує зберегти результат запиту, то йому це вдасться не більш, чим через 4 секунди.

4 ПЛАНУВАННЯ ПРОГРАМНОГО ПРОЕКТУ

4.1 Розрахунок тривалості проекту

Для того щоб, розрахувати тривалість проекту використаний метод PERT[6]. Метод PERT являється інструментом, який розраховує очікуваний час тривалості проекту або процесу. Необхідно розрахувати очікувану тривалість кожного процесу, а потім використовуючи ці данні можна знайти очікувану тривалість проекту. Для цього використаємо формулу 4.1

$$t_e = \frac{a + 4m + b}{6} \tag{4.1}$$

де a – оптимістичний час виконання процесу,

m – найбільш імовірний час виконання процесу,

b – песимістичний час виконання процесу,

t_e – середньозважений час процесу.

Після знаходження тривалості процесу, потрібно знайти відхилення кожного процесу, а також проекту у цілому (формула 4.2 -4.3).

$$\sigma_e = \frac{b - a}{6}, \tag{4.2}$$

$$\sigma_T = \sqrt{\sum \sigma_e^2} \tag{4.3}$$

де a – оптимістичний час виконання процесу,

σ_e – відхилення часу процесу,

b – песимістичний час виконання процесу,

σ_T – відхилення часу проекту.

Отримані результати записані у таблиці 4.1

Таблиця 4.1 – Розрахунки тривалості процесів

Назва процесу	m, днів	a, днів	b, днів	t_e, днів	σ_e, днів
Вивчення предметної області	5	4	8	5,33	0,66
Опис вимог	17	10	20	14,33	1,66
Проектування архітектури	9	5	11	8,66	1
Розробка транслятора	13	10	25	13,16	2,5
Розробка інтерфейсу користувача	6	2	8	5,66	1
Тестування	4	1	6	3,83	0,83
Розробка керівництва користувача	5	1	6	4,5	0,83
Висновки	4	1	5	3,66	0,66

Отримані розрахунки σ_e , дозволяють знайти відхилення тривалості проекту $\sigma_T = 3{,}64$. Далі для знаходження імовірності виконання проекту у заданий строк, знайдена величина стандартних відхилень (формула 4.4)

$$Z = \frac{T - T_e}{\sigma_T} \tag{4.4}$$

де Z – кількість стандартних відхилень від середньої величини,

T – тривалість проекту по графіку,

T_e – тривалість критичного шляху,

σ_T – відхилення часу проекту.

При вихідних даних $T = 63$, $T_e = 59.16, \sigma_T = 3{,}64$, отримаємо значення величини $Z = 1.05$, використовуючи таблицю накопленого нормального розподілення з Z > 0, то імовірність виконання проекту у поставлений період дорівнює 0.85

4.2 Оцінка тривалості проекту

Для того щоб оцінити тривалість проекту, його необхідно розбити на логічні частини, для цього використовують декомпозицію процесу. Виконавши її отримано такі частини: переддипломна практика, вивчення предметної області, опис специфікації вимог, проектування архітектури, розробка транслятора, розробка інтерфейсу користувача, тестування транслятора, розробка керівництва користувача, висновки, захист кваліфікаційної роботи. Після того як, отримані терміни, можна використавши їх, створити наглядне відображення за допомогою діаграми Ганта (рис. 4.1).

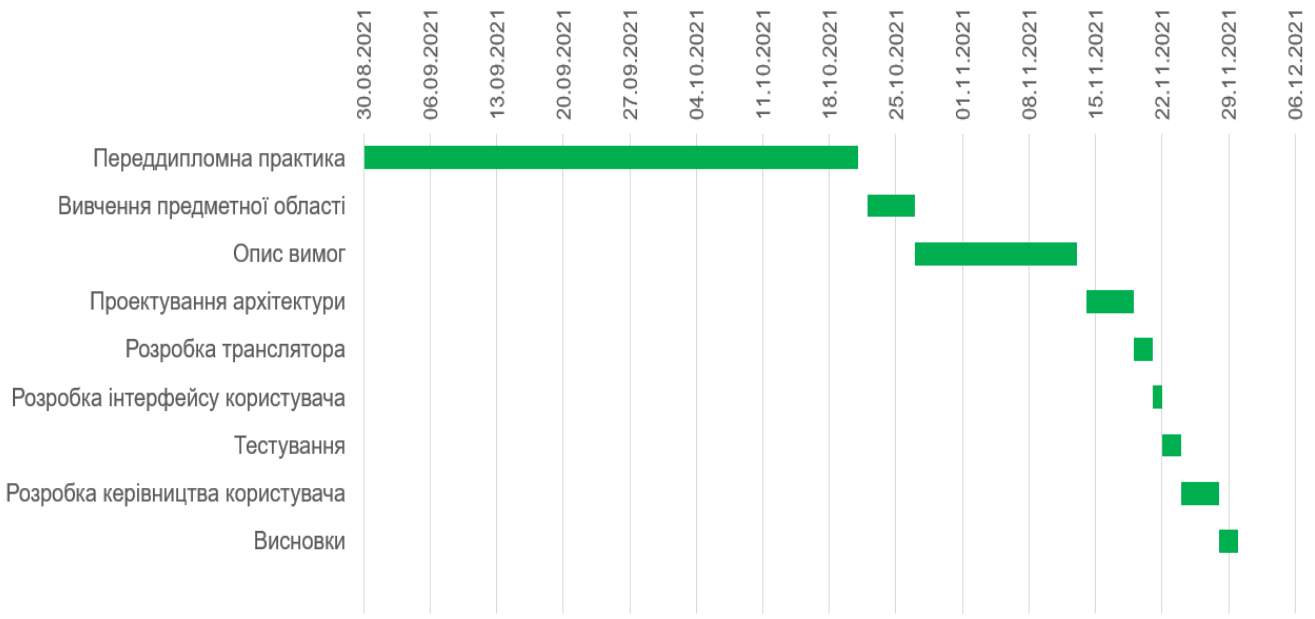

Рисунок 4.1 – Діаграма Ганта

Проаналізувавши діаграму Ганта, можна побачити, що всі етапи розробки йдуть послідовно, а не паралельно, тому ці етапи і лежать у критичному шляху проекту.

4.3 Ідентифікація та аналіз ризиків

Ризик – це проблема яка ще не з’явилася, а проблема – це ризик який матеріалізувався. Ризик – це завжди наслідки та імовірність [7]. Використовуючи класифікацію Архіпєнкова С., наслідки та імовірність діляться на три категорії відповідно. Наслідки мають одну з категорій: катастрофічні, критичні та помірні. Імовірність має такі категорії як, мало імовірно, імовірно та з великою імовірністю. Далі потрібно ідентифікувати ризики, використавши для цього підхід «досвід експертів». Згідно з цього підходу можна ідентифікувати п’ять ризиків:

1) Реалізація невідповідної функціональності.

2) Розробка неправильного інтерфейсу користувача.

3) Недоліки календарного планування

4) Недостатні знання спеціаліста

5) Непотрібна оптимізація та відточування деталей

Далі проведений аналіз цих ризиків по виявленню причин, наслідків та збитків, результат приведений у таблиці 4.2

Таблиця 4.2 – Аналіз ризиків

Ризик	Причина	Наслідок	Збиток
Реалізація невідповідної функціональності	Вимоги до системи визначені не точно	Велика кількість переробок	Затримки у строках здачі проекту
Розробка неправильного інтерфейсу користувача	Вимоги зручності використання визначені не точно	Підвисить рівень складності використання систему кінцевим користувачем	Затримки у строках здачі проекту
Недостатні знання спеціаліста	Спеціаліст має мало досвіду у цій галузі	Велика кількість помилок	Затримки у строках здачі проекту

Продовження таблиці 4.2

Недоліки календарного планування	Неправильні очікування тривалості робіт	Зниження працездатності системи	Затримки у строках здачі проекту
Непотрібна оптимізація та відточування деталей	Бажання спеціаліста зробити ідеальну систему	Зниження продуктивності системи	Затримки у строках здачі проекту

Далі проведений якісний аналіз ідентифікованих ризиків, для них необхідно було виявити імовірність, категорію наслідків, ранг наслідку. Ранг підраховується таким чином, спочатку оцінена імовірність: «мало імовірно» - 1, «імовірно» - 2 та з «великою імовірністю» - 3, потім оцінена категорія наслідків: «катастрофічні» - 3, «критичні» - 2 та «помірні» - 1, добуток двох значень формує ранг. Результати якісного аналізу приведені у таблиці 4.3

Таблиця 4.3 – Якісний аналіз ризиків

Ризик	Імовірність	Категорія наслідку	Ранг
Реалізація невідповідної функціональності	Мало імовірно	Критичні	2
Розробка неправильного інтерфейсу користувача	Мало імовірно	Помірні	1
Недостатні знання спеціаліста	Імовірно	Критичні	4

Продовження таблиці 4.3

Недоліки календарного планування	Імовірно	Катастрофічні	6
Непотрібна оптимізація та відточування деталей	Мало імовірно	Критичні	2

З таблиці 4.3 отриманий найважніший ризик, це – «Недолік календарного планування». Тому щоб попередити його, необхідно дуже детально проводити аналіз етапів та їх тривалості, залишати додатковий час на важливі етапи.

5 ПРОЕКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

5.1 Детальне проектування системи

Для детального проектування системи, спочатку необхідно розробити граматику. З її допомогою, можна очікувати як буде розгорнутий вхідний запит. Відображення абстрактного синтаксичного дерева, для майбутньої граматики, показано на рис. 5.1.

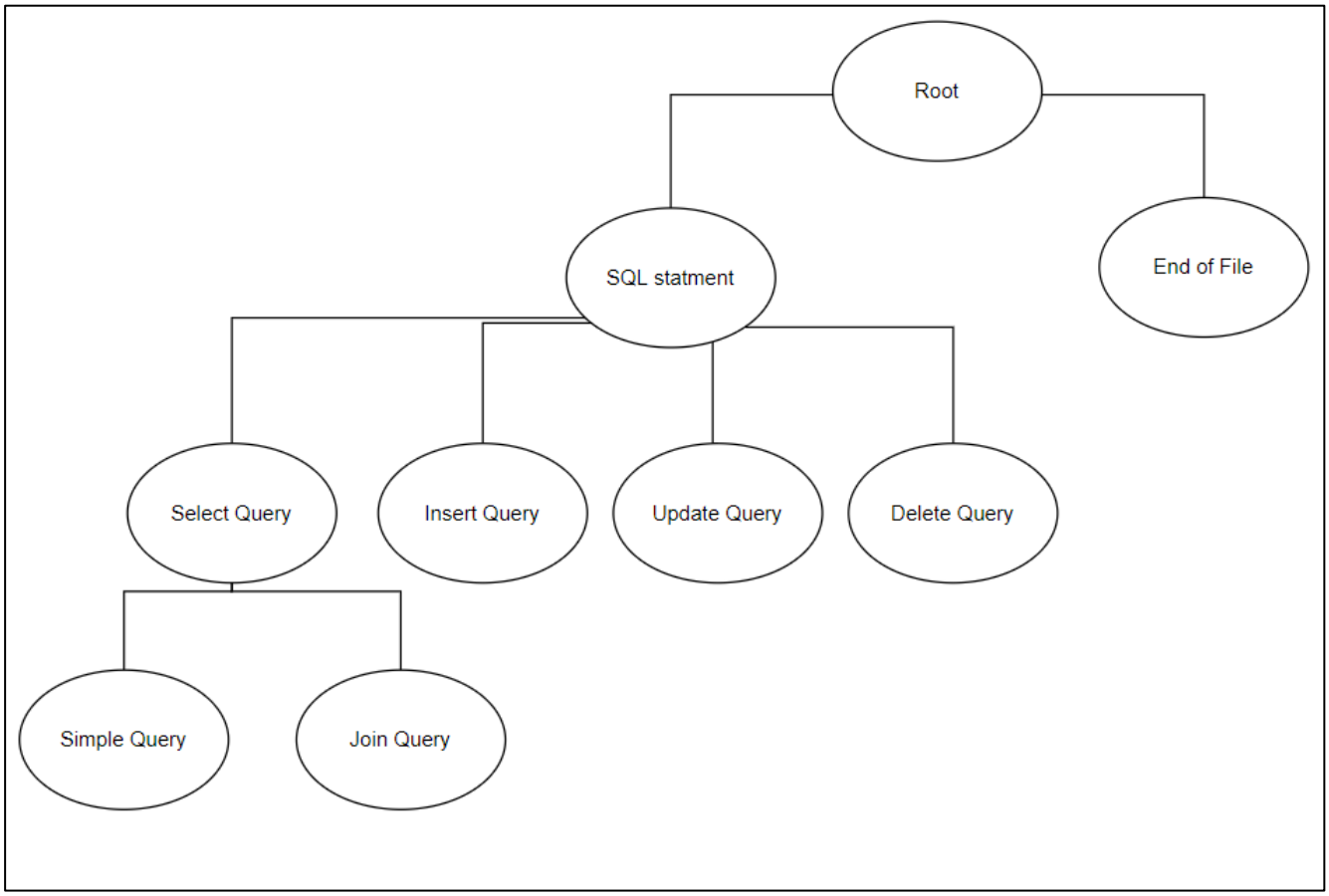

Рисунок 5.1 – Абстрактне синтаксичне дерево.

Спочатку зображений головний вузол – Root, він завжди ділиться на запит SQL та символ завершення файлу. Потім вхідний запит, аналізується на одну з DML команд, а саме – INSERT, SELECT, DELETE, UPDATE. Якщо команда відповідає синтаксису запиту SELECT, то вона ділиться на звичайну вибірку, або на вибірку з об’єднанням таблиць.

Виконавши декомпозицію предметної області, опираючись на процес отримання кінцевого результату, який був описаний у пункті 2.1, а також на абстрактне синтаксичне дерево, можна виділити такі модулі:

Графічний інтерфейс користувача (GUI) – цей модуль необхідний для того щоб, користувач мав можливість взаємодіяти з програмним продуктом, а також для того щоб, користувач мав можливість бачити результат роботи продукту.

Носій (Device) – цей модуль необхідний для того щоб, завантажувати у систему SQL запити, а не формувати їх самостійно у редакторі, а також він використовується для збереження вже обробленого запиту, результат якого створений мовою ArangoDB.

Транслятор (Translator) – цей модуль необхідний для організації робіт програмного продукту, він відповідає за всі етапи перетворення вхідного запиту на SQL до результату на ArangoDB, а саме створення простих одиниць (токенів), перетворення токенів у синтаксичне дерево та обхід цього дерева для створення результату.

Лексичний аналізатор (Lexer) – цей модуль необхідний для того щоб, з вхідного потоку символів створити прості одиниці (токени), які далі необхідні для трансляції

Синтаксичний аналізатор (Parser) – цей модуль необхідний для того щоб, з отриманих токенів, створити синтаксичне дерево викликів вузлів граматики.

Обробник (Listener) – цей модуль необхідний для того щоб, з отриманого синтаксичного дерева визивати, конкретні вузли які відповідають за різні типи запитів.

Обробник запитів INSERT (InsertQueryProcessor) – цей модуль відповідає за обробку SQL запитів типу INSERT, в аналоги мови ArangoDB.

Обробник запитів UPDATE (UpdateQueryProcessor) – цей модуль відповідає за обробку SQL запитів типу UPDATE, в аналоги мови ArangoDB.

Обробник запитів DELETE (DeleteQueryProcessor) – цей модуль відповідає за обробку SQL запитів типу UPDATE, в аналоги мови ArangoDB.

Обробник запитів на отримання даних з таблиць (SelectQueryProcessor) – цей модуль відповідає за обробку SQL запитів типу SELECT, в аналоги мови ArangoDB.

Допоміжний модуль для обробки запитів SELECT з конструкцією JOIN (JoinUtils)

5.2 Діаграми системи та їх опис

Після того як виявлені основні модулі необхідно спроектувати як вони будуть між собою взаємодіяти, які матимуть функції та атрибути. Тому спочатку приведена діаграма класів та опис усіх сутностей за допомогою статичного представлення моделі, тобто класів (рис 5.2). На діаграмі класів зображені такі класи як – GUI, Parser, Device, Lexer, Translator, Listener, InsertQueryProcessor, UpdateQueryProcessor, DeleteQueryProcessor, SelectQueryProcessor, JoinUtils. Пари, GUI – Translator, GUI – Device, поєднані між собою направленою асоціацією та мають відношення один до одного. Зв’язок Translator – Device, також має зв’язок направленої асоціації для того щоб, завантажувати та зберігати вхідні та вихідні запити. Пари, Translator – Lexer, Translator – Parser, Translator – Listener , поєднані між собою агрегацією, тобто вони входять в структуру транслятора та він викликає їх у необхідному йому порядку. Далі зв’язки, Listener – InsertQueryProcessor, Listener – UpdateQueryProcessor, Listener – DeleteQueryProcessor, Listener – SelectQueryProcessor, це все композиції. Тобто вони є розширенням класу Listener, так як додають власну логіку для обробки вхідного запиту, написаного на мові SQL. Залишається зв’язок направленої асоціації між SelectQueryProcessor та JoinUtils. Цей зв’язок необхідний для того щоб, запити з вибірки даних мали можливість, об’єднання з іншими таблицями. Ці два класи мають відношення один до багатьох, тому що основний клас SelectQueryProcessor, повинен мати зв’язок з JoinUtils, так як від початку трансляції, не визначено, чи потрібен даний клас.

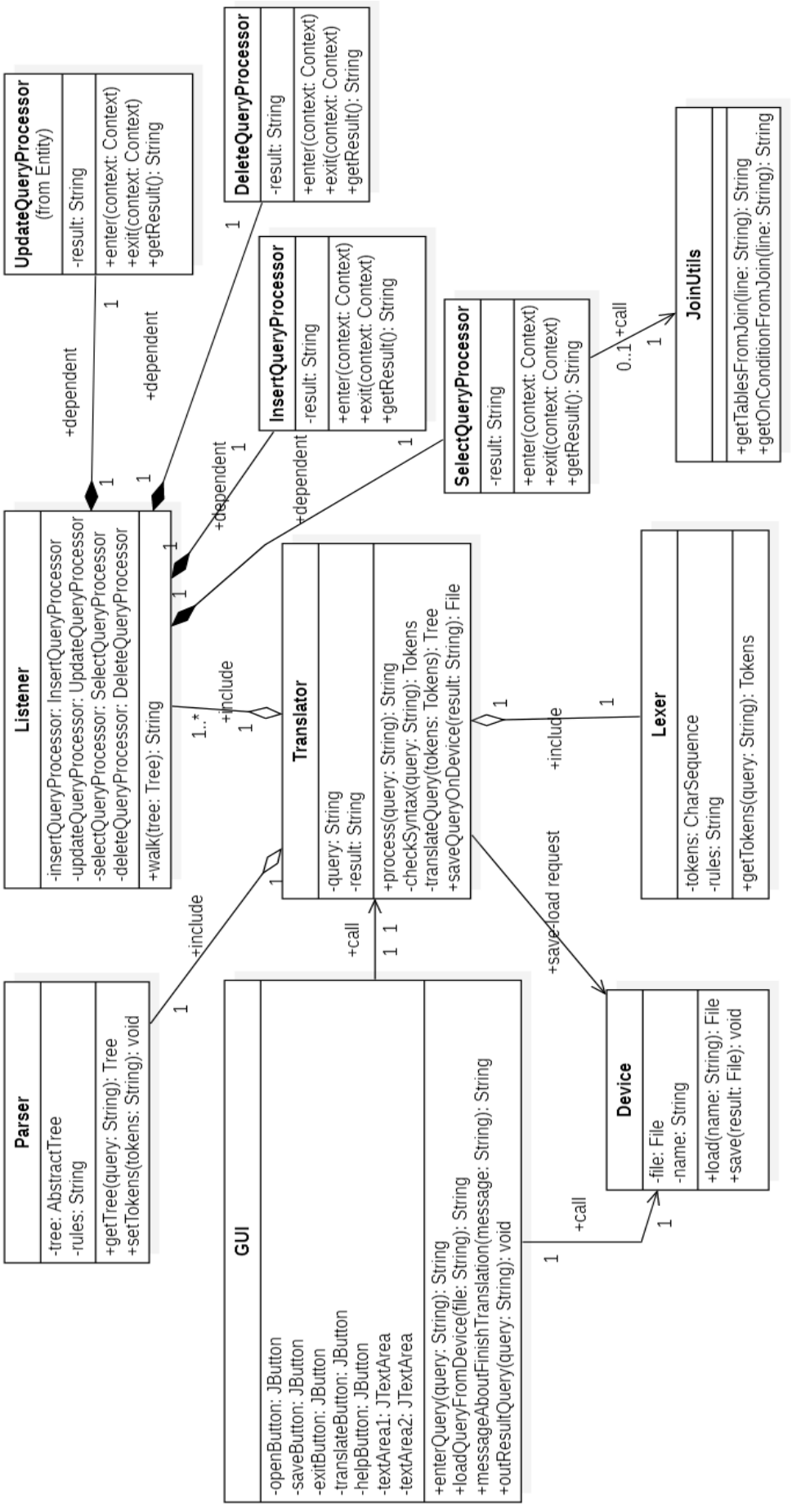

Рисунок 5.2 – Діаграма класів

Далі наведений детальний опис атрибутів та методів класів.

Атрибути та методи сутності Device:

- file : File – цей атрибут зберігає файл, який був використаний для завантаження запиту.
- name : String – атрибут зберігає ім’я файлу.
- load (name:String): File – завантажує запит по вказаному шляху, та повертає файл у форматі «.sql».
- save (result:File) : void – зберегти запит у файлі на носій

Атрибути та методи сутності GUI:

- openButton : JButton – це атрибут інтерфейсу користувача, а саме кнопка, відкрити файл у системі.
- saveButton : JButton – це атрибут інтерфейсу користувача, а саме кнопка, зберегти файл на девайс.
- exitButton : JButton – це атрибут інтерфейсу користувача, а саме кнопка, вийти з системи.
- translateButton : JButton – це атрибут інтерфейсу користувача, а саме кнопка, перекласти запит з однієї мови на іншу.
- helpButton : JButton – це атрибут інтерфейсу користувача, а саме кнопка, яка викликає інформацію по користуванню програмою.
- textArea1 : JTextArea – це атрибут інтерфейсу користувача, текстове поле до якого вноситься запит SQL.
- textArea2 : JTextArea– це атрибут інтерфейсу користувача, текстове поле до якого вноситься результат трансляції.
- enterQuery(query:String): String– метод який дозволяє ввести запит SQL до системи.
- loadQueryFromDevice(file:String) : String – визвати діалогове вікно завантаження файлу, та завантажити запит з файлу, шлях до якого вказаний у параметрі
- messageAboutFinishTranslation(message:String) : String – показати повідомлення про завершення трансляції

- outResultQuery(query:String): void – показати результат трансляції у текстове поле textArea2.

Атрибути та методи сутності Translator:

- query: String – цей атрибут зберігає запит на мові SQL.
- result: String – атрибут зберігає результат обробки запиту, тобто запит на ArangoDB.
- process(query:String): String – метод який займається обробкою перетворення запиту з SQL до ArangoDB. Викликає інші методи класу а саме: checkSyntax (query:String): Tokens, translateQuery (tokens:Tokens): Tree
- checkSyntax(query:String): Tokens – метод який отримує на вхід запит на SQL і повертає набір токенів.
- translateQuery(tokens:Tokens): Tree – метод який отримує на вхід набір токенів, а повертає синтаксичне дерево, побудоване на правилах граматики.
- saveQueryOnDevice(result:String): File – метод який зберігає результат запиту у файл.

Атрибути та методи сутності Lexer:

- rules: String – атрибут зберігає правила, за якими розбиває текст на прості лексичні одиниці, токени.
- tokens: CharSequence – атрибут який зберігає токени, тобто слова з алфавіту у конкретному запиті.
- getTokens(query:String): Tokens – отримати токени з запиту

Атрибути та методи сутності Parser:

- rules: String – атрибут зберігає правила, за якими формує з токенів, абстрактне дерево.
- tree: AbstractTree – атрибут зберігає абстрактне дерево яке було сформоване з синтаксичних правил та запиту
- getTree(query:String): Tree – отримати синтаксичне дерево с запиту SQL
- setTokens(tokens:String): void – набір токенів які необхідно зберегти у класі

Атрибути та методи сутності Listener:

- insertQueryProcessor: InsertQueryProcessor – атрибут зберігає об’єкт класу який обробляє синтаксичне дерево для роботи с запитами «INSERT».

- selectQueryProcessor: SelectQueryProcessor – атрибут зберігає об’єкт класу який обробляє синтаксичне дерево для роботи с запитами «SELECT».

- updateQueryProcessor: UpdateQueryProcessor– атрибут зберігає об’єкт класу який обробляє синтаксичне дерево для роботи с запитами «UPDATE».

- deleteQueryProcessor: DeleteQueryProcessor– атрибут зберігає об’єкт класу який обробляє синтаксичне дерево для роботи с запитами «DELETE».

- walk(tree:Tree): String – метод обробляє синтаксичне дерево і повертає результат обробки, а саме команду у форматі ArangoDB.

Атрибути та методи сутності InsertQueryProcessor

- result: String – атрибут зберігає результат обробки запиту, тобто запит на ArangoDB.

- enter(context:Context) – вхід у логічний вузол, а саме той, який прописаний у граматиці.

- exit(context:Context) – вихід з логічного вузла, а саме той, який прописаний у граматиці.

- getResult(): String – отримати результати запиту

Атрибути та методи сутності SelectQueryProcessor

- result: String – атрибут зберігає результат обробки запиту, тобто запит на ArangoDB.

- enter(context:Context) – вхід у логічний вузол, а саме той, який прописаний у граматиці.

- exit(context:Context) – вихід з логічного вузла, а саме той, який прописаний у граматиці.

- getResult(): String– отримати результати запиту

Атрибути та методи сутності UpdateQueryProcessor

- result: String – атрибут зберігає результат обробки запиту, тобто запит на ArangoDB.

- enter(context:Context) – вхід у логічний вузол, а саме той, який прописаний у граматиці.

- exit(context:Context) – вихід з логічного вузла, а саме той, який прописаний у граматиці.

- getResult(): String– отримати результати запиту

Атрибути та методи сутності DeleteQueryProcessor

- result: String – атрибут зберігає результат обробки запиту, тобто запит на ArangoDB.

- enter(context:Context) – вхід у логічний вузол, а саме той, який прописаний у граматиці.

- exit(context:Context) – вихід з логічного вузла, а саме той, який прописаний у граматиці.

- getResult(): String– отримати результати запиту

Атрибути та методи сутності JoinUtils

- getTablesFromJoin(line:String): String – метод який виводить з вхідної строки яка містить конструкцію JOIN, назву таблиць.

- getOnConditionFromJoin(line:String): String – метод який виводить з вхідної строки яка містить конструкцію JOIN, умову з’єднання ON.

Далі представлена діаграма діяльності усього бізнес процесу.

Діаграми діяльності — це графічне зображення робочих процесів і дій з підтримкою вибору, ітерації та паралельності. В UML діаграми діяльності призначені для моделювання, як обчислювальних, так і організаційних процесів, а також потоків даних, що перетинаються з відповідними видами діяльності. Хоча діаграми діяльності в першу чергу показують загальний напрям процесу, вони також можуть включати елементи, які показують напрям даних між видами діяльності через одне або кілька сховищ даних.

Він зображений на рисунку 5.3, все починається з того, що потрібно створити запит, його можна завантажити або написати в ручну, далі цей запит перевіряється на синтаксис SQL, якщо не має помилок то починається трансляція запиту. Далі результат запиту має бути виведений на екран, або збережений на носій.

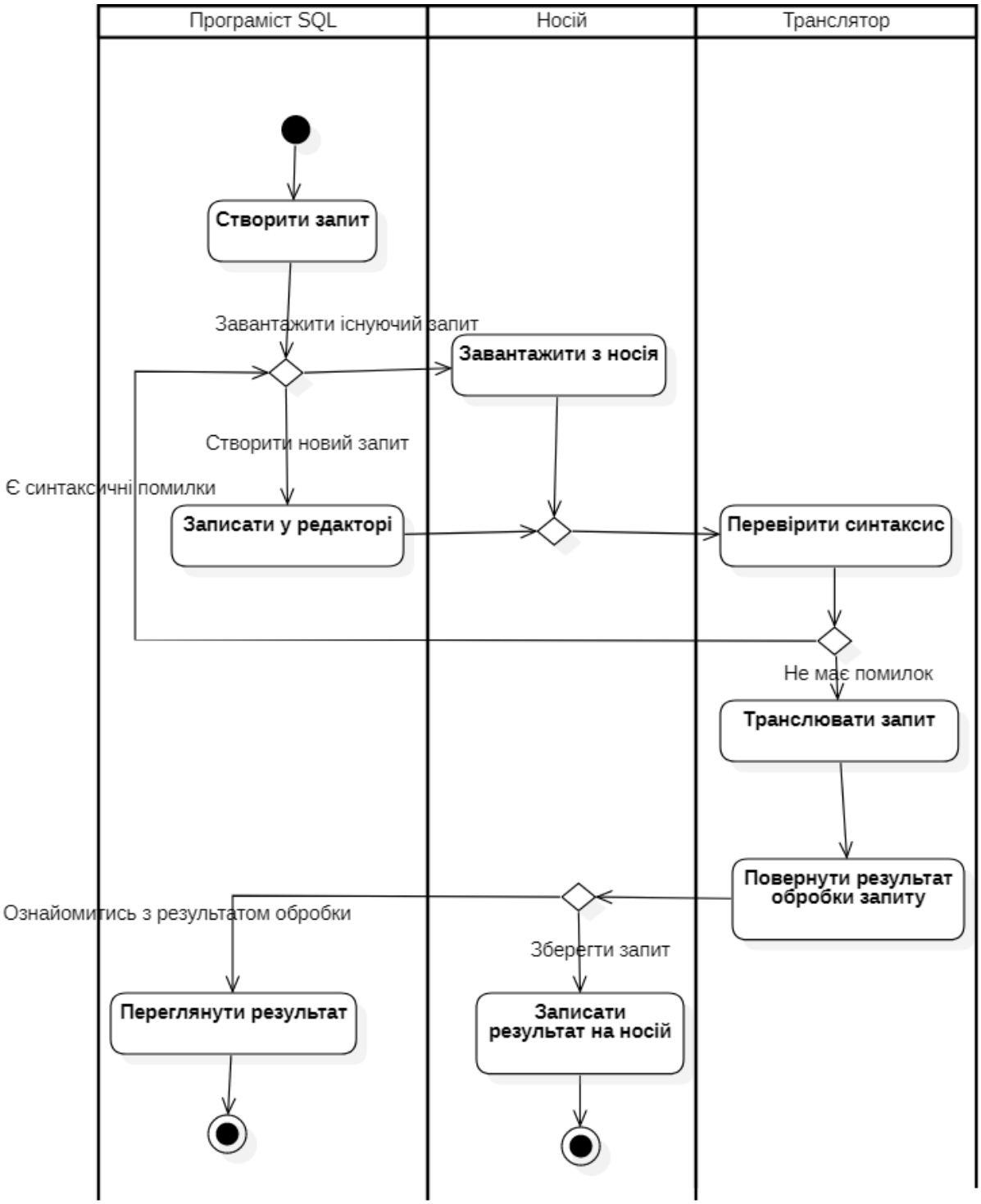

Рисунок 5.3 – Діаграма діяльності

Для того щоб відобразити послідовність взаємодії компонентів була створена таблиця повідомлень процесу на високому рівні (табл. 5.1), завдяки цій таблиці також можна створити діаграму комунікації. Діаграма комунікації моделює взаємодії між об'єктами або частинами в термінах впорядкованих повідомлень.

Комунікаційні діаграми представляють комбінацію інформації, взятої з діаграм класів, послідовності і варіантів використання, описуючи відразу і статичну структуру і динамічну поведінку системи.

Таблиця 5.1 – Повідомлення процесу на високому рівні

Номер повідомлення	Об'єкт – відправника повідомлення	Об’єкт – отримувач повідомлення	Назва
1	Програміст SQL	GUI	Вибір запиту з носія
2	GUI	Носій	Завантаження запиту
3	Носій	GUI	Return
4	Програміст SQL	GUI	Редагування запиту та відправка на трансляцію
5	GUI	Транслятор	Трансляція запиту
6	Транслятор	GUI	Результат трансляції
7	Програміст SQL	GUI	Зберегти запит
8	GUI	Носій	Збереження запиту

Були графічно відображені взаємодії компонентів та користувача (рис 5.4). Стрілка вказує напрям комунікації, а номер - порядок послідовності відправки повідомлень.

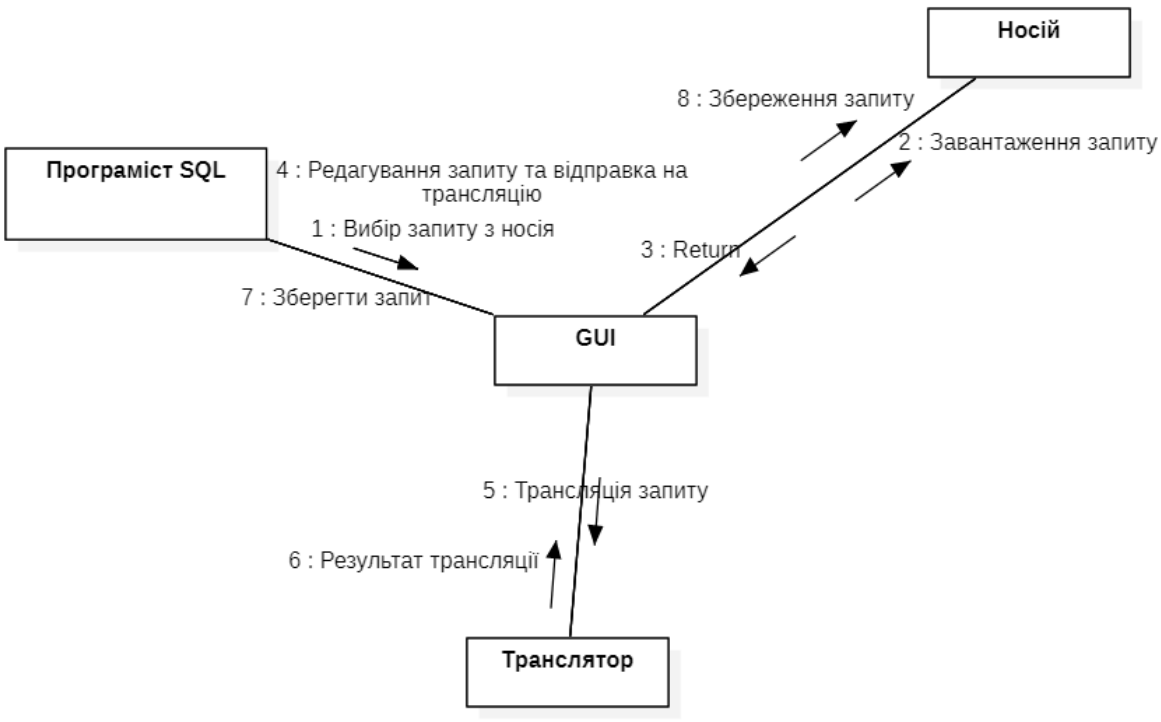

Рисунок 5.4 – Діаграма комунікації

5.3 Структура сховища даних

Для зберігання даних у програмі, була обрана файлова система. Необхідно мати два файли, формату «.g4». для повноцінної роботи системи, а саме для зберігання терміналів та для синтаксичних правил (граматики). Термінал – це об'єкт, безпосередньо присутній в словах мови, відповідного граматиці, і має конкретне, незмінне значення (узагальнення поняття «букви»). У формальних мовах, що використовуються на комп'ютері, в якості терміналів, зазвичай беруть всі або частину стандартних символів ASCII - латинські букви, цифри і спеціальні символи. У файлі з терміналами структура така, спочатку файлу прописується властивість, що він зберігає термінали (являється лексичним аналізатором) потім кожний термінал визначається множиною символів алфавіту (рис 5.5).

```
lexer grammar MySqlLexer;

channels { MYSQLCOMMENT, ERRORCHANNEL }

AND:                                    'AND';
AS:                                     'AS';
ASC:                                    'ASC';
BEFORE:                                 'BEFORE';
BETWEEN:                                'BETWEEN';
```

Рисунок 5.5 – Структура файлу з терміналами

Структура файлу який зберігає граматику, а саме синтаксичні правила, складніша. Все починається з головного вузла, далі йде низхідний розбір правил у яких відображені альтернативи «|», ключові слова (з лексичного аналізатору), та нетермінали які далі розкладаються на такі компоненти, до поки не буде нетермінальних слів (рис 5.6)

```
parser grammar MySqlParser;

options { tokenVocab=MySqlLexer; }

dmlStatement
    :  insertStatement | updateStatement
    | deleteStatement;

deleteStatement
    : singleDeleteStatement
    ;

insertStatement
    : INSERT
      IGNORE? INTO? tableName
      (
        ('(' columns=uidList ')')? insertStatementValue
      )
    ;

updateStatement
    : singleUpdateStatement
    ;

// details

insertStatementValue
    :  insertFormat=(VALUES | VALUE)
      '(' expressionsWithDefaults ')'
    ;
```

Рисунок 5.6 – Структура файлу з синтаксичними правилами

5.4 Інтерфейс користувача

На рисунку 5.7 зображений прототип графічного інтерфейсу. Рядок з меню необхідний для того, щоб – завантажити файл, зберегти файл, скопіювати, вирізати, вставити, отримати допомогу. Далі йде текстове поле у який користувач записує запит. Потім натискає на кнопку «Трансформувати», і у наступному текстовому полі отримує сформований результат, написаний на мові ArangoDB

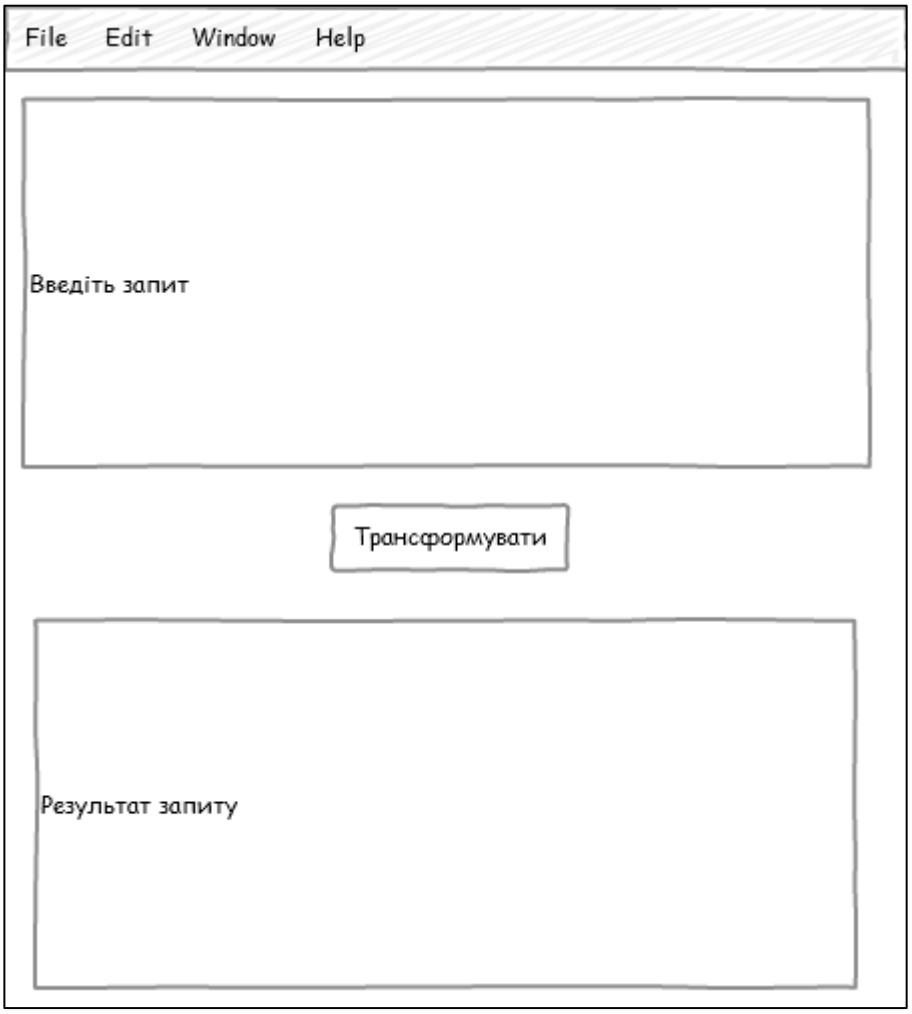

Рисунок 5.7 – Прототип графічного інтерфейсу

6 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

6.1 Інструменти розробки

Для розробки програмного продукту використовується фреймворк – ANTLR. ANTLR - це потужний генератор парсерів, який можна використовувати для читання, обробки, виконання або перекладу структурованого тексту, або двійкових файлів. Він широко використовується в академічних та промислових установах, для створення різноманітних мов, інструментів і фреймворків. Oracle використовує ANTLR в рамках IDE SQL Developer і його інструментів міграції. Можна створювати всілякі корисні інструменти, такі як читачі конфігураційних файлів, парсери JSON. З опису формальної мови, що називається граматикою, ANTLR генерує парсер для цієї мови, яка може автоматично створювати дерева розбору, які є структурами даних, що представляють, як граматика відповідає входу. ANTLR також автоматично генерує обробники дерева, які можна використовувати для відвідування вузлів цих дерев для виконання коду, що стосується програми. ANTLR широко використовується, тому що його легко зрозуміти, потужний, гнучкий, генерує доступний для читання висновок, поставляється з повним джерелом під ліцензією BSD і активно підтримується [8].

Наступний інструмент розробки, бібліотека – java.lang. Забезпечує класи, які є фундаментальними для проектування мови програмування Java. Найбільш важливими класами є Object, який є коренем ієрархії класів, і Class, екземпляри якого представляють класи під час виконання.

Часто необхідно представити значення примітивного типу, як якщо б він був об'єктом. Класи обгортки Boolean, Character, Integer, Long, Float, та Double служать для цієї мети. Об'єкт типу Double, наприклад, містить поле, тип якого є double, що представляє це значення таким чином, що посилання на нього може зберігатися в змінної типу посилання. Ці класи також надають ряд методів для перетворення серед примітивних значень, а також підтримують такі стандартні методи, як equals і hashCode. Клас Void – це нестандартний клас, який містить посилання на об'єкт класу, що представляє тип void. Клас Math забезпечує зазвичай використовувані

математичні функції, такі як синус, косинус і квадратний корінь. Класи String, StringBuffer і StringBuilder подібним чином забезпечують зазвичай використовувані операції на рядках символів. Клас Throwable охоплює об'єкти, які можуть бути згенеровані оператором throw. Підкласи Throwable являють собою помилки та винятки.

Ще одна бібліотека яка використовується – java.util. Вона включає велику кількість допоміжних класів, широко використовуваних у розробці. Ці класи використовуються для роботи з набором об'єктів, взаємодії з системними функціями низького рівня, для роботи з математичними функціями, генерації випадкових чисел і маніпуляцій з датою і часом. Однією із головних переваг використання бібліотеки, це інтерфейс Collection. Він є основою всієї ієрархії класів-колекцій і визначає базову функціональність будь-якої колекції – набір методів які дозволяють додавати, видаляти, вибирати елементи колекції. Класи які реалізують інтерфейс Collection, можуть містити дублікати і порожні (null) значення. AbstractCollection, будучи абстрактним класом забезпечує основу для створення конкретних класів колекцій і містить реалізацію деяких методів визначених у інтерфейсі Collection.

Бібліотека – Swing, необхідна для того щоб, створити інтерфейс користувача. Swing — інструментарій для створення графічного інтерфейсу користувача (GUI) мовою програмування Java. Swing розробляли для забезпечення більшого набору програмних компонентів, для створення графічного інтерфейсу користувача, ніж у інструментарію AWT. Компоненти Swing підтримують специфічні look-and-feel модулі, що динамічно підключаються. Завдяки ним, можлива емуляція графічного інтерфейсу платформи (тобто до компоненту можна динамічно підключити інші, специфічні для даної операційної системи вигляд і поведінку). Основним недоліком таких компонентів є – відносно повільна робота, хоча останнім часом це не вдалося виправити, через зростання потужності персональних комп'ютерів. Позитивна сторона — універсальність інтерфейсу створених програм на всіх платформах [9].

Наступний інструмент розробки це текстовий формат який використовується в мові ArangoDB це – JSON. JSON – це текстовий формат обміну даними між комп'ютерами. Він базується на тексті та може бути прочитаним людиною. Формат дає змогу описувати об'єкти та інші структури даних. Цей формат використовується переважно для передачі структурованої інформації через мережу. Розробив і популяризував формат Дуглас Крокфорд. JSON знайшов своє головне призначення в написанні веб-програм, а саме при використанні технології AJAX. Ця технологія, що використовується в AJAX, виступає як заміна XML, під час асинхронної передачі структурованої інформації між клієнтом та сервером. При цьому перевагою JSON перед XML є те, що він дозволяє, складні структури в атрибутах, займає менше місця і прямо інтерпретується за допомогою JavaScript в об'єкти. JSON будується на двох структурах:

- Набір пар назва/значення. У різних мовах це реалізовано як об'єкт, запис, структура, словник, хеш-таблиця, список із ключем або асоціативним масивом.
- Впорядкований список значень. У багатьох мовах це реалізовано як масив, вектор, список або послідовність.

Це універсальні структури даних. Теоретично всі сучасні мови програмування підтримують їх у тій чи іншій формі. Оскільки JSON використовується для обміну даними між різними мовами програмування, то є сенс будувати його на цих структурах [10].

Також у ArangoDB використовується змішаний формат, а саме бінарна форма JSON. BSON — це бінарна форма представлення простих структур даних і асоціативних масивів (які називають об'єктами або документами). Назва «BSON» заснована на визначенні JSON і неофіційно означає «Binary JSON» (бінарний JSON). Порівняно з JSON, BSON є ефективним з точки зору, як розміру збережених даних, так і швидкості сканування. Великі елементи в документі BSON мають префікс з довжиною документа для полегшення перебору [11].

6.2 Реалізація інтерфейсу користувача

Спроектована модель інтерфейсу користувача, була реалізована за допомогою бібліотеки Swing. Є головне вікно яке зображено на рисунку 6.1, вікно складається з – рядку з меню, який необхідний для того, щоб – завантажити файл, зберегти файл, скопіювати, вирізати, вставити, отримати допомогу; текстового поля у яке користувач записує запит на мові SQL, для прикладу записаний запит INSERT; кнопки «Трансформувати», яка перетворює запит з SQL на ArangoDB; текстового поля, яке отримує сформований результат, написаний на мові ArangoDB

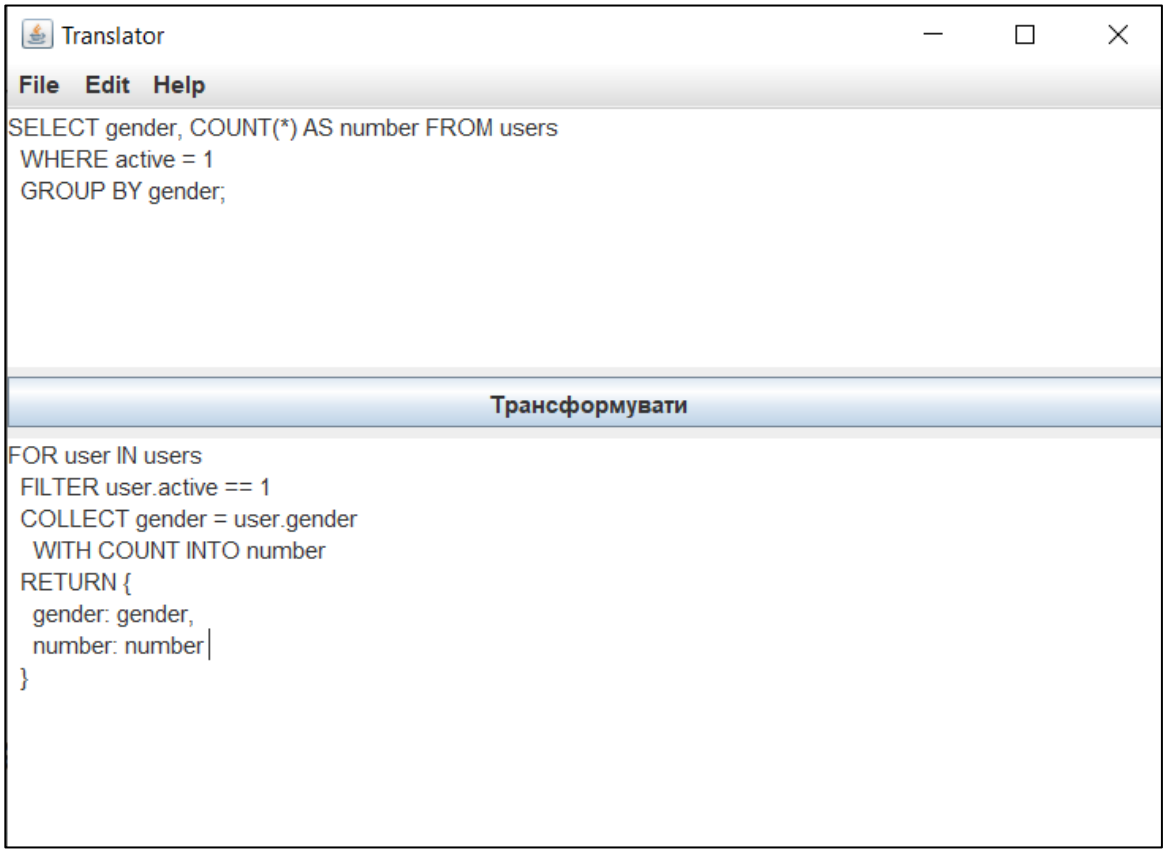

Рисунок 6.1 – Графічний інтерфейс

Далі більш детально описане контекстне меню «File», яке складається з пунктів «Відкрити файл», «Зберегти файл», «Вихід» (рис 6.2).

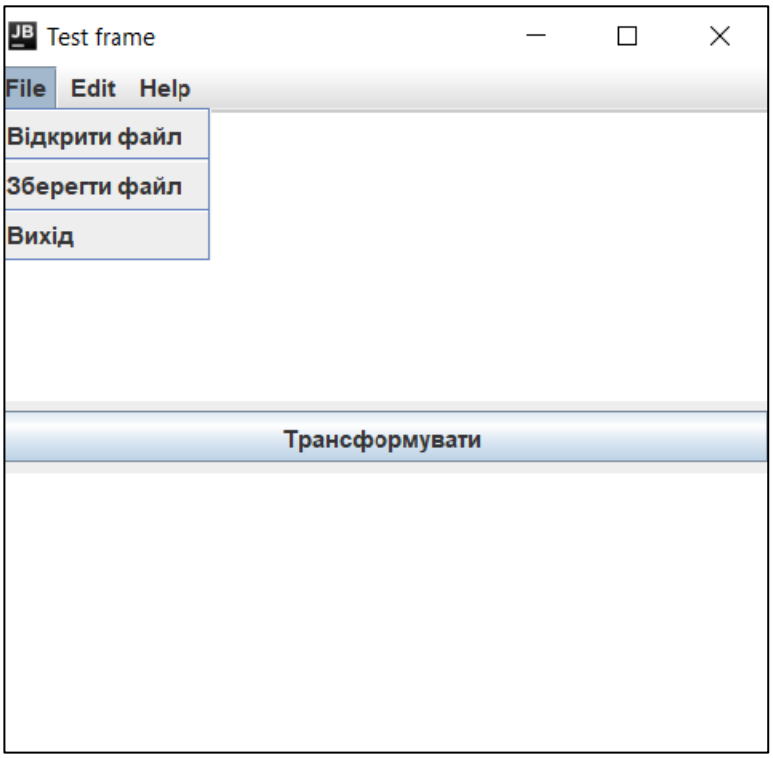

Рисунок 6.2 – Контекстне меню «File»

Для того щоб відкрити файл, необхідно обрати пункт «Відкрити файл», з’явиться вікно (рис 6.3), у якому необхідно обрати шлях до файлу типу «.sql»

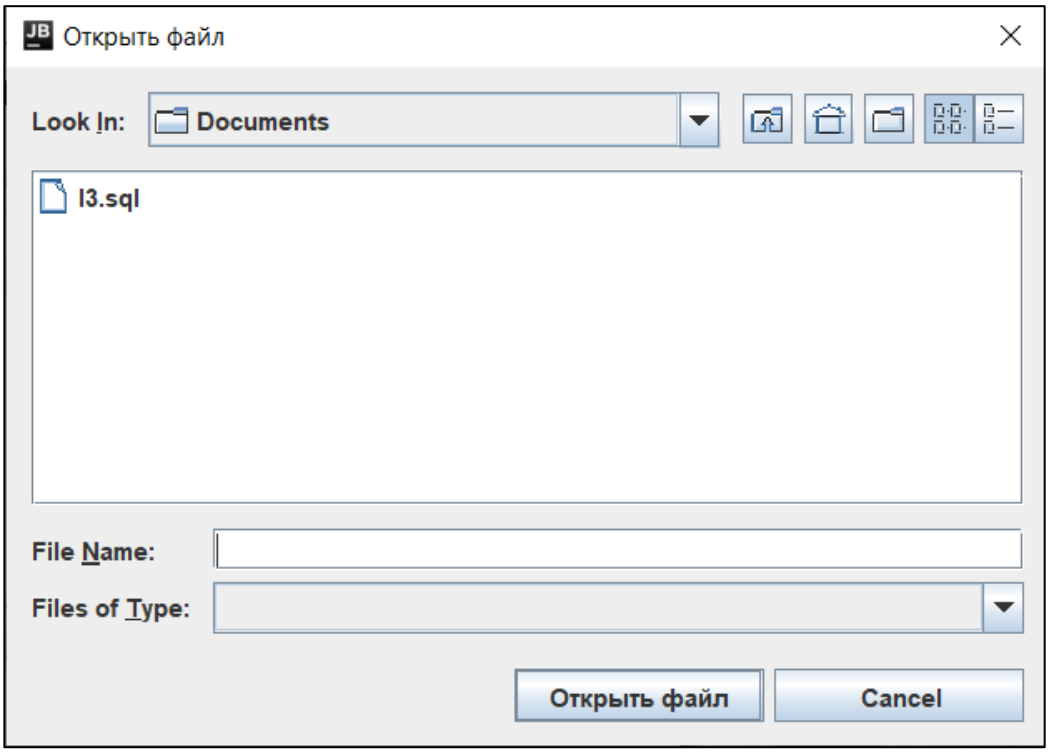

Рисунок 6.3 – Вікно пункту меню «Відкрити файл»

Після того як був отриманий результат трансляції, його можна зберегти обрав пункт меню «Зберегти файл», потім вказати необхідний шлях та назву файлу, як вказано на рисунку 6.4. Пункт «Вихід» відповідає за закриття програми.

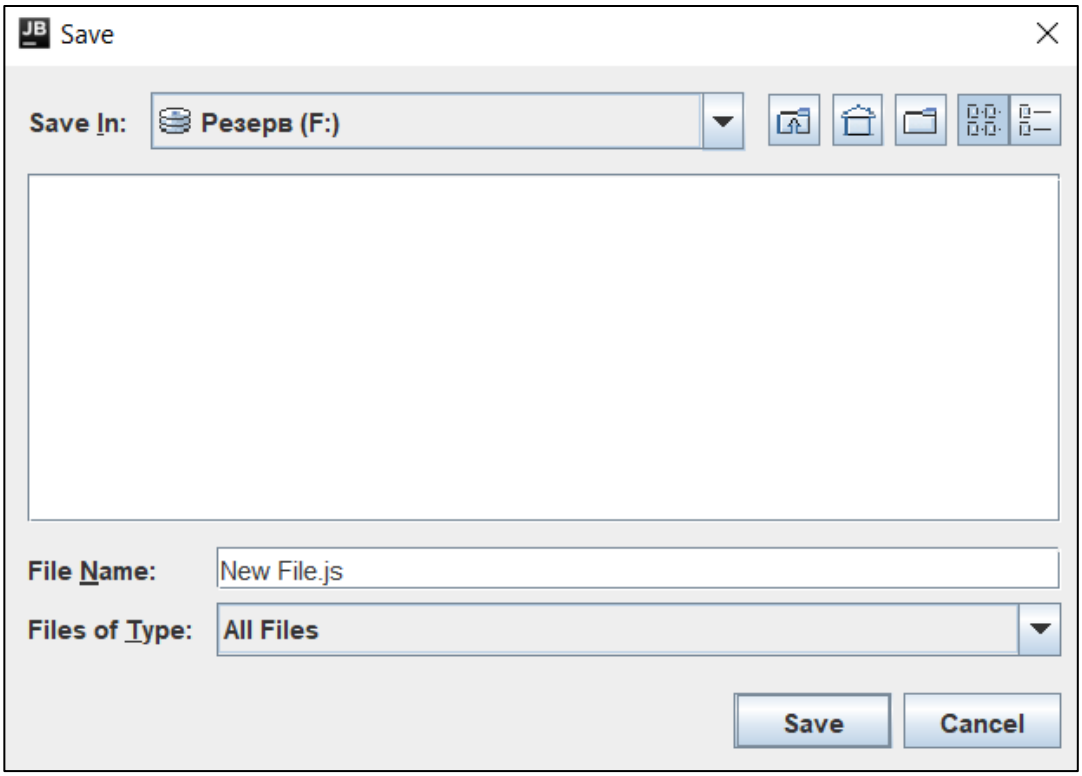

Рисунок 6.4 – Вікно пункту меню «Зберегти файл»

6.3 Побудова функціональної діаграми

Метод функціональних діаграм або діаграм причинно-наслідкових зв'язків допомагає систематично вибирати високорезультативні тести. Крім цього, метод функціональних діаграм дає корисний побічний ефект, так як дозволяє виявляти неповноту і неоднозначність вихідних специфікацій. Методика використання функціональних діаграм наступна: специфікація розбивається на "робочі" ділянки, так як для великих специфікацій функціональні діаграми стають занадто громіздкими. У специфікації визначаються причини і наслідки, причина – це окрема вхідна умова або клас еквівалентних вхідних умов, наслідок – це вихідна умова (результат виконання програми).

Для тестування побудуємо функціональну діаграму. Необхідно трансформувати запити INSERT, DELETE, UPDATE, написані на мові SQL, на функції ArangoDB, а саме INSERT {} INTO, UPDATE {} WITH {} IN, FOR IN FILTER REMOVE IN відповідно. Якщо не збігається синтаксис команди SQL, то видається повідомлення «Неправильний синтаксис». Виділяємо причини та наслідки.

Причини:

- 1 – Вводиться запит INSERT
- 2 – Вводиться запит DELETE
- 3 – Вводиться запит UPDATE
- 4 – Коректний синтаксис команди

Наслідки:

- 21 – Виводиться результат INSERT {} INTO
- 22 – Виводиться результат UPDATE {} WITH {} IN
- 23 – Виводиться результат FOR IN FILTER REMOVE IN
- 24 – Видається повідомлення «Неправильний синтаксис»

Побудуємо функціональну діаграму, яка пов’язує причини та наслідки (рис. 6.5).

Для наочності графа використовуються проміжні вершини 11, 12, 13, які об’єднують між собою вершини «1» та «4», «2» та «4», «3» та «4» відповідно.

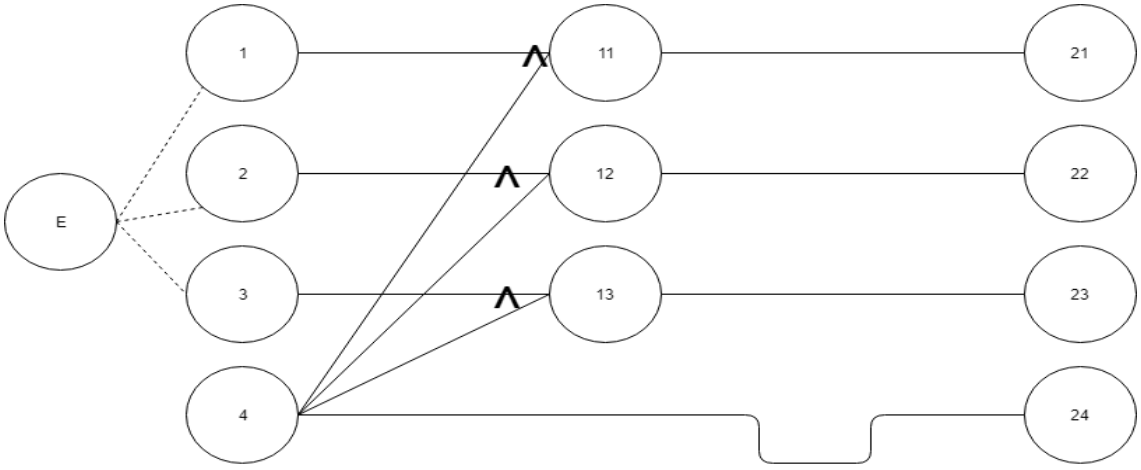

Рисунок 6.5 – Функціональна діаграма

Отриманий граф забезпечується обмеженнями: причини 1, 2 і 3 не можуть заповняться одночасно, так як це концептуально різні команди

6.4 Побудова та скорочення таблиці рішень

Будуємо таблицю рішень по графу рис 6.5. Для цього фіксуємо в стані «1» по черзі всі наслідки.

«21» – в стані «1», якщо проміжна вершина «11» буде в стані «1». Вона, в свою чергу, буде знаходиться в стані «1» тільки в тому випадку, коли причина «1» в стані «1» та причина «4» в стані «1» одночасно, тому що вони пов’язані логічним І.

«22» – в стані «1», якщо проміжна вершина «12» буде в стані «1». Вона, в свою чергу, буде знаходиться в стані «1» тільки в тому випадку, коли причина «2» в стані «1» та причина «4» в стані «1» одночасно, тому що вони пов’язані логічним І.

«23» – в стані «1», якщо проміжна вершина «13» буде в стані «1». Вона, в свою чергу, буде знаходиться в стані «1» тільки в тому випадку, коли причина «3» в стані «1» та причина «4» в стані «1» одночасно, тому що вони пов’язані логічним І.

«24» - буде в стані «1» тільки в тому випадку, якщо причина «4», з якою вона пов’язана, буде знаходиться в стані «0»

Результати аналізу записані у таблиці 6.1.

Таблиця 6.1 – Таблиця рішень

Тип	Вершина	1	2	3	4
Причина	1	1	-	-	-
	2	-	1	-	-
	3	-	-	1	-
	4	1	1	1	0
Наслідок	21	1	-	-	-
	22	-	1	-	-
	23	-	-	1	-
	24	-	-	-	1

6.5 Результати тестування

Для демонстрації контрольного прикладу, був обраний приклад трансляції запиту «DELETE FROM USERS WHERE NAME = ‘OLEG’ AND AGE = 20». Спочатку запит розбивається на токени, далі використовуючи правила граматики, побудоване синтаксичне дерево. Спочатку йдуть нетермінали, до того моменту, як парсер не визначить що, це запит на видалення даних. Далі парсер виділяє ключові слова, а саме «DELETE» «FROM» «WHERE», та назву таблиці й вираз умови. Наступний крок перевіряє, з чого складається вираз умови, в даному випадку він комплексний, тому він розкладається на складові (прості умови) та ключове слово «AND». Прості умови розкладаються на назву характеристики (атрибуту) та його значення між якими знаходиться термінал «=». Самі значення можуть мати тип звичайного тексту, або тип числа. Далі, у коді програми реалізований обробник, цього синтаксичного дерева, який перекладає запит у аналогічний запит на мові ArangoDB. Після обробки отриманий результат FOR user IN users FILTER user.name == ‘OLEG’ AND user.age ==20 REMOVE user IN users

Далі наведені приклади, для останніх випадків із таблиці рішень, але вже без детального розбору.

INSERT { name: " OLEG"} INTO users – правильно написаний запит INSERT (наприклад INSERT INTO USERS (NAME) VALUES (‘OLEG’)) (причина 1 = «1») та (причина 4 = «1»).

UPDATE {name: "OLEG" }WITH {name: "SASHA"} IN users правильно написаний запит UPDATE (наприклад UPDATE USERS SET NAME = ‘SASHA’ WHERE NAME=’OLEG’) (причина 3 = «1») та (причина 4 = «1»).

Повідомлення «Неправильний синтаксис» - написаний неправильно запит (наприклад DELETE USERS WHERE NAME = ‘OLEG’) (причина 4 = «0»).

6.6 Керівництво користувача

Для того щоб користуватися програмним забезпеченням, спочатку необхідно зайти у саму програму. Далі буде форма яка зображена на рисунку 6.6.

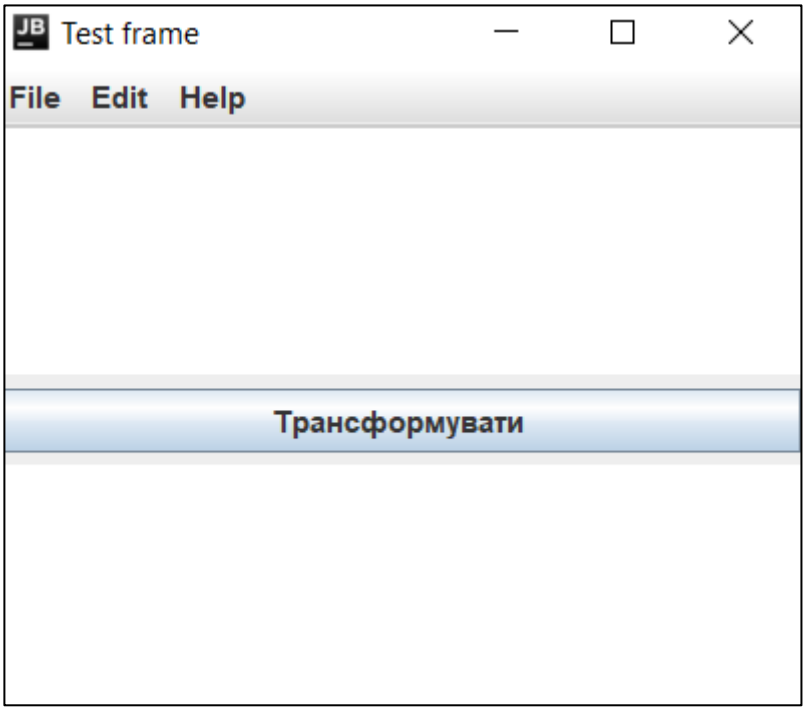

Рисунок 6.6 – Початкова форма програми

Далі можливі два варіанти подій. Перший завантажити у програму запит, який буде відображений у верхньому текстовому полі, робиться це так. Спочатку необхідно у контекстному меню обрати пункт «File», «Відкрити файл», та обрати необхідний файл типу «.sql». Другий варіант, одразу записати запит написаний мовою SQL, а саме «INSERT», «DELETE», «UPDATE», «SELECT». У текстове поле. У будь-якому варіанті – результат один, запит записаний та готовий до трансляції (рис 6.7). Коли запит буде повністю написаний, необхідно натиснути кнопку «Трансформувати» , через деякий час програма покаже повідомлення про завершення трансляції, після цього у другому текстовому полі буде відображений аналог запиту, написаний за допомогою ArangoDB (рис 6.8).

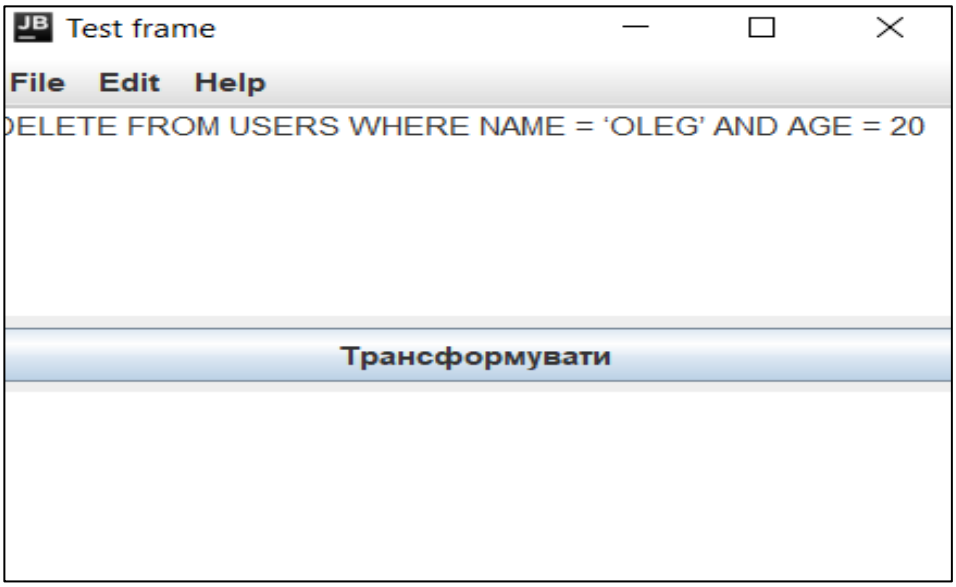

Рисунок 6.7 – Заповнена форма програми

Якщо запит був сформований неправильно, то програма покаже повідомлення про помилку, та про те, щоб користувач заново сформував запит.

Зауваження – запит сформований на SQL, не регістрочуттєвий, тому не важливо яким регістром написаний запит, програма сама перетворить запит до верхнього регістру.

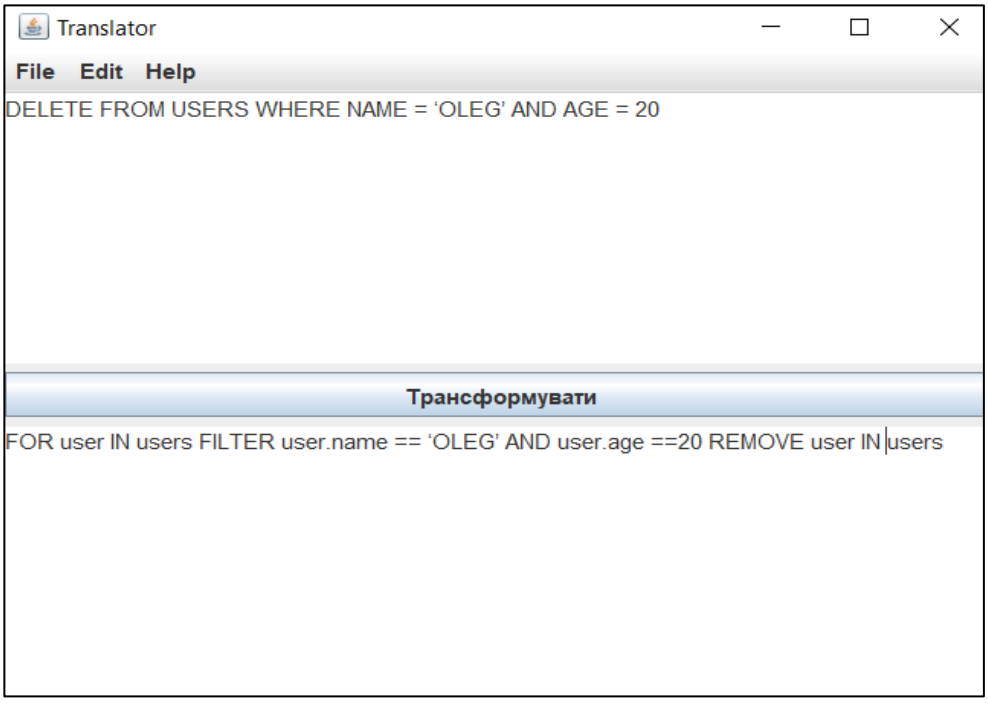

Рисунок 6.8 – Результат трансляції

7 ВИПРОБУВАННЯ СИСТЕМИ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

Після того, як програмний продукт був розроблений, необхідно перевірити досягнення поставленої мети, а саме зменшення часу необхідного для переносу існуючої реляційної бази даних створеної на мові SQL, до мультимодельної мови, ArangoDB, та розширення функціональності існуючих трансляторів шляхом додавання команд маніпуляції з даним, тобто SQL команди «INSERT», «DELETE», «UPDATE».

Була поставлена гіпотеза, перевірити чи зменшився час необхідний для переносу існуючої реляційної бази, для перевірки буде використана метрика – середній час виконання п’ятнадцяти запитів у розробленому продукті та в одному з аналогів.

Для цього необхідно провести дослідження, яке полягає у перевірці п’ятнадцяти, однакових запитів «SELECT», спочатку у програмах аналогах, а потім у розробленому трансляторі. Щоб дослідження мало достовірні данні, необхідно виконувати його на одному й тому самому апаратному обладнані. Також самі запити «SELECT» матимуть три різні складності:

- Звичайний запит до однієї таблиці, з секціями умовами та сортуванням.
- Запит з об’єднанням таблиць (JOIN).
- Запит з об’єднанням таблиць (JOIN), і секціями умовами та сортуванням.

Результати порівняння експерименту зображені у таблиці 7.1.

Таблиця 7.1 – Час виконання запитів у двох системах

Номер запиту	Виконання запиту у розробленому трансляторі, мс.	Виконання запиту у в аналогічній системі «Sql_to_ArangoDb», мс.	Зменшення часу у відсотках, %
1	350	386	9,3
2	327	367	10
3	306	341	10,2

Продовження таблиці 7.1

Номер запиту	Виконання запиту у розробленому трансляторі, мс.	Виконання запиту у в аналогічній системі «Sql_to_ArangoDb», мс.	Зменшення часу у відсотках, %
4	380	454	16,2
5	347	394	11,9
6	689	743	7,2
7	703	764	7,9
8	654	699	6,4
9	693	734	5,5
10	715	777	7
11	450	501	10,1
12	479	537	10,8
13	464	526	11,7
14	485	545	11,0
15	446	567	21,3

Знаючи зменшення часу у відсотках для кожного запиту, можна розрахувати, за допомогою середнього арифметичного, середнє зменшення часу. Після розрахунку отримано значення – 10,6%. Відобразимо отримані дані з таблиці 7.1 на графіку (рис. 7.1). З графіку можна побачити три різні області, у першій знаходяться прості запити, вони виконуються швидше за інших. В другій області запити з об’єднанням і у третій запити з об’єднанням та секціями умовами та сортуванням. У експерименті не порівнювались інші DML запити, а саме INSERT, DELETE, UPDATE, так як вони мають завжди мають однакову структуру, тільки різну кількість параметрів. Вони транслюються однаково у розробляємому трансляторі та в програмних аналогах.

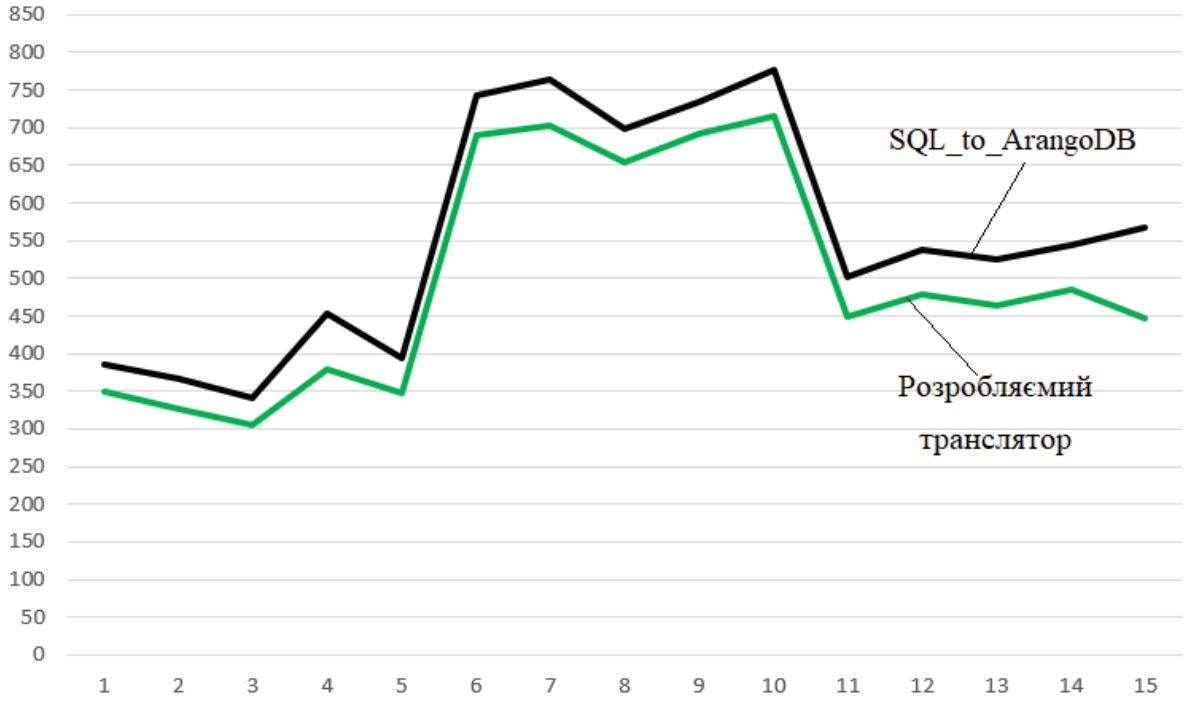

Рисунок 7.1 – Графік порівняння двох систем

Так як запити у розробленому трансляторі транслюються швидше, то і на графіку вони знаходиться нижче за аналог «Sql_to_ArangoDb». Отримані дані дозволяють підтвердити гіпотезу про зменшення необхідного для переносу часу.

ВИСНОВКИ

На основі мови програмування Java та фреймворку ANTLR, був створений мовний транслятор, який виконує трансляцію запитів написаних на мові SQL, у запити мови ArangoDB. Метою роботи було розширення функціональності аналогів та зменшення часу необхідного на трансляцію, і розроблений програмний продукт реалізував це. Розробка дозволяє отримувати аналоги реляційних запитів INSERT, DELETE, UPDATE, SELECT, в відповідні запити мови ArangoDB, Транслятор вирішує проблему тривалості переносу бази даних з одного типу до іншого, тим що зменшує час, у середньому на 10.6%, який для цього необхідний програмісту SQL, якщо він використовує створену систему. Транслятор може бути розширений наступною функціональністю, наприклад він міг би підключатись до консолі ArangoDB, і за бажанням користувача, одразу виконувати команди у сховищі. Або ж підключатись до серверу SQL, і коли користувач виконує команду SELECT у трансляторі, робити запит до сховища та отримані результати у таблиці перетворювати до виду результатів з бази ArangoDB.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. NoSQL: нова методологія розробки нереляційних баз даних. : Пер. С англ. – М.: ООО «И.Д. Вильяме», 2013. – 192 с.

2. Стеченко Д.М., Чмир О.С. Методологія наукових досліджень: Підручник: - К.: Знання, 2005. – 190 с.

3. Грофф, Джеймс Р., Вайнберг, Пол Н., Оппелъ, Эндрю Дж. Г89 SQL: повний посібник, 3-е изд. : Пер. с англ. – М.: ООО «И.Д. Вильяме», 2015. – 960 с.

4. Эрик Редмонд, Джим. Р. Уилсон. Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL = ArangoDB in Action. — ДМК Пресс, 2013. — 384 с.

5. Легалов А.И., Швец Д.А., Легалов И.А. Формальные языки и трансляторы М. 2007. – 213 с.

6. А. Кофман, Г. Дебазей, Сетевые методы планирования / Издательство «Прогресс М. 1968. – 183 с.

7. Архипенков, С. Лекции по управлению программными проектами / С. Архипенков. – М. Самиздат, 2009. – 128 с.

8. Парр, Теренс. The Definitive ANTLR 4 Reference/ Парр, Теренс. – The Pragmatic Programmers, 2012. – с. 322

9. Шилдт, Герберт. Java 8. Полное руководство. 9-е издание / Герберт Шилдт. – М. :ООО «І. Д Вільямс», 2015. – 1376 с.

10. Крокфорд Д. The application/json Media Type for JavaScript Object Notation (JSON)/ Крокфорд Д. — Internet Engineering Task Force, 2006. – 27 с.

11. Шашанк Тиварі. Professional NoSQL. — Packt Publishing, 2011. — 384 с.

12. Молдованова О.В. Языки программирования и методы трансляции: Учебное пособие. – Новосибирск/СибГУТИ, 2012. – 134с.

13. Kungurtsev, O. B., Zinovatnaya S. L & Potochniak, I. V. “Method of Searching Term Interpretations for Domain Dictionaries, Used for Developing Software”. Applied Aspects of Information Technology. Publ. Science і Technical. Odessa: Ukraine. 2019; Vol. 2 No.1: 11–19. DOI: https://doi.org/10.15276/aait.02.2019.1

14. Maksymov O. S., Malakhov E. V. & Mezhuyev V. I.“Model and method for representing complex dynamic information objects based on LMS-trees in NoSQL databases”. Herald of Advanced Information Technology. Publ. Nauka i Tekhnika. Odessa: Ukraine. 2021; Vol. 4 No. 3: 211–224. DOI: https://doi.org/10.15276/hait.03.2021.1

ДОДАТОК А. ЛІСТИНГ ПРОГРАМИ

Файл граматики MySqlParser.g4

```
parser grammar MySqlParser;

options { tokenVocab=MySqlLexer; }


// Top Level Description

root
    : sqlStatements? MINUSMINUS? EOF
    ;

sqlStatements
    : (sqlStatement MINUSMINUS? SEMI? | emptyStatement)*
    (sqlStatement (MINUSMINUS? SEMI)? | emptyStatement)
    ;

sqlStatement
    : dmlStatement
;
emptyStatement
    : SEMI
    ;


dmlStatement
    :  insertStatement | updateStatement
    | deleteStatement;


// Data Definition Language

//    Primary DML Statements



deleteStatement
    : singleDeleteStatement
    ;



insertStatement
    : INSERT
      IGNORE? INTO? tableName
      (
        ('(' columns=uidList ')')? insertStatementValue
      )
    ;

updateStatement
    : singleUpdateStatement
    ;

// details

insertStatementValue
```

```
    :  insertFormat=(VALUES | VALUE)
      '(' expressionsWithDefaults ')'
    ;

updatedElement
    : fullColumnName '=' (expression | DEFAULT)
    ;


//    Detailed DML Statements

singleDeleteStatement
    : DELETE
    FROM tableName
      (WHERE expression)?
    ;


singleUpdateStatement
    : UPDATE tableName
      SET updatedElement (',' updatedElement)*
      (WHERE expression)?
    ;

// details


//    DB Objects

fullId
    : uid (DOT_ID | '.' uid)?
    ;

tableName
    : fullId
    ;

fullColumnName
    : uid
    ;

uid
    : simpleId
    | REVERSE_QUOTE_ID
    | CHARSET_REVERSE_QOUTE_STRING
    ;

simpleId
    : ID
    | keywordsCanBeId
    ;


//    Literals

decimalLiteral
    : DECIMAL_LITERAL | ZERO_DECIMAL | ONE_DECIMAL | TWO_DECIMAL
    ;

stringLiteral
    : (
        STRING_CHARSET_NAME? STRING_LITERAL
        | START_NATIONAL_STRING_LITERAL
```

```
          ) STRING_LITERAL+
        | (
            STRING_CHARSET_NAME? STRING_LITERAL
            | START_NATIONAL_STRING_LITERAL
          )
        ;

    booleanLiteral
        : TRUE | FALSE;

    constant
        : stringLiteral | decimalLiteral
        | '-' decimalLiteral
        | booleanLiteral
        | REAL_LITERAL | BIT_STRING
        | NOT? nullLiteral=(NULL_LITERAL | NULL_SPEC_LITERAL)
        ;

    //    Common Lists

    uidList
        : uid (',' uid)*
        ;

    expressionsWithDefaults
        : expressionOrDefault (',' expressionOrDefault)*
        ;


    //    Common Expressons

    expressionOrDefault
        : expression | DEFAULT
        ;



    //    Expressions, predicates

    // Simplified approach for expression
    expression
        :
         expression                  logicalOperator                  expression
#logicalExpression
        | predicate                                          #predicateExpression
        ;

    predicate
        :             left=predicate      comparisonOperator      right=predicate
#binaryComparasionPredicate
        |             (LOCAL_ID            VAR_ASSIGN)?            expressionAtom
#expressionAtomPredicate
        ;


    // Add in ASTVisitor nullNotnull in constant
    expressionAtom
        : constant                                          #constantExpressionAtom
        |                                                          fullColumnName
#fullColumnNameExpressionAtom
        ;

    comparisonOperator
```

```
    : '=' | '>' | '<' | '<' '=' | '>' '='
    | '<' '>' | '!' '=' | '<' '=' '>'
    ;

logicalOperator
    : AND | '&' '&' | XOR | OR | '|' '|'
    ;

//    Simple id sets
//     (that keyword, which can be id)

keywordsCanBeId
    :  NAME
    ;
```

Файл з токенами MySqlLexer.g4

```
lexer grammar MySqlLexer;

channels { MYSQLCOMMENT, ERRORCHANNEL }

// SKIP

SPACE:                                [ \t\r\n]+    -> channel(HIDDEN);
SPEC_MYSQL_COMMENT:                   '/*!' .+? '*/' -> channel(MYSQLCOMMENT);
COMMENT_INPUT:                        '/*' .*? '*/' -> channel(HIDDEN);
LINE_COMMENT:                         (
                                        ('-- ' | '#') ~[\r\n]* ('\r'? '\n' |
EOF)
                                        | '--' ('\r'? '\n' | EOF)
                                      ) -> channel(HIDDEN);


// Keywords
// Common Keywords

ADD:                                  'ADD';
ALL:                                  'ALL';
ALTER:                                'ALTER';
ALWAYS:                               'ALWAYS';
ANALYZE:                              'ANALYZE';
AND:                                  'AND';
AS:                                   'AS';
ASC:                                  'ASC';
BEFORE:                               'BEFORE';
BETWEEN:                              'BETWEEN';
BOTH:                                 'BOTH';
BY:                                   'BY';
CALL:                                 'CALL';
CASCADE:                              'CASCADE';
CASE:                                 'CASE';
CAST:                                 'CAST';
CHANGE:                               'CHANGE';
CHARACTER:                            'CHARACTER';
CHECK:                                'CHECK';
COLLATE:                              'COLLATE';
COLUMN:                               'COLUMN';
CONDITION:                            'CONDITION';
CONSTRAINT:                           'CONSTRAINT';
CONTINUE:                             'CONTINUE';
CONVERT:                              'CONVERT';
CREATE:                               'CREATE';
```

```
CROSS:                          'CROSS';
CURRENT_USER:                   'CURRENT_USER';
CURSOR:                         'CURSOR';
DATABASE:                       'DATABASE';
DATABASES:                      'DATABASES';
DECLARE:                        'DECLARE';
DEFAULT:                        'DEFAULT';
DELAYED:                        'DELAYED';
DELETE:                         'DELETE';
DESC:                           'DESC';
DESCRIBE:                       'DESCRIBE';
DETERMINISTIC:                  'DETERMINISTIC';
DISTINCT:                       'DISTINCT';
DISTINCTROW:                    'DISTINCTROW';
DROP:                           'DROP';
EACH:                           'EACH';
ELSE:                           'ELSE';
ELSEIF:                         'ELSEIF';
ENCLOSED:                       'ENCLOSED';
ESCAPED:                        'ESCAPED';
EXISTS:                         'EXISTS';
EXIT:                           'EXIT';
EXPLAIN:                        'EXPLAIN';
FALSE:                          'FALSE';
FETCH:                          'FETCH';
FOR:                            'FOR';
FORCE:                          'FORCE';
FOREIGN:                        'FOREIGN';
FROM:                           'FROM';
FULLTEXT:                       'FULLTEXT';
GENERATED:                      'GENERATED';
GRANT:                          'GRANT';
GROUP:                          'GROUP';
HAVING:                         'HAVING';
HIGH_PRIORITY:                  'HIGH_PRIORITY';
IF:                             'IF';
IGNORE:                         'IGNORE';
IN:                             'IN';
INDEX:                          'INDEX';
INFILE:                         'INFILE';
INNER:                          'INNER';
INOUT:                          'INOUT';
INSERT:                         'INSERT';
INTERVAL:                       'INTERVAL';
INTO:                           'INTO';
IS:                             'IS';
ITERATE:                        'ITERATE';
JOIN:                           'JOIN';
KEY:                            'KEY';
;
NULL_LITERAL:                   'NULL';
ON:                             'ON';
OR:                             'OR';
ORDER:                          'ORDER';
OUT:                            'OUT';
OUTER:                          'OUTER';
OUTFILE:                        'OUTFILE';
PARTITION:                      'PARTITION';
PRIMARY:                        'PRIMARY';
PROCEDURE:                      'PROCEDURE';
PURGE:                          'PURGE';
RANGE:                          'RANGE';
READ:                           'READ';
```

```
READS:                          'READS';
REFERENCES:                     'REFERENCES';
REGEXP:                         'REGEXP';
RELEASE:                        'RELEASE';
RENAME:                         'RENAME';
REPEAT:                         'REPEAT';
REPLACE:                        'REPLACE';
REQUIRE:                        'REQUIRE';
RESTRICT:                       'RESTRICT';
RETURN:                         'RETURN';
REVOKE:                         'REVOKE';
RIGHT:                          'RIGHT';
RLIKE:                          'RLIKE';
SCHEMA:                         'SCHEMA';
SCHEMAS:                        'SCHEMAS';
SELECT:                         'SELECT';
SET:                            'SET';
SEPARATOR:                      'SEPARATOR';
SHOW:                           'SHOW';
SPATIAL:                        'SPATIAL';
SQL:                            'SQL';
SQLEXCEPTION:                   'SQLEXCEPTION';
SQLSTATE:                       'SQLSTATE';
SQLWARNING:                     'SQLWARNING';
SQL_BIG_RESULT:                 'SQL_BIG_RESULT';
SQL_CALC_FOUND_ROWS:            'SQL_CALC_FOUND_ROWS';
SQL_SMALL_RESULT:               'SQL_SMALL_RESULT';
SSL:                            'SSL';
STARTING:                       'STARTING';
STRAIGHT_JOIN:                  'STRAIGHT_JOIN';
TABLE:                          'TABLE';
TERMINATED:                     'TERMINATED';
THEN:                           'THEN';
TO:                             'TO';
TRAILING:                       'TRAILING';
TRIGGER:                        'TRIGGER';
TRUE:                           'TRUE';
UNDO:                           'UNDO';
UNION:                          'UNION';
UNIQUE:                         'UNIQUE';
UNLOCK:                         'UNLOCK';
UNSIGNED:                       'UNSIGNED';
UPDATE:                         'UPDATE';
USAGE:                          'USAGE';
USE:                            'USE';
USING:                          'USING';
VALUES:                         'VALUES';
WHEN:                           'WHEN';
WHERE:                          'WHERE';
WHILE:                          'WHILE';
WITH:                           'WITH';
WRITE:                          'WRITE';
XOR:                            'XOR';
ZEROFILL:                       'ZEROFILL';


// DATA TYPE Keywords

TINYINT:                        'TINYINT';
SMALLINT:                       'SMALLINT';
MEDIUMINT:                      'MEDIUMINT';
INT:                            'INT';
INTEGER:                        'INTEGER';
```

```
BIGINT:                                 'BIGINT';
REAL:                                   'REAL';
DOUBLE:                                 'DOUBLE';
PRECISION:                              'PRECISION';
FLOAT:                                  'FLOAT';
DECIMAL:                                'DECIMAL';
DEC:                                    'DEC';
NUMERIC:                                'NUMERIC';
DATE:                                   'DATE';
TIME:                                   'TIME';
TIMESTAMP:                              'TIMESTAMP';
DATETIME:                               'DATETIME';
YEAR:                                   'YEAR';
CHAR:                                   'CHAR';
VARCHAR:                                'VARCHAR';
NVARCHAR:                               'NVARCHAR';
NATIONAL:                               'NATIONAL';
BINARY:                                 'BINARY';
VARBINARY:                              'VARBINARY';
TINYBLOB:                               'TINYBLOB';
BLOB:                                   'BLOB';
MEDIUMBLOB:                             'MEDIUMBLOB';
LONGBLOB:                               'LONGBLOB';
TINYTEXT:                               'TINYTEXT';
TEXT:                                   'TEXT';
MEDIUMTEXT:                             'MEDIUMTEXT';
LONGTEXT:                               'LONGTEXT';
ENUM:                                   'ENUM';
VARYING:                                'VARYING';
SERIAL:                                 'SERIAL';


// Interval type Keywords

YEAR_MONTH:                             'YEAR_MONTH';
DAY_HOUR:                               'DAY_HOUR';
DAY_MINUTE:                             'DAY_MINUTE';
DAY_SECOND:                             'DAY_SECOND';
HOUR_MINUTE:                            'HOUR_MINUTE';
HOUR_SECOND:                            'HOUR_SECOND';
MINUTE_SECOND:                          'MINUTE_SECOND';
SECOND_MICROSECOND:                     'SECOND_MICROSECOND';
MINUTE_MICROSECOND:                     'MINUTE_MICROSECOND';
HOUR_MICROSECOND:                       'HOUR_MICROSECOND';
DAY_MICROSECOND:                        'DAY_MICROSECOND';

// Keywords, but can be ID
// Common Keywords, but can be ID

ACCOUNT:                                'ACCOUNT';
ACTION:                                 'ACTION';
AFTER:                                  'AFTER';
AGGREGATE:                              'AGGREGATE';
ALGORITHM:                              'ALGORITHM';
ANY:                                    'ANY';
AT:                                     'AT';
AUTHORS:                                'AUTHORS';
AUTOCOMMIT:                             'AUTOCOMMIT';
AUTOEXTEND_SIZE:                        'AUTOEXTEND_SIZE';
AUTO_INCREMENT:                         'AUTO_INCREMENT';
AVG_ROW_LENGTH:                         'AVG_ROW_LENGTH';
BEGIN:                                  'BEGIN';
BINLOG:                                 'BINLOG';
```

```
BIT:                    'BIT';
BLOCK:                  'BLOCK';
BOOL:                   'BOOL';
BOOLEAN:                'BOOLEAN';
BTREE:                  'BTREE';
CACHE:                  'CACHE';
CASCADED:               'CASCADED';
CHAIN:                  'CHAIN';
CHANGED:                'CHANGED';
CHANNEL:                'CHANNEL';
CHECKSUM:               'CHECKSUM';
PAGE_CHECKSUM:          'PAGE_CHECKSUM';
CIPHER:                 'CIPHER';
CLIENT:                 'CLIENT';
CLOSE:                  'CLOSE';
COALESCE:               'COALESCE';
CODE:                   'CODE';
COLUMNS:                'COLUMNS';
COLUMN_FORMAT:          'COLUMN_FORMAT';
COMMENT:                'COMMENT';
COMMIT:                 'COMMIT';
COMPACT:                'COMPACT';
COMPLETION:             'COMPLETION';
COMPRESSED:             'COMPRESSED';
COMPRESSION:            'COMPRESSION';
CONCURRENT:             'CONCURRENT';
CONNECTION:             'CONNECTION';
CONSISTENT:             'CONSISTENT';
CONTAINS:               'CONTAINS';
CONTEXT:                'CONTEXT';
CONTRIBUTORS:           'CONTRIBUTORS';
COPY:                   'COPY';
CPU:                    'CPU';
DATA:                   'DATA';
DATAFILE:               'DATAFILE';
DEALLOCATE:             'DEALLOCATE';
DEFAULT_AUTH:           'DEFAULT_AUTH';
DEFINER:                'DEFINER';
DELAY_KEY_WRITE:        'DELAY_KEY_WRITE';
DES_KEY_FILE:           'DES_KEY_FILE';
DIRECTORY:              'DIRECTORY';
DISABLE:                'DISABLE';
DISCARD:                'DISCARD';
DISK:                   'DISK';
DO:                     'DO';
DUMPFILE:               'DUMPFILE';
DUPLICATE:              'DUPLICATE';
DYNAMIC:                'DYNAMIC';
ENABLE:                 'ENABLE';
ENCRYPTION:             'ENCRYPTION';
END:                    'END';
ENDS:                   'ENDS';
ENGINE:                 'ENGINE';
ENGINES:                'ENGINES';
ERROR:                  'ERROR';
ERRORS:                 'ERRORS';
ESCAPE:                 'ESCAPE';
EVEN:                   'EVEN';
EVENT:                  'EVENT';
EVENTS:                 'EVENTS';
EVERY:                  'EVERY';
EXCHANGE:               'EXCHANGE';
EXCLUSIVE:              'EXCLUSIVE';
```

```
EXPIRE:                         'EXPIRE';
EXPORT:                         'EXPORT';
EXTENDED:                       'EXTENDED';
EXTENT_SIZE:                    'EXTENT_SIZE';
FAST:                           'FAST';
FAULTS:                         'FAULTS';
FIELDS:                         'FIELDS';
FILE_BLOCK_SIZE:                'FILE_BLOCK_SIZE';
FILTER:                         'FILTER';
FIRST:                          'FIRST';
FIXED:                          'FIXED';
FLUSH:                          'FLUSH';
FOLLOWS:                        'FOLLOWS';
FOUND:                          'FOUND';
FULL:                           'FULL';
FUNCTION:                       'FUNCTION';
GENERAL:                        'GENERAL';
GLOBAL:                         'GLOBAL';
GRANTS:                         'GRANTS';
GROUP_REPLICATION:              'GROUP_REPLICATION';
HANDLER:                        'HANDLER';
HASH:                           'HASH';
HELP:                           'HELP';
HOST:                           'HOST';
HOSTS:                          'HOSTS';
IDENTIFIED:                     'IDENTIFIED';
IGNORE_SERVER_IDS:              'IGNORE_SERVER_IDS';
IMPORT:                         'IMPORT';
INDEXES:                        'INDEXES';
INITIAL_SIZE:                   'INITIAL_SIZE';
INPLACE:                        'INPLACE';
INSERT_METHOD:                  'INSERT_METHOD';
INSTALL:                        'INSTALL';
INSTANCE:                       'INSTANCE';
INVOKER:                        'INVOKER';
IO:                             'IO';
IO_THREAD:                      'IO_THREAD';
IPC:                            'IPC';
ISOLATION:                      'ISOLATION';
ISSUER:                         'ISSUER';
JSON:                           'JSON';
KEY_BLOCK_SIZE:                 'KEY_BLOCK_SIZE';
LANGUAGE:                       'LANGUAGE';
LAST:                           'LAST';
LEAVES:                         'LEAVES';
LESS:                           'LESS';
LEVEL:                          'LEVEL';
LIST:                           'LIST';
LOCAL:                          'LOCAL';
LOGFILE:                        'LOGFILE';
LOGS:                           'LOGS';
MASTER:                         'MASTER';
MASTER_AUTO_POSITION:           'MASTER_AUTO_POSITION';
MASTER_CONNECT_RETRY:           'MASTER_CONNECT_RETRY';
MASTER_DELAY:                   'MASTER_DELAY';
MASTER_HEARTBEAT_PERIOD:        'MASTER_HEARTBEAT_PERIOD';
MASTER_HOST:                    'MASTER_HOST';
MASTER_LOG_FILE:                'MASTER_LOG_FILE';
MASTER_LOG_POS:                 'MASTER_LOG_POS';
MASTER_PASSWORD:                'MASTER_PASSWORD';
MASTER_PORT:                    'MASTER_PORT';
MASTER_RETRY_COUNT:             'MASTER_RETRY_COUNT';
MASTER_SSL:                     'MASTER_SSL';
```

```
MAX_CONNECTIONS_PER_HOUR:           'MAX_CONNECTIONS_PER_HOUR';
MAX_QUERIES_PER_HOUR:               'MAX_QUERIES_PER_HOUR';
MAX_ROWS:                           'MAX_ROWS';
MAX_SIZE:                           'MAX_SIZE';
MAX_UPDATES_PER_HOUR:               'MAX_UPDATES_PER_HOUR';
MAX_USER_CONNECTIONS:               'MAX_USER_CONNECTIONS';
MEDIUM:                             'MEDIUM';
MERGE:                              'MERGE';
MID:                                'MID';
MIGRATE:                            'MIGRATE';
MIN_ROWS:                           'MIN_ROWS';
MODE:                               'MODE';
MODIFY:                             'MODIFY';
MUTEX:                              'MUTEX';
MYSQL:                              'MYSQL';
NAME:                               'NAME';
NAMES:                              'NAMES';
NCHAR:                              'NCHAR';
NEVER:                              'NEVER';
NEXT:                               'NEXT';
NO:                                 'NO';
NODEGROUP:                          'NODEGROUP';
NONE:                               'NONE';
OFFLINE:                            'OFFLINE';
OFFSET:                             'OFFSET';
OJ:                                 'OJ';
OLD_PASSWORD:                       'OLD_PASSWORD';
ONE:                                'ONE';
ONLINE:                             'ONLINE';
ONLY:                               'ONLY';
OPEN:                               'OPEN';
OPTIMIZER_COSTS:                    'OPTIMIZER_COSTS';
OPTIONS:                            'OPTIONS';
OWNER:                              'OWNER';
PACK_KEYS:                          'PACK_KEYS';
PAGE:                               'PAGE';
PARSER:                             'PARSER';
PARTIAL:                            'PARTIAL';
PARTITIONING:                       'PARTITIONING';
PARTITIONS:                         'PARTITIONS';
PASSWORD:                           'PASSWORD';
PHASE:                              'PHASE';
PLUGIN:                             'PLUGIN';
PLUGIN_DIR:                         'PLUGIN_DIR';
PLUGINS:                            'PLUGINS';
PORT:                               'PORT';
PRECEDES:                           'PRECEDES';
PREPARE:                            'PREPARE';
PRESERVE:                           'PRESERVE';
PREV:                               'PREV';
PROCESSLIST:                        'PROCESSLIST';
PROFILE:                            'PROFILE';
PROFILES:                           'PROFILES';
PROXY:                              'PROXY';
QUERY:                              'QUERY';
QUICK:                              'QUICK';
REBUILD:                            'REBUILD';
RECOVER:                            'RECOVER';
REDO_BUFFER_SIZE:                   'REDO_BUFFER_SIZE';
REDUNDANT:                          'REDUNDANT';
RELAY:                              'RELAY';
RELAY_LOG_FILE:                     'RELAY_LOG_FILE';
RELAY_LOG_POS:                      'RELAY_LOG_POS';
```

```
RELAYLOG:                       'RELAYLOG';
REMOVE:                         'REMOVE';
REORGANIZE:                     'REORGANIZE';
RESET:                          'RESET';
RESUME:                         'RESUME';
RETURNS:                        'RETURNS';
ROLLBACK:                       'ROLLBACK';
ROLLUP:                         'ROLLUP';
ROTATE:                         'ROTATE';
ROW:                            'ROW';
ROWS:                           'ROWS';
ROW_FORMAT:                     'ROW_FORMAT';
SAVEPOINT:                      'SAVEPOINT';
SCHEDULE:                       'SCHEDULE';
SECURITY:                       'SECURITY';
SERVER:                         'SERVER';
SESSION:                        'SESSION';
SHARE:                          'SHARE';
SHARED:                         'SHARED';
SIGNED:                         'SIGNED';
SIMPLE:                         'SIMPLE';
SLAVE:                          'SLAVE';
SLOW:                           'SLOW';
SNAPSHOT:                       'SNAPSHOT';
SOCKET:                         'SOCKET';
SOME:                           'SOME';
SONAME:                         'SONAME';
SOUNDS:                         'SOUNDS';
SOURCE:                         'SOURCE';
SQL_AFTER_GTIDS:                'SQL_AFTER_GTIDS';
SQL_AFTER_MTS_GAPS:             'SQL_AFTER_MTS_GAPS';
SQL_BEFORE_GTIDS:               'SQL_BEFORE_GTIDS';
SQL_BUFFER_RESULT:              'SQL_BUFFER_RESULT';
SQL_CACHE:                      'SQL_CACHE';
SQL_NO_CACHE:                   'SQL_NO_CACHE';
SQL_THREAD:                     'SQL_THREAD';
START:                          'START';
STARTS:                         'STARTS';
STATS_AUTO_RECALC:              'STATS_AUTO_RECALC';
STATS_PERSISTENT:               'STATS_PERSISTENT';
STATS_SAMPLE_PAGES:             'STATS_SAMPLE_PAGES';
STATUS:                         'STATUS';
STOP:                           'STOP';
STORAGE:                        'STORAGE';
STORED:                         'STORED';
STRING:                         'STRING';
SUBJECT:                        'SUBJECT';
SUBPARTITION:                   'SUBPARTITION';
SUBPARTITIONS:                  'SUBPARTITIONS';
SUSPEND:                        'SUSPEND';
SWAPS:                          'SWAPS';
SWITCHES:                       'SWITCHES';
TABLESPACE:                     'TABLESPACE';
TEMPORARY:                      'TEMPORARY';
TEMPTABLE:                      'TEMPTABLE';
THAN:                           'THAN';
TRADITIONAL:                    'TRADITIONAL';
TRANSACTION:                    'TRANSACTION';
TRIGGERS:                       'TRIGGERS';
TRUNCATE:                       'TRUNCATE';
UNDEFINED:                      'UNDEFINED';
UNDOFILE:                       'UNDOFILE';
UNDO_BUFFER_SIZE:               'UNDO_BUFFER_SIZE';
```

```
UNINSTALL:                              'UNINSTALL';
UNKNOWN:                                'UNKNOWN';
UNTIL:                                  'UNTIL';
UPGRADE:                                'UPGRADE';
USER:                                   'USER';
USE_FRM:                                'USE_FRM';
USER_RESOURCES:                         'USER_RESOURCES';
VALIDATION:                             'VALIDATION';
VALUE:                                  'VALUE';
VARIABLES:                              'VARIABLES';
VIEW:                                   'VIEW';
VIRTUAL:                                'VIRTUAL';
WAIT:                                   'WAIT';
WARNINGS:                               'WARNINGS';
WITHOUT:                                'WITHOUT';
WORK:                                   'WORK';

// Group function Keywords

AVG:                                    'AVG';
BIT_AND:                                'BIT_AND';
BIT_OR:                                 'BIT_OR';
BIT_XOR:                                'BIT_XOR';
COUNT:                                  'COUNT';
GROUP_CONCAT:                           'GROUP_CONCAT';
MAX:                                    'MAX';
MIN:                                    'MIN';
STD:                                    'STD';
STDDEV:                                 'STDDEV';
STDDEV_POP:                             'STDDEV_POP';
STDDEV_SAMP:                            'STDDEV_SAMP';
SUM:                                    'SUM';
VAR_POP:                                'VAR_POP';
VAR_SAMP:                               'VAR_SAMP';
VARIANCE:                               'VARIANCE';


// Operators
// Operators. Assigns

VAR_ASSIGN:                             ':=';
PLUS_ASSIGN:                            '+=';
MINUS_ASSIGN:                           '-=';
MULT_ASSIGN:                            '*=';
DIV_ASSIGN:                             '/=';
MOD_ASSIGN:                             '%=';
AND_ASSIGN:                             '&=';
XOR_ASSIGN:                             '^=';
OR_ASSIGN:                              '|=';


// Operators. Arithmetics

STAR:                                   '*';
DIVIDE:                                 '/';
MODULE:                                 '%';
PLUS:                                   '+';
MINUSMINUS:                             '--';
MINUS:                                  '-';
DIV:                                    'DIV';
MOD:                                    'MOD';
```

```
    // Operators. Comparation

    EQUAL_SYMBOL:                   '=';
    GREATER_SYMBOL:                 '>';
    LESS_SYMBOL:                    '<';
    EXCLAMATION_SYMBOL:             '!';


    // Operators. Bit

    BIT_NOT_OP:                     '~';
    BIT_OR_OP:                      '|';
    BIT_AND_OP:                     '&';
    BIT_XOR_OP:                     '^';


    // Constructors symbols

    DOT:                            '.';
    LR_BRACKET:                     '(';
    RR_BRACKET:                     ')';
    COMMA:                          ',';
    SEMI:                           ';';
    AT_SIGN:                        '@';
    ZERO_DECIMAL:                   '0';
    ONE_DECIMAL:                    '1';
    TWO_DECIMAL:                    '2';
    SINGLE_QUOTE_SYMB:              '\'';
    DOUBLE_QUOTE_SYMB:              '"';
    REVERSE_QUOTE_SYMB:             '`';
    COLON_SYMB:                     ':';


    // Literal Primitives


    START_NATIONAL_STRING_LITERAL:  'N' SQUOTA_STRING;
    STRING_LITERAL:                 DQUOTA_STRING | SQUOTA_STRING;
    DECIMAL_LITERAL:                DEC_DIGIT+;
    HEXADECIMAL_LITERAL:            'X' '\'' (HEX_DIGIT HEX_DIGIT)+ '\''
                                    | '0X' HEX_DIGIT+;

    REAL_LITERAL:                   (DEC_DIGIT+)? '.' DEC_DIGIT+
                                    | DEC_DIGIT+ '.' EXPONENT_NUM_PART
                                    |    (DEC_DIGIT+)?    '.'    (DEC_DIGIT+
EXPONENT_NUM_PART)
                                    | DEC_DIGIT+ EXPONENT_NUM_PART;
    NULL_SPEC_LITERAL:              '\\' 'N';
    BIT_STRING:                     BIT_STRING_L;
    STRING_CHARSET_NAME:            '_' CHARSET_NAME;

    // Identifiers

    ID:                             ID_LITERAL;
    // DOUBLE_QUOTE_ID:             '"' ~'"'+ '"';
    REVERSE_QUOTE_ID:               '`' ~'`'+ '`';
    STRING_USER_NAME:               (
                                      SQUOTA_STRING | DQUOTA_STRING
                                      | BQUOTA_STRING | ID_LITERAL
                                    ) '@'
                                    (
                                      SQUOTA_STRING | DQUOTA_STRING
                                      | BQUOTA_STRING | ID_LITERAL
```

```
                                        );
    LOCAL_ID:                           '@'
                                    (
                                      [A-Z0-9._$]+
                                      | SQUOTA_STRING
                                      | DQUOTA_STRING
                                      | BQUOTA_STRING
                                    );
    GLOBAL_ID:                          '@' '@'
                                    (
                                      [A-Z0-9._$]+
                                      | BQUOTA_STRING
                                    );


    // Fragments for Literal primitives

    fragment EXPONENT_NUM_PART:         'E' [-+]? DEC_DIGIT+;
    fragment ID_LITERAL:                [A-Z_$0-9]*?[A-Z_$]+?[A-Z_$0-9]*;
    fragment DQUOTA_STRING:             '"' ( '\\'. | '""' | ~('"'| '\\') )* '"';
    fragment SQUOTA_STRING:              '\'' ('\\'. | '\'\'' | ~('\'' | '\\'))*
'\'';
    fragment BQUOTA_STRING:             '`' ( '\\'. | '``' | ~('`'|'\\'))* '`';
    fragment HEX_DIGIT:                 [0-9A-F];
    fragment DEC_DIGIT:                 [0-9];
    fragment BIT_STRING_L:              'B' '\'' [01]+ '\'';
```

Обробник запитів маніпуляції з данними WorkWithTableData

```
    package work;

    import grammar.parserClasses.MySqlParser;
    import grammar.parserClasses.MySqlParserBaseListener;

    import java.util.ArrayList;
    import java.util.List;

    public class WorkWithTableData extends MySqlParserBaseListener {
        private String query;
        public String getQuery(){
            return this.query;
        }
        private List<String> logicalOp=new ArrayList<>();
        private List<String> binComp=new ArrayList<>();
        private List<String> updatedEl=new ArrayList<>();

        @Override
        public void enterInsertStatement(MySqlParser.InsertStatementContext ctx) {


        }

        @Override
        public void exitInsertStatement(MySqlParser.InsertStatementContext ctx) {
            List<MySqlParser.UidContext> a1=ctx.columns.uid();
            List<MySqlParser.ExpressionOrDefaultContext>
a2=ctx.insertStatementValue().expressionsWithDefaults().expressionOrDefault();
            query="db."+ctx.tableName().getText().toLowerCase()+".insert({";
            if(a1.size()!=a2.size()){
                throw    new       IllegalArgumentException("Wrong    number    of
columns/values");
            }
            for(int i=0;i<a1.size();i++){
```

```
                query=query+"\""+a1.get(i).getText().toLowerCase()+"\"           :
\""+a2.get(i).getText().toLowerCase();
                if(i+1!=a1.size()){
                    query=query+" \", ";
                }
            }
            query=query+"})";
        }



        @Override
        public void enterUpdatedElement(MySqlParser.UpdatedElementContext ctx) {
        }

        @Override
        public void exitUpdatedElement(MySqlParser.UpdatedElementContext ctx) {
            updatedEl.add(ctx.fullColumnName().getText()+":
\""+ctx.expression().getText()+"\"");
        }


        @Override
        public                                                              void
enterSingleDeleteStatement(MySqlParser.SingleDeleteStatementContext ctx) {

        }

        @Override
        public                                                              void
exitSingleDeleteStatement(MySqlParser.SingleDeleteStatementContext ctx) {
            query="db."+ctx.tableName().getText()+".remove({";
            if(binComp.size()==0){
                throw new IllegalArgumentException("No where causion");
            }
            for (int i=0;i<logicalOp.size()+1;i++){
                query=query+binComp.get(i);
                if(i!=logicalOp.size()){
                    query=query+" , ";
                }
            }
            query=query+"})";
        }



        @Override
        public                                                              void
enterSingleUpdateStatement(MySqlParser.SingleUpdateStatementContext ctx) {

        }

        @Override
        public                                                              void
exitSingleUpdateStatement(MySqlParser.SingleUpdateStatementContext ctx) {
            query="db."+ctx.tableName().getText()+".update({";
            if(binComp.size()==0){
                throw new IllegalArgumentException("No where causion");
            }
            for (int i=0;i<logicalOp.size()+1;i++){
                query=query+binComp.get(i);
                if(i!=logicalOp.size()){
                    query=query+" , ";
```

```
                }
            }
            query=query+"}, {$set: {";
            for (int i=0;i<updatedEl.size();i++){
                query=query+updatedEl.get(i);
                if(i+1!=updatedEl.size()){
                    query=query+" , ";
                }
            }
            query=query+"}})";
        }

        @Override
        public                                                                  void
exitBinaryComparasionPredicate(MySqlParser.BinaryComparasionPredicateContext ctx) {
            if("=".equals(ctx.comparisonOperator().getText())){
                binComp.add(ctx.left.getText()+": \""+ctx.right.getText()+"\"");
            }
            if(">".equals(ctx.comparisonOperator().getText())){
                binComp.add(ctx.left.getText()+":              {              $gt:
"+ctx.right.getText()+"}");
            }
            if("<".equals(ctx.comparisonOperator().getText())){
                binComp.add(ctx.left.getText()+":              {              $lt:
"+ctx.right.getText()+"}");
            }
        }


        @Override
        public void enterLogicalOperator(MySqlParser.LogicalOperatorContext ctx) {
            logicalOp.add(ctx.getText());
        }

        @Override
        public void exitLogicalOperator(MySqlParser.LogicalOperatorContext ctx) {

        }


    }
```

Клас інтерфейсу користувача MainWindow

```
package work;

import grammar.parserClasses.MySqlLexer;
import grammar.parserClasses.MySqlParser;
import grammar.parserClasses.Translator;
import org.antlr.v4.runtime.CharStreams;
import org.antlr.v4.runtime.CommonTokenStream;
import org.antlr.v4.runtime.tree.ParseTree;
import org.antlr.v4.runtime.tree.ParseTreeWalker;

import javax.swing.*;
import javax.swing.event.MenuEvent;
import javax.swing.event.MenuListener;
import javax.swing.filechooser.FileFilter;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;
import java.nio.charset.StandardCharsets;
```

```
import java.util.Scanner;

public class MainWindow {
    private JPanel panel1;
    private JTextArea textArea1;
    private JButton Button;
    private JTextArea textArea2;

    public MainWindow() {
        JFrame frame = new JFrame("Test frame");
        frame.setContentPane(panel1);
        frame.validate();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JMenuBar menuBar = new JMenuBar();
        JMenu fileMenu = new JMenu("File");

        JMenuItem openItem = new JMenuItem("Відкрити файл");
        fileMenu.add(openItem);
        fileMenu.addSeparator();
        JMenuItem saveItem = new JMenuItem("Зберегти файл");
        fileMenu.add(saveItem);
        fileMenu.addSeparator();
        JMenuItem exitItem = new JMenuItem("Вихід");
        fileMenu.add(exitItem);

        openItem.addActionListener(actionEvent -> {
            JFileChooser fileopen = new JFileChooser();
            fileopen.addChoosableFileFilter(new FileFilter() {
                @Override
                public boolean accept(File file) {
                    String filename = file.getName();
                    int i = filename.lastIndexOf('.');
                    if ( i>0 && i<filename.length()-1 ) {
                        return
"sql".equals(filename.substring(i+1).toLowerCase());
                    }
                    return false;
                }

                @Override
                public String getDescription() {
                    return null;
                }
            });
            fileopen.setAcceptAllFileFilterUsed(true);
            int ret = fileopen.showDialog(null, "Открыть файл");
            if (ret == JFileChooser.APPROVE_OPTION) {
                File file = fileopen.getSelectedFile();
                try {
                    FileReader fileReader=new FileReader(file);
                    Scanner scanner = new Scanner(fileReader);
                    textArea1.setText("");
                    while (scanner.hasNextLine()){
                        textArea1.append(scanner.nextLine()+"\n");
                    }
                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                }
            }
        });
        saveItem.addActionListener(actionEvent -> {
            JFileChooser fc = new JFileChooser();
            fc.setSelectedFile(new File("New File.js"));
```

```
                if (fc.showSaveDialog(null) == JFileChooser.APPROVE_OPTION) {
                    try {
                        FileOutputStream fileStream = new
FileOutputStream(fc.getSelectedFile());
                        OutputStreamWriter outputStreamWriter = new
OutputStreamWriter(fileStream, StandardCharsets.UTF_8);
                        outputStreamWriter.write(textArea2.getText());
                        outputStreamWriter.close();
                        fileStream.close();
                    }
                    catch (Exception e) {
                        System.out.println("Что-то пошло не так...");
                    }
                }

            });
            openItem.addActionListener(actionEvent -> {

            });

            JMenu newMenu = new JMenu("Edit");
            newMenu.addMenuListener(new MenuListener() {
                @Override
                public void menuSelected(MenuEvent e) {
                    JOptionPane.showMessageDialog(null,
                            "Тут будет выполнятся возможноть редактировать
проект");
                }

                @Override
                public void menuDeselected(MenuEvent e) {

                }

                @Override
                public void menuCanceled(MenuEvent e) {

                }
            });
            JMenu helpMenu = new JMenu("Help");
            helpMenu.addMenuListener(new MenuListener() {
                @Override
                public void menuSelected(MenuEvent e) {
                    JOptionPane.showMessageDialog(null,
                            "Тут будет выполнятся вызов справки");
                }

                @Override
                public void menuDeselected(MenuEvent e) {

                }

                @Override
                public void menuCanceled(MenuEvent e) {

                }
            });



            menuBar.add(fileMenu);
            menuBar.add(newMenu);
            menuBar.add(helpMenu);
```

```
            frame.setJMenuBar(menuBar);

            frame.pack();
            frame.setLocationRelativeTo(null);
            frame.setVisible(true);
            Button.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    textArea2.setText(new
Translator(textArea1.getText().toUpperCase()).getResult().toUpperCase());
                }
            });
        }
    }
```

Клас тестів програми

```
package work;

import grammar.parserClasses.MySqlLexer;
import grammar.parserClasses.MySqlParser;
import org.antlr.v4.runtime.CharStreams;
import org.antlr.v4.runtime.CommonTokenStream;
import org.antlr.v4.runtime.tree.ParseTree;
import org.antlr.v4.runtime.tree.ParseTreeWalker;
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;

public class WorkWithTableDataTest {
    MySqlLexer lexer;
    CommonTokenStream tokens;
    MySqlParser parser;
    ParseTree tree;
    ParseTreeWalker walker;
    WorkWithTableData workWithTableData;
    @Before
    public void setUp(){

    }
    @Test
    public void insertRightTest(){
         lexer = new MySqlLexer(CharStreams.fromString("INSERT INTO USERS (NAME)
VALUES ('OLEG');"));
         tokens = new CommonTokenStream(lexer);
         parser = new MySqlParser(tokens);
         tree = parser.root();
         walker = new ParseTreeWalker();
         workWithTableData =new WorkWithTableData();

         walker.walk(workWithTableData, tree);

        String result = workWithTableData.getQuery();
        assertEquals("db.users.insert({\"name\" : \"'oleg'})",result);

        lexer=new MySqlLexer(CharStreams.fromString("INSERT INTO USERS (NAME,AGE)
VALUES ('OLEG',20);"));
        tokens = new CommonTokenStream(lexer);
        parser = new MySqlParser(tokens);
        tree = parser.root();
        walker = new ParseTreeWalker();
        workWithTableData =new WorkWithTableData();
```

```
        walker.walk(workWithTableData, tree);

        result = workWithTableData.getQuery();
        assertEquals("db.users.insert({\"name\"   :   \"'oleg'   \",   \"age\"   :
\"20})",result);
    }

    @Test
    public void updateRight(){
        lexer  =  new  MySqlLexer(CharStreams.fromString("UPDATE  USERS  SET  NAME  =
'SASHA' WHERE NAME='OLEG'"));
        tokens = new CommonTokenStream(lexer);
        parser = new MySqlParser(tokens);
        tree = parser.root();
        walker = new ParseTreeWalker();
        workWithTableData =new WorkWithTableData();

        walker.walk(workWithTableData, tree);

        String result = workWithTableData.getQuery();
        assertEquals("db.USERS.update({NAME:     \"OLEG\"},     {$set:     {NAME:
\"SASHA\"}})",result);

        lexer=new MySqlLexer(CharStreams.fromString("UPDATE USERS SET NAME = 'SASHA'
WHERE NAME='OLEG' AND AGE>20"));
        tokens = new CommonTokenStream(lexer);
        parser = new MySqlParser(tokens);
        tree = parser.root();
        walker = new ParseTreeWalker();
        workWithTableData =new WorkWithTableData();

        walker.walk(workWithTableData, tree);

        result = workWithTableData.getQuery();
        assertEquals("db.USERS.update({NAME: \"OLEG\" , AGE: { $gt: 20}}, {$set:
{NAME: \"SASHA\"}})",result);
    }

    @Test
    public void deleteRight(){
        lexer = new MySqlLexer(CharStreams.fromString("DELETE FROM USERS WHERE NAME
= 'OLEG'"));
        tokens = new CommonTokenStream(lexer);
        parser = new MySqlParser(tokens);
        tree = parser.root();
        walker = new ParseTreeWalker();
        workWithTableData =new WorkWithTableData();

        walker.walk(workWithTableData, tree);

        String result = workWithTableData.getQuery();
        assertEquals("db.USERS.remove({NAME: \"OLEG\"})",result);

        lexer=new MySqlLexer(CharStreams.fromString("DELETE FROM USERS WHERE NAME =
'OLEG' AND AGE=20"));
        tokens = new CommonTokenStream(lexer);
        parser = new MySqlParser(tokens);
        tree = parser.root();
        walker = new ParseTreeWalker();
        workWithTableData =new WorkWithTableData();

        walker.walk(workWithTableData, tree);
```

```
        result = workWithTableData.getQuery();
        assertEquals("db.USERS.remove({NAME: \"OLEG\" , AGE: \"20\"})",result);
    }
    @Test(expected = IllegalArgumentException.class)
    public void insertWrongCount(){
        lexer = new MySqlLexer(CharStreams.fromString("INSERT INTO USERS (NAME,AGE)
VALUES ('OLEG');"));
        tokens = new CommonTokenStream(lexer);
        parser = new MySqlParser(tokens);
        tree = parser.root();
        walker = new ParseTreeWalker();
        workWithTableData =new WorkWithTableData();

        walker.walk(workWithTableData, tree);
    }
    @Test(expected = IllegalArgumentException.class)
    public void deleteWrongCount(){
        lexer = new MySqlLexer(CharStreams.fromString("DELETE FROM USERS"));
        tokens = new CommonTokenStream(lexer);
        parser = new MySqlParser(tokens);
        tree = parser.root();
        walker = new ParseTreeWalker();
        workWithTableData =new WorkWithTableData();

        walker.walk(workWithTableData, tree);
    }
    @Test(expected = IllegalArgumentException.class)
    public void updateWrongCount(){
        lexer = new MySqlLexer(CharStreams.fromString("UPDATE USERS SET NAME =
‘SASHA’"));
        tokens = new CommonTokenStream(lexer);
        parser = new MySqlParser(tokens);
        tree = parser.root();
        walker = new ParseTreeWalker();
        workWithTableData =new WorkWithTableData();

        walker.walk(workWithTableData, tree);
    }
}
```